Documentação de Projeto

para o sistema

HookCI

Versão 1.1.0

Projeto de sistema elaborado pelo(s) aluno(s) Gabriel Dolabela Marques e Henrique Carvalho Almeida e apresentado ao curso de **Engenharia de Software** da **PUC Minas** como parte do Trabalho de Conclusão de Curso (TCC) sob orientação de conteúdo dos professores João Pedro Oliveira Batisteli, Leonardo Vilela Cardoso e Joana Gabriela Ribeiro de Souza, orientação acadêmica do professor Cleiton Silva Tavares e orientação de TCC II do professor Cleiton Silva Tavares.

14/12/2025

Tabela de Conteúdo

Histórico de Revisões

Nome	Data	Razões para Mudança	Versão
Gabriel Dolabela Henrique Almeida	06/04	Criação do documento	0.1.0
Gabriel Dolabela Henrique Almeida	21/04	Adição de Diagramas de Sequência, Classes e Comunicação	0.2.0
Henrique Almeida	23/04	Correções da criação do documento	0.2.1
Gabriel Dolabela Henrique Almeida	06/05	Adição de Arquitetura Lógica, Diagramas de Estados, Diagramas de Componentes, Diagramas de Implantação, do modelo de configuração e do glossário	0.3.0
Gabriel Dolabela	14/05	Correções dos Diagramas de Sequência	0.3.1
Gabriel Dolabela Henrique Almeida	24/05	Adição de Casos de Teste, Cronograma e Processo de Implementação	0.4.0
Henrique Almeida	19/10	Atualização dos diagramas de Sequência e Classes	0.4.1
Gabriel Dolabela Henrique Almeida	22/11	Revisão de diagramas de Sequência, Estados e Classes	0.4.2
Gabriel Dolabela Henrique Almeida	22/11	Adição do Post-mortem e revisão do Processo de Implementação	1.0.0
Gabriel Dolabela Henrique Almeida	14/12	Adição e revisão de itens.	1.1.0

1. Introdução

HookCI consiste em uma ferramenta para a execução da Integração Contínua (CI, do inglês *Continuous Integration*) de forma local, integrando Docker[1] e Git *hooks*[2] para validar *commits* e *pushes*. A proposta do HookCI se distingue das soluções tradicionais de Integração Contínua, como GitHub Actions[3] e GitLab CI[4], ao trazer o processo de CI diretamente para o ambiente local do desenvolvedor. Essa abordagem proporciona um retorno mais rápido durante o desenvolvimento e elimina a dependência e a carga adicional sobre servidores remotos. Com isso, a ferramenta funciona como uma etapa preventiva no fluxo de trabalho, assegurando que apenas códigos previamente validados por testes sejam encaminhados ao repositório central.

A concepção do HookCI originou-se de uma demanda prática vivenciada pelos autores no cotidiano do desenvolvimento de software. A frustração recorrente com o ciclo de *feedback* lento das ferramentas de CI remotas, onde é necessário enviar o código ao repositório e aguardar a execução em servidores externos para só então identificar falhas, motivou a busca por uma solução mais ágil. O objetivo central, portanto, tornou-se eliminar esse gargalo, permitindo a validação imediata e local sem sacrificar a consistência do ambiente.

Embora existam soluções como o Husky[5], que executam comandos diretamente no ambiente local, o HookCI busca avançar esse modelo ao isolar a execução dos testes em contêineres Docker, garantindo maior consistência e reprodutibilidade do ambiente. Além disso, oferece suporte à configuração por meio de arquivos YAML e disponibiliza uma Interface de Linha de Comando (CLI, do inglês Command Line Interface) dedicada, conferindo maior organização, padronização e robustez ao processo de CI local.

Este documento agrega: 1) a elaboração e revisão de modelos de domínio e 2) modelos de projeto para o sistema HookCI. A referência principal para a descrição geral do problema, domínio e requisitos do sistema é este documento de especificação que descreve a visão de domínio do sistema. Tal especificação acompanha este documento. Anexo a este documento também se encontra o Glossário.

2. Modelos de Usuário e Requisitos

Esta seção detalha os perfis de usuários do sistema HookCI, detalha os modelos de casos de uso e histórias de usuários que definem os requisitos e funcionalidades esperadas da ferramenta, e apresenta os Diagramas de Sequência do Sistema e os Contratos de Operações.

---

[1] https://www.docker.com/
[2] https://git-scm.com/
[3] https://github.com/features/actions
[4] https://docs.gitlab.com/
[5] https://huskyci.opensource.globo.com/

2.1 Descrição de Atores

Desenvolvedor: É o principal ator, responsável por implementar funcionalidades, escrever código e rodar testes. Usa o HookCI para validar *commits* e *pushes*, para garantir que apenas mudanças aprovadas na suíte de testes sejam integradas ao repositório. O ator aproveita do retorno imediato proporcionado pela rodagem dos testes localmente, usando Docker para isolamento e coesão.

Administrador: Atua na configuração e na manutenção do ambiente de desenvolvimento e dos parâmetros de execução da ferramenta. É responsável por definir, por meio de arquivos YAML, as regras e os comandos de teste a serem utilizados pelo HookCI.

2.2 Modelos de Usuários

Para melhor compreender as necessidades e expectativas dos usuários do HookCI, foram criadas personas que representam os perfis de uso da ferramenta. A seguir, são apresentadas duas personas que englobam os principais perfis de desenvolvedores e administradores envolvidos. A Tabela 1 descreve a persona do usuário Márcio Nunes, um desenvolvedor iniciante que deseja ter um retorno rápido e assertivo relativo à qualidade de seu código. HookCI garante consistência e evita conflitos com as configurações locais.

	Márcio Nunes, Desenvolvedor Júnior
Descrição	Márcio tem 23 anos e está iniciando sua carreira como desenvolvedor. Trabalha em uma pequena equipe de desenvolvimento e precisa de ferramentas que facilitem o aprendizado e ofereçam retorno rápido sobre a qualidade do código.
Objetivos	• Obter respostas imediatas sobre possíveis falhas em seus *commits*, permitindo aprendizado contínuo. • Adquirir confiança na integração de novas funcionalidades sem comprometer a integridade do repositório.
Dores	• Dificuldade em identificar rapidamente os erros decorrentes de testes mal sucedidos. • Processos de CI que, por serem remotos, atrasam seu fluxo de trabalho.

Tabela 1. Persona Márcio

A Tabela 2 descreve a persona do usuário Paulo Lopes, um desenvolvedor administrador de um time que deseja garantir a qualidade de seu projeto a todo momento de forma automatizada e padronizada. HookCI possibilita configuração centralizada via arquivos YAML e facilita a

instalação automatizada dos *hooks* do Git, assegurando que todos os membros de sua equipe trabalhem com o mesmo ambiente de CI, independentemente das particularidades de suas máquinas.

	Paulo Lopes, Desenvolvedor Sênior
Descrição	Paulo tem 29 anos e atua como desenvolvedor sênior em uma equipe de médio porte. Além de codificar, ele também assume funções de administração de times, configurando o ambiente de desenvolvimento e definindo as diretrizes para a integração contínua.
Objetivos	• Estabelecer um fluxo de trabalho que minimize falhas e garanta a qualidade do código entregue. • Otimizar o tempo de execução dos testes, mantendo um ciclo de retorno rápido e eficiente.
Dores	• Dificuldade em padronizar os ambientes de desenvolvimento entre os membros da equipe. • Gerenciar configurações de testes que, quando executados em ambientes distintos, podem apresentar resultados inconsistentes.

Tabela 2. Persona Paulo

2.3 Modelo de Casos de Uso e Histórias de Usuários

Esta subseção apresenta o diagrama de casos de uso e as histórias de usuários que guiam o desenvolvimento da aplicação HookCI.

2.3.1 Diagrama de Casos de Uso

A Figura 1 apresenta o diagrama de Casos de Uso (UC, do inglês *Use Case*) do sistema HookCI, no qual são representados três atores: Desenvolvedor, Administrador e o sistema Git. O Desenvolvedor interage com a documentação e inicia a execução de testes por meio da CLI; o Administrador realiza a inicialização do projeto na ferramenta; e o sistema Git, por sua vez, dispara a execução dos testes automaticamente via Git *hooks*.

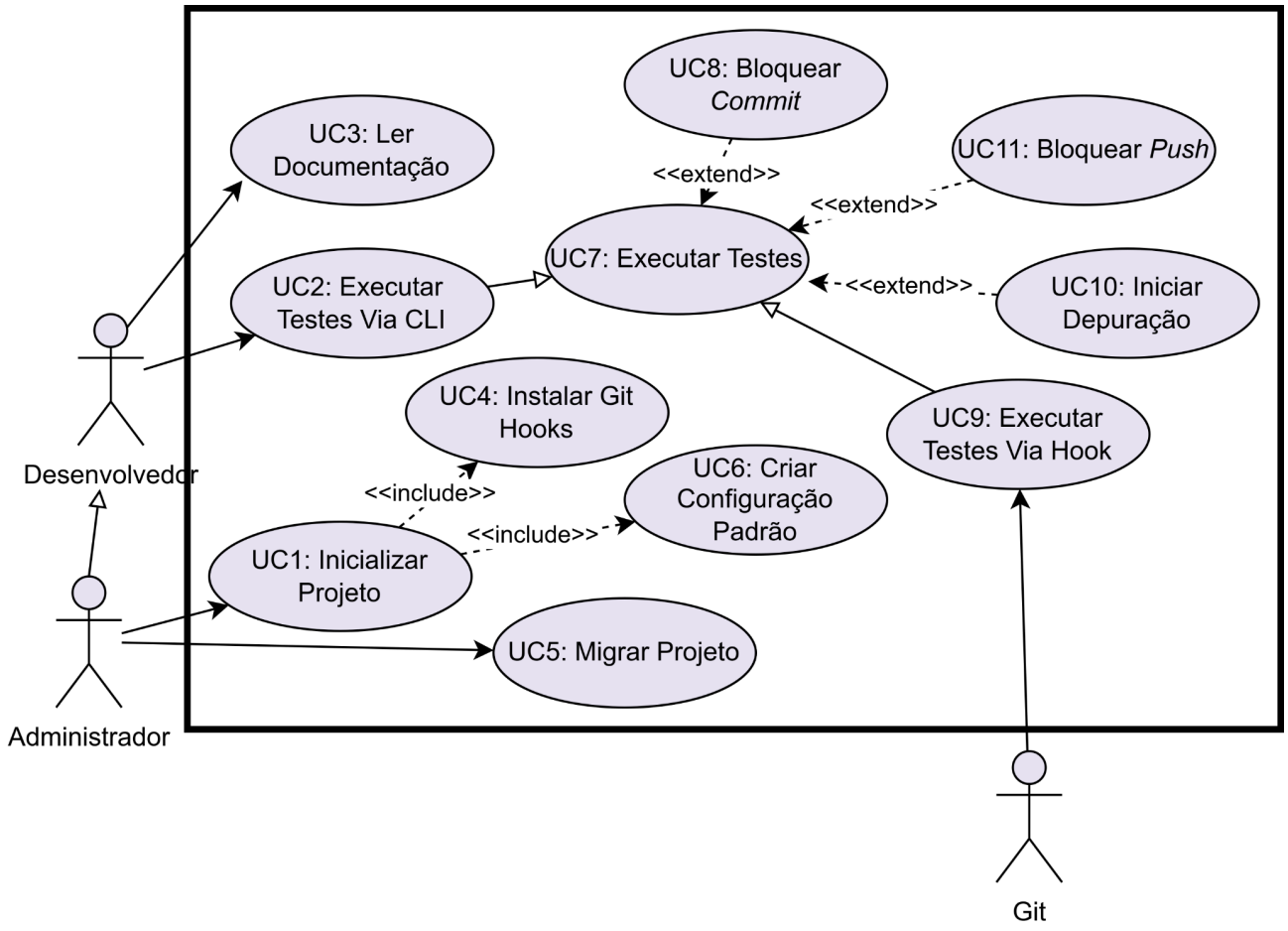

Figura 1. Diagrama de caso de uso

2.3.2 História de Usuários

As histórias de usuários a seguir foram desenvolvidas seguindo uma análise das necessidades e expectativas dos potenciais usuários da ferramenta HookCI. O processo envolve conversas com os desenvolvedores e tem como foco capturar cenários de uso realísticos e funcionalidades necessárias para otimizar o fluxo de trabalho de integração contínua local. O desenvolvimento de cada História de Usuário (US, do inglês *User History*) corresponde a uma funcionalidade ou requisito na visão do usuário.

US01: Como desenvolvedor, desejo acessar a documentação dos comandos disponíveis via CLI, para entender como utilizar a ferramenta corretamente (relativo ao UC3)

US02: Como administrador, desejo inicializar um projeto com o HookCI, para configurar o ambiente local e preparar a estrutura básica de integração contínua. (relativo ao UC1)

US03: Como administrador, desejo que seja criado um arquivo YAML de configuração de forma automática, para definir os parâmetros de execução de testes no projeto. (relativo ao UC1 e UC6)

US04: Como administrador, desejo que os *hooks* do Git sejam instalados automaticamente no projeto, para evitar configurações manuais propensas a erro. (relativo ao UC1 e UC4)

US05: Como administrador, desejo editar o arquivo de configuração YAML, para personalizar comandos, variáveis de ambiente e políticas de execução. (relativo ao UC6)

US06: Como administrador, desejo migrar a configuração pré-existente para a mais atual da ferramenta de forma automática. (relativo ao UC5)

US07: Como administrador, desejo ser alertado pela ferramenta caso a configuração não possa ser automaticamente migrada. (relativo ao UC5)

US08: Como desenvolvedor, desejo que os testes sejam executados automaticamente ao realizar um *commit* ou *push*, para evitar que código defeituoso seja integrado ao repositório. (relativo ao UC7, UC8, UC9 e UC11)

US09: Como desenvolvedor, desejo executar testes manualmente via CLI, para verificar o comportamento do sistema sob demanda. (relativo ao UC2 e UC7)

US10: Como desenvolvedor, desejo que os testes sejam executados em um ambiente Docker isolado, para garantir a consistência entre ambientes de desenvolvimento. (relativo ao UC2, UC9 e UC7)

US11: Como desenvolvedor, desejo visualizar os *logs* da execução de testes na linha de comando em caso de erro, a fim de entender o que foi executado e identificar falhas rapidamente. (relativo ao UC2 e UC7)

US12: Como desenvolvedor, desejo ajustar o nível de verbosidade dos *logs* da CLI, para escolher entre mensagens mais detalhadas ou mais concisas conforme a situação. (relativo ao UC2 e UC7)

US13: Como desenvolvedor, desejo ter acesso a um modo de depuração em caso de falha nos testes, para investigar interativamente o ambiente de execução e entender a origem dos problemas. (relativo ao UC10 e UC7)

US14: Como administrador, desejo validar a estrutura e os valores do arquivo YAML de configuração, para garantir que ele esteja correto antes da execução. (relativo ao UC1, UC5 e UC6)

US15: Como administrador, desejo configurar testes não essenciais que apenas avisem o desenvolvedor caso falhem. (relativo ao UC2, UC7, UC8 e UC11)

US16: Como administrador, desejo configurar uma imagem Docker personalizada através de um nome e versão, para definir o ambiente de execução da CI. (relativo ao UC6)

US17: Como administrador, desejo utilizar um *Dockerfile* para definir o ambiente de testes, para maior controle sobre dependências e ferramentas instaladas. (relativo ao UC6)

US18: Como administrador, desejo montar automaticamente o diretório atual do repositório no contêiner durante a execução dos testes, para garantir que o código-fonte correto seja testado. (relativo ao UC9 e UC7)

US19: Como administrador, desejo configurar filtros para commits e branches, para reduzir escopo de testes. (relativo ao UC6 e UC9)

2.4 Diagrama de Sequência do Sistema e Contrato de Operações

Nesta subseção é apresentado o Diagrama de Sequência do Sistema (DSS) e os Contratos de Operações. A Figura 2 ilustra a interação para exibir a ajuda, contemplando a história de usuário US01. O usuário envia o comando “hookci help” ao Sistema, que processa a solicitação e retorna o texto de ajuda diretamente para o console do usuário. A Tabela 3 detalha o contrato para a operação “Mostrar ajuda”

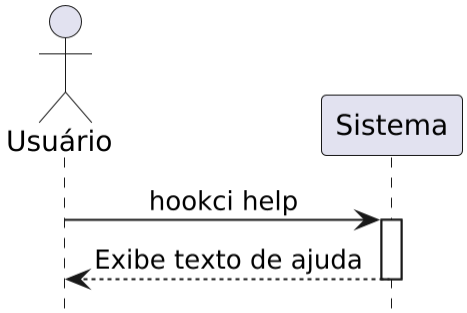

Figura 2. DSS para a operação “Mostrar ajuda”

Contrato	Mostrar ajuda
Operação	display_help()
Pré-condições	Ferramenta instalada no ambiente.
Pós-condições	Texto informativo apresentado no console.

Tabela 3. Contrato da operação “Mostrar ajuda”

A Figura 3 descreve o fluxo para inicializar um projeto, contemplando as histórias de usuário US02, US03, US04. O Administrador aciona o Sistema com o comando “hookci init”. O Sistema, então, executa as etapas internas necessárias, como a criação do arquivo de configuração padrão e a

instalação dos hooks do Git, e informa ao Administrador se a operação foi bem-sucedida ou se o projeto já estava inicializado. A Tabela 4 detalha o contrato para a operação “Inicializar projeto”

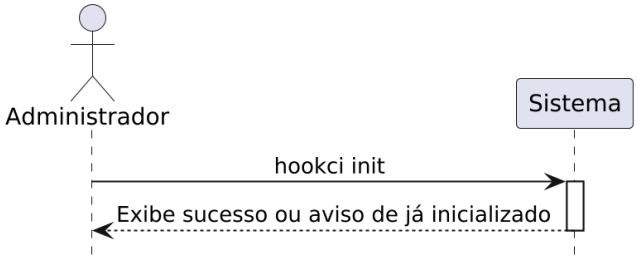

Figura 3. DSS para a operação “Inicializar projeto”

Contrato	Inicializar projeto
Operação	init_project()
Pré-condições	Ferramenta instalada no ambiente, Projeto Git na pasta atual ou mãe, Docker instalado.
Pós-condições	Ferramenta instalada no projeto.

Tabela 4. Contrato da operação “Inicializar projeto”

A Figura 4 apresenta o processo de migração de configuração, contemplando as histórias de usuário US06, US07. O Administrador inicia a operação com o comando “hookci migrate”. O Sistema lê o arquivo de configuração existente, processa a migração para a versão mais recente e, se bem-sucedido, reescreve o arquivo. O Administrador é então informado sobre o resultado da migração. A Tabela 5 detalha o contrato para a operação “Migrar configuração”

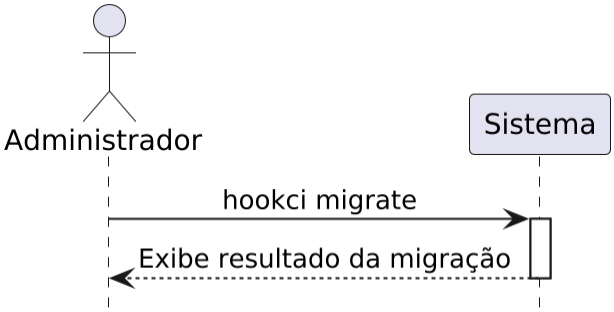

Figura 4. DSS para a operação “Migrar configuração”

Contrato	Migrar configuração
Operação	migrate_config()
Referências cruzadas	Histórias de usuário: US02, US03.
Pré-condições	Ferramenta instalada no ambiente e no projeto, Projeto Git na pasta atual ou mãe.
Pós-condições	Configuração atualizada para sua última versão.

Tabela 5. Contrato da operação “Migrar configuração”

A Figura 5 ilustra a execução da CI disparada automaticamente por um *hook* do Git, contemplando as histórias de usuário US05, US08, US10, US11, US14, US15, US16, US17, US18 e US19. O sistema Git invoca o Sistema HookCI. O Sistema carrega a configuração, prepara o ambiente Docker e executa as etapas de teste. Com base no resultado dos testes, o Sistema retorna um status para o Git, que permitirá ou abortará a operação Git. A Tabela 6 detalha o contrato para a operação "Executar CI via *Hook*"

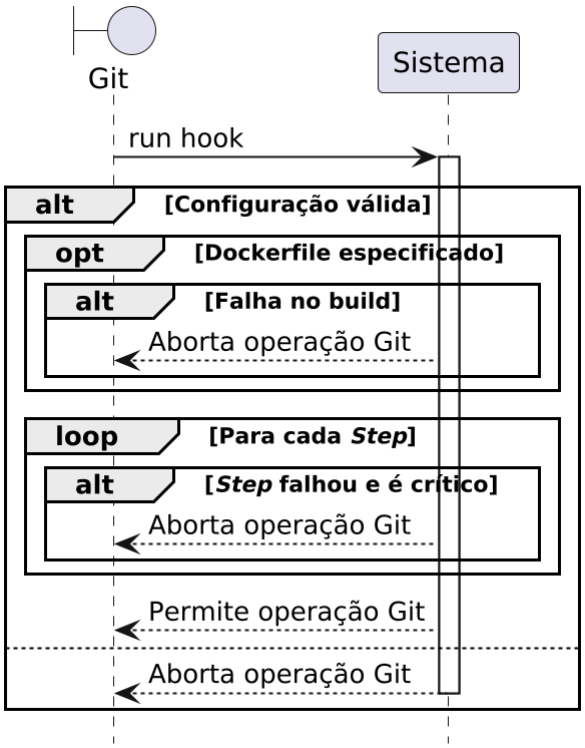

Figura 5. DSS para a operação "Executar CI via *Hook*"

Contrato	Executar CI via *Hook*
Operação	run_hook()
Referências cruzadas	Histórias de usuário: US02, US03, US04.
Pré-condições	Ferramenta instalada no ambiente e no projeto, Projeto Git na pasta atual ou mãe, configuração de projeto desatualizada.
Pós-condições	Operação git permitida ou bloqueada.

Tabela 6. Contrato da operação "Executar CI via *Hook*"

A Figura 6 detalha a execução manual da CI, contemplando as histórias de usuário US05, US09, US10, US11, US12, US13, US14, US15, US16, US17 e US18. O Desenvolvedor aciona o Sistema com o comando "hookci run". O Sistema, de forma similar à execução via hook, valida a configuração, gerencia o ambiente Docker e executa os testes. Ao final, o resultado da execução,

incluindo *logs*, é exibido diretamente no console para o Desenvolvedor. A Tabela 7 detalha o contrato para a operação “Executar CI”

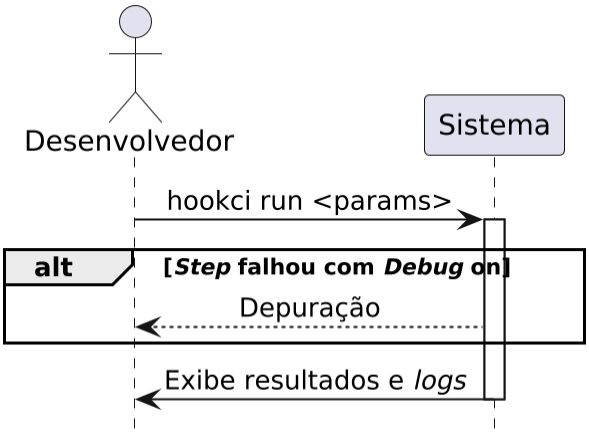

Figura 6. DSS para a operação “Executar CI”

Contrato	Executar CI
Operação	run()
Referências cruzadas	Histórias de usuário: US02, US03, US04.
Pré-condições	Ferramenta instalada no ambiente e no projeto, Projeto Git na pasta atual ou mãe, configuração de projeto desatualizada.
Pós-condições	Resultados e *logs* apresentados.

Tabela 7. Contrato da operação “Executar CI”

3. Modelos de Projeto

Esta seção estabelece o Projeto e a Arquitetura do sistema HookCI. A partir dos modelos e requisitos da Seção 2, os Modelos de Projeto aqui detalhados, incluindo Diagramas de Classes, de Sequência, de Comunicação e a Arquitetura Lógica, fornecem a visão estática e dinâmica para a implementação do software.

3.1 Diagrama de Classes

A Figura 7 detalha a estrutura estática do sistema HookCI. As classes são organizadas em camadas arquiteturais: *Presentation*, responsável pela CLI; *Application*, que orquestra os serviços e as histórias de usuário; *Domain*, contendo a lógica de negócio central e entidades como Configuration e TestStep; e *Infrastructure*, que gerencia interações externas com o sistema Git, a plataforma Docker e o sistema de arquivos. O diagrama ilustra as classes principais, suas interfaces e os relacionamentos entre elas, como herança, composição e dependência. Este modelo serve como um mapa dos componentes de software e de como eles se conectam para formar o sistema.

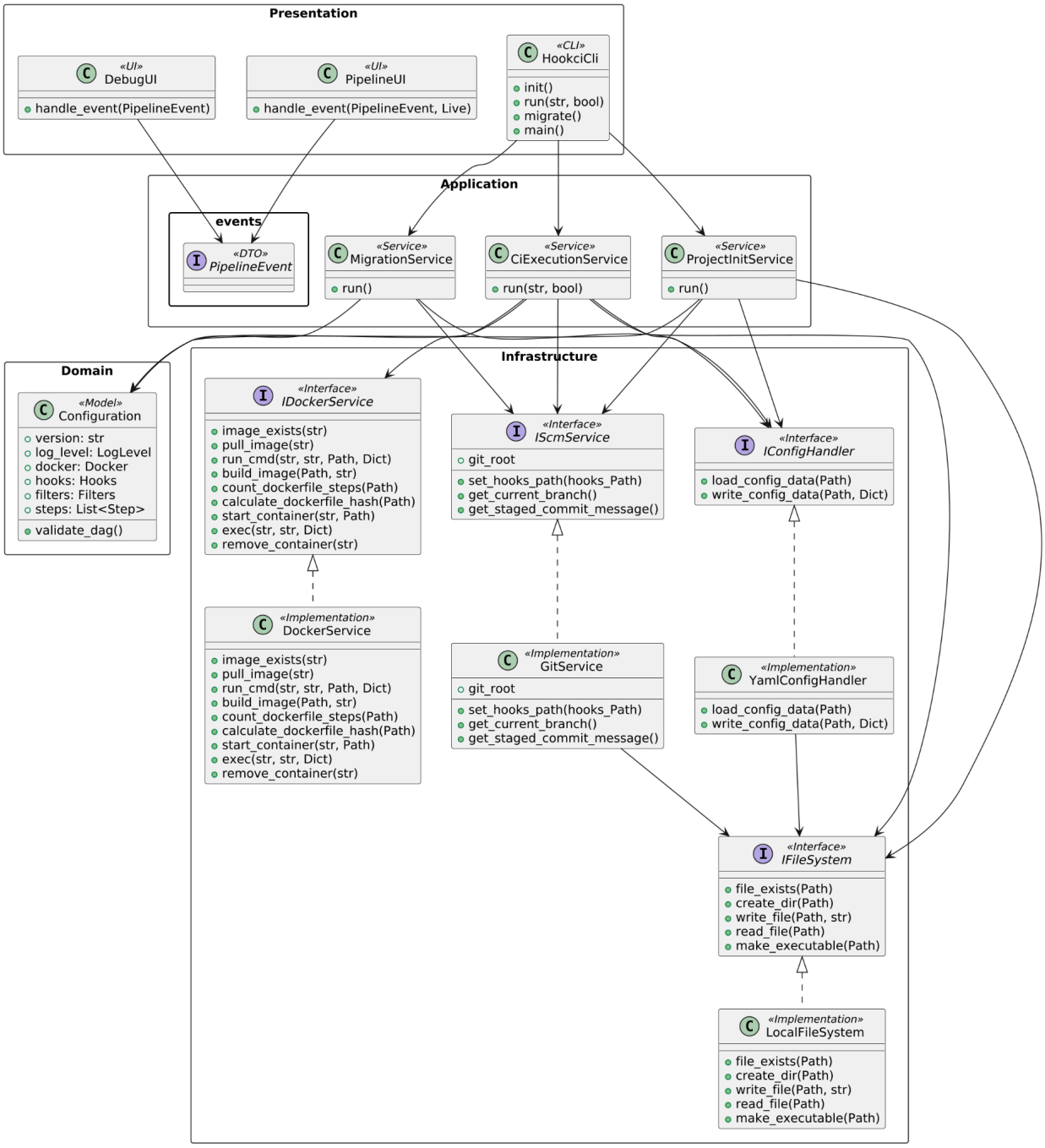

Figura 7. Diagrama de Classes do sistema

3.2 Diagramas de Sequência

A Figura 8 descreve a sequência para exibir a ajuda ao usuário. O Usuário executa o comando "hookci help" na CLI. A camada de *Presentation* processa esta solicitação e responde diretamente ao Usuário, exibindo o texto de ajuda formatado.

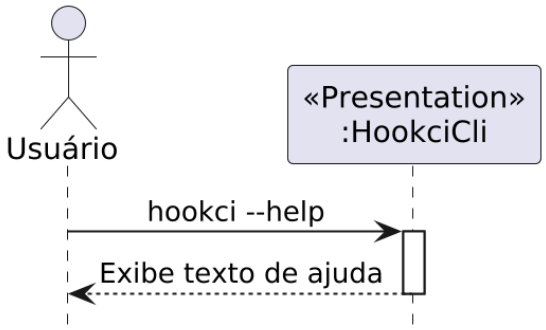

Figura 8. Diagrama de Sequência para a operação “Mostrar ajuda”

A Figura 9 ilustra o fluxo de inicialização do projeto, acionado pelo comando “hookci init” executado pelo Administrador. A CLI encaminha a requisição para a camada de *Application*. Esta camada coordena com a camada de *Infrastructure* para realizar duas ações principais: primeiro, localizar a raiz do repositório Git e, caso o projeto ainda não tenha sido inicializado, criar o arquivo de configuração padrão; segundo, instalar os *scripts* de *hook* necessários no diretório .git/ do repositório. O resultado da operação, sucesso ou um aviso caso o projeto já esteja inicializado, é então comunicado ao Administrador pela CLI.

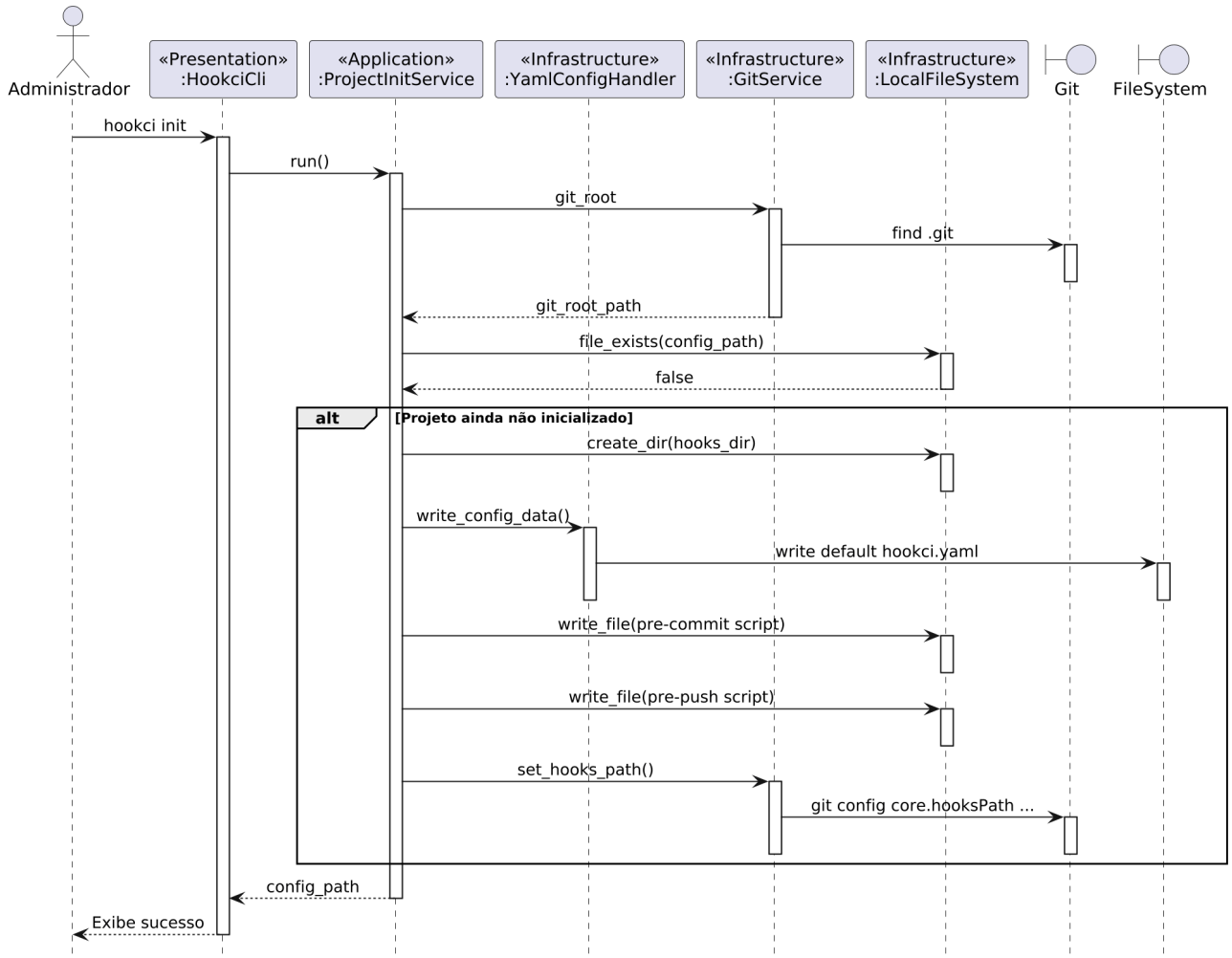

Figura 9. Diagrama de Sequência para a operação “Inicializar projeto”

A Figura 10 detalha o processo de migração da configuração, iniciado pelo Administrador com o comando “hookci migrate”. A camada de *Application* solicita à camada de *Infrastructure* a leitura do arquivo de configuração existente. Em seguida, a própria camada de *Application* processa os dados lidos para convertê-los ao formato mais recente. Se a migração for possível e bem-sucedida, a *Application* instrui a *Infrastructure* a escrever o novo arquivo de configuração. Por fim, a CLI informa o Administrador sobre o sucesso ou eventuais problemas na migração.

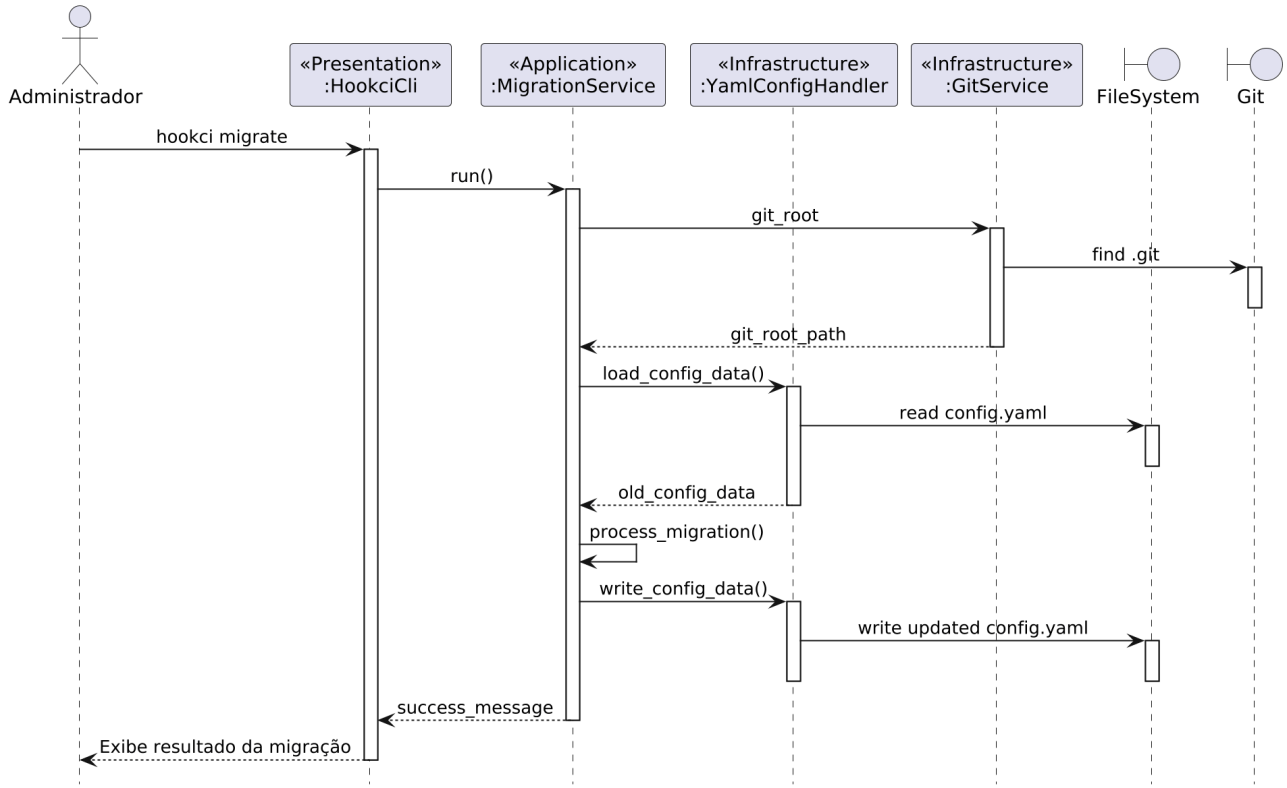

Figura 10. Diagrama de Sequência para a operação “Migrar configuração”

A Figura 11 demonstra a execução automática da CI, disparada por um *hook* do sistema Git, como o *pre-commit*. O Git executa o *script* do *hook*, que por sua vez invoca a CLI do HookCI. A camada de *Application* é acionada e, interagindo com as camadas de *Domain* e *Infrastructure*, carrega e valida a configuração do arquivo YAML; obtém as informações sobre o ambiente Docker, seja uma imagem pré-existente ou um *Dockerfile* a ser construído; executa sequencialmente cada etapa de teste definida na configuração dentro de contêineres Docker isolados. Ao final, a CLI retorna um código de saída para o Git: 0 indica sucesso, permitindo que a operação Git continue; um código diferente de 0 indica falha em uma etapa crítica, fazendo com que a operação Git seja abortada.

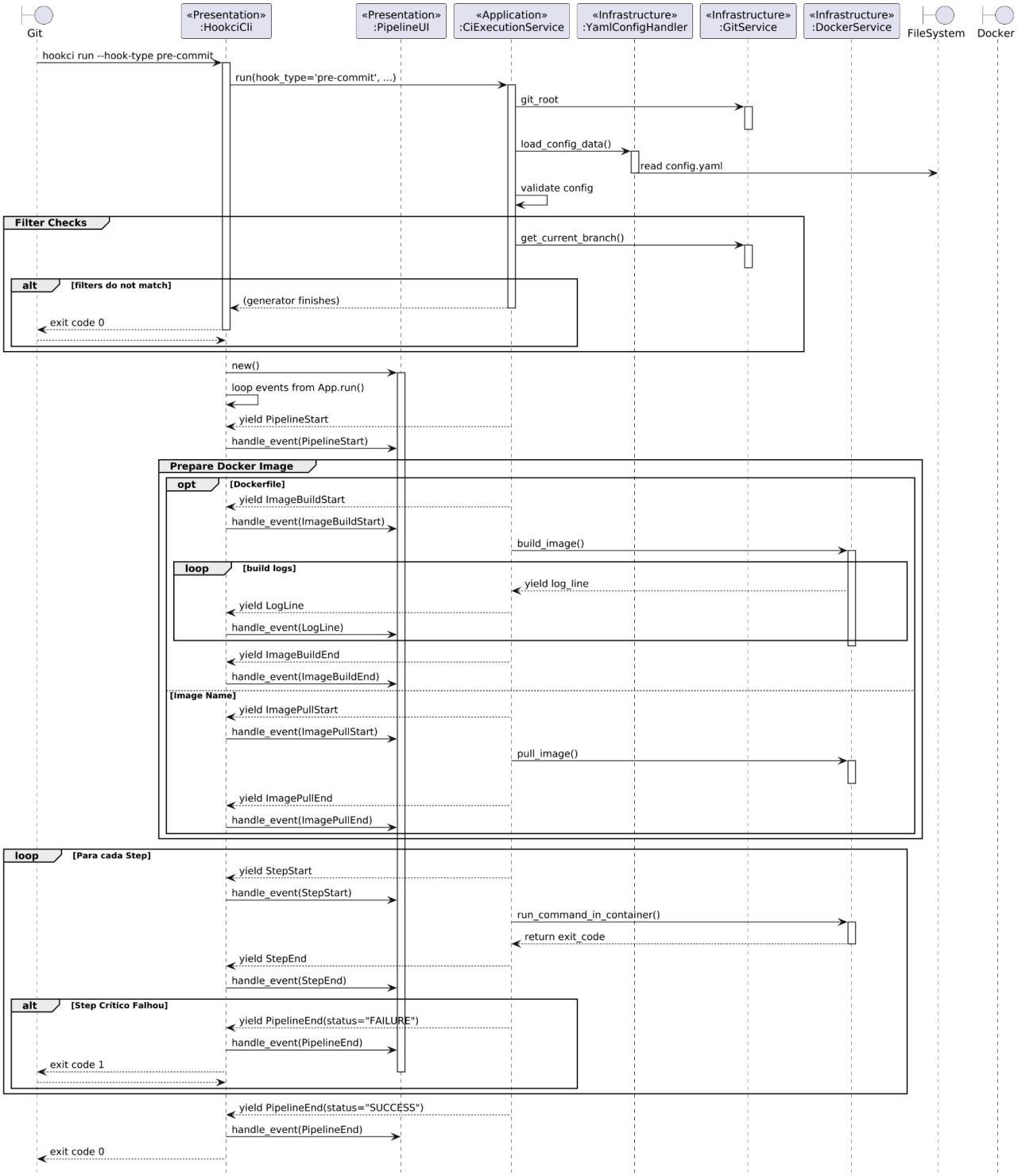

Figura 11. Diagrama de Sequência para a operação “Executar CI via *Hook*”

A Figura 12 representa a execução manual da CI, iniciada pelo Desenvolvedor através do comando “hookci run” na CLI. O fluxo de execução das etapas de teste é semelhante ao disparado pelo *hook* Git, envolvendo as camadas de *Application*, *Domain* e *Infrastructure* para validar a configuração,

gerenciar o ambiente Docker e executar os testes em contêineres. A principal diferença reside no início e fim da interação: ela é explicitamente solicitada pelo Desenvolvedor e o resultado final, incluindo *logs* e o status de sucesso ou falha, é exibido diretamente no terminal para o Desenvolvedor, sem interferir no fluxo de uma operação Git.

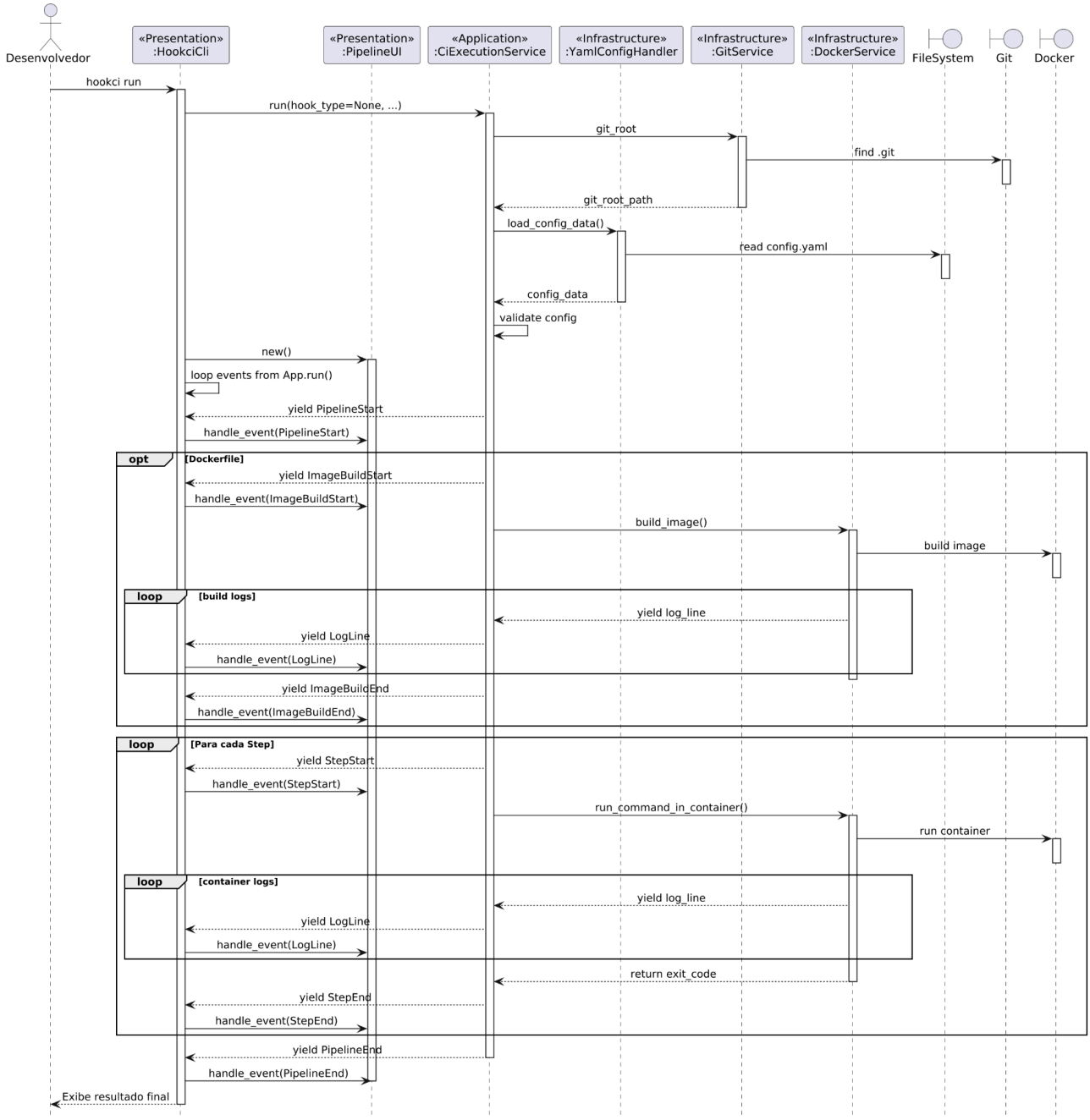

Figura 12. Diagrama de Sequência para a operação “Executar CI”

3.3 Diagramas de Comunicação

A Figura 13 mostra a interação para exibir a ajuda. O ator Usuário interage com a instância HookciCli da camada de apresentação, que responde diretamente.

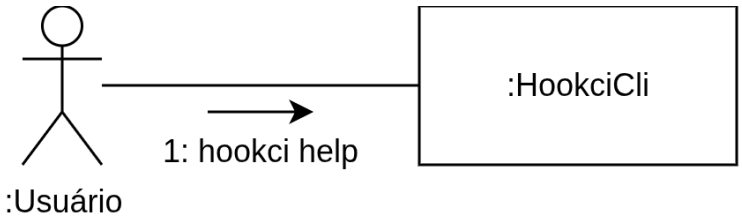

Figura 13. Diagrama de Comunicação para a operação "Mostrar ajuda"

A Figura 14 detalha as colaborações para inicializar um projeto. O Administrador aciona HookciCli, que delega para ProjectInitializationService. Este serviço interage com GitService e com YamlConfigurationHandler para configurar o projeto e instalar os *hooks*.

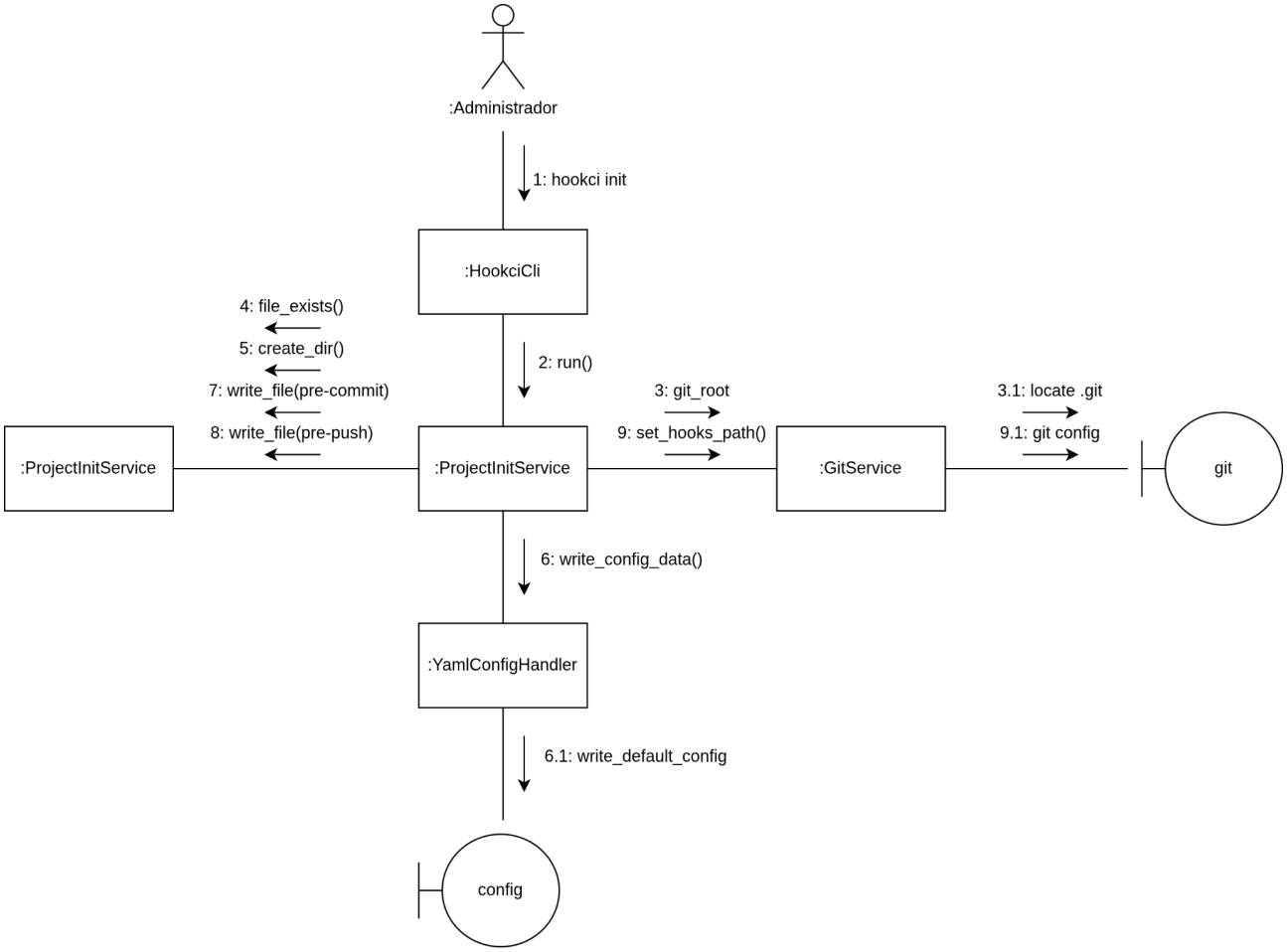

Figura 14. Diagrama de Comunicação para a operação "Inicializar projeto"

A Figura 15 ilustra as interações na migração de configuração. O Administrador inicia o processo via HookciCli, que chama o MigrationService. Este serviço usa o YamlConfigurationHandler para ler e escrever no ConfigFile, após processar a migração internamente.

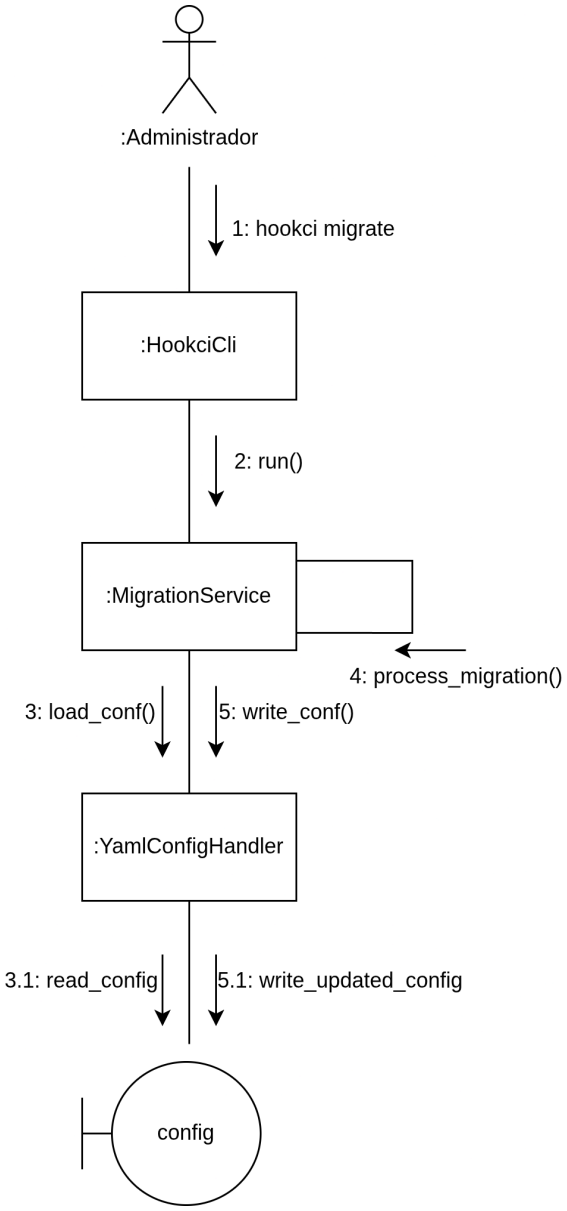

Figura 15. Diagrama de Comunicação para a operação "Migrar configuração"

A Figura 16 mostra a colaboração quando a CI é disparada por um *hook* do Git. O *hook* executa HookciCli, que invoca CiExecutionService. O serviço coordena a leitura da configuração, validação e obtenção de informações, construção opcional de imagem e execução de testes em contêineres.

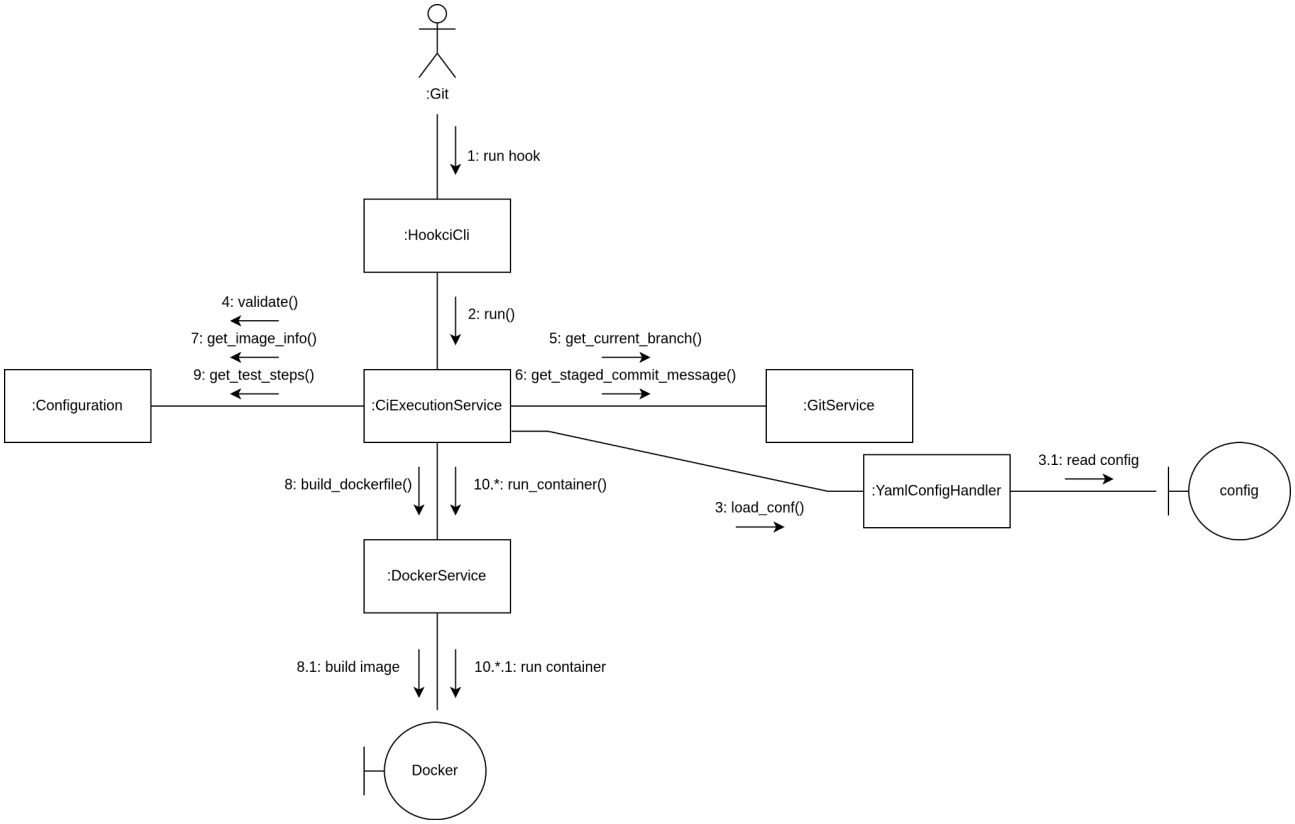

Figura 16. Diagrama de Comunicação para a operação "Executar CI via *hook*"

A Figura 17 apresenta a comunicação para a execução manual da CI. A interação é iniciada pelo Desenvolvedor através do HookciCli. A sequência de colaborações entre CiExecutionService, YamlConfigurationHandler, Configuration, DockerService e as fronteiras ConfigFile e DockerSystem é semelhante à execução via *hook* (Figura 16).

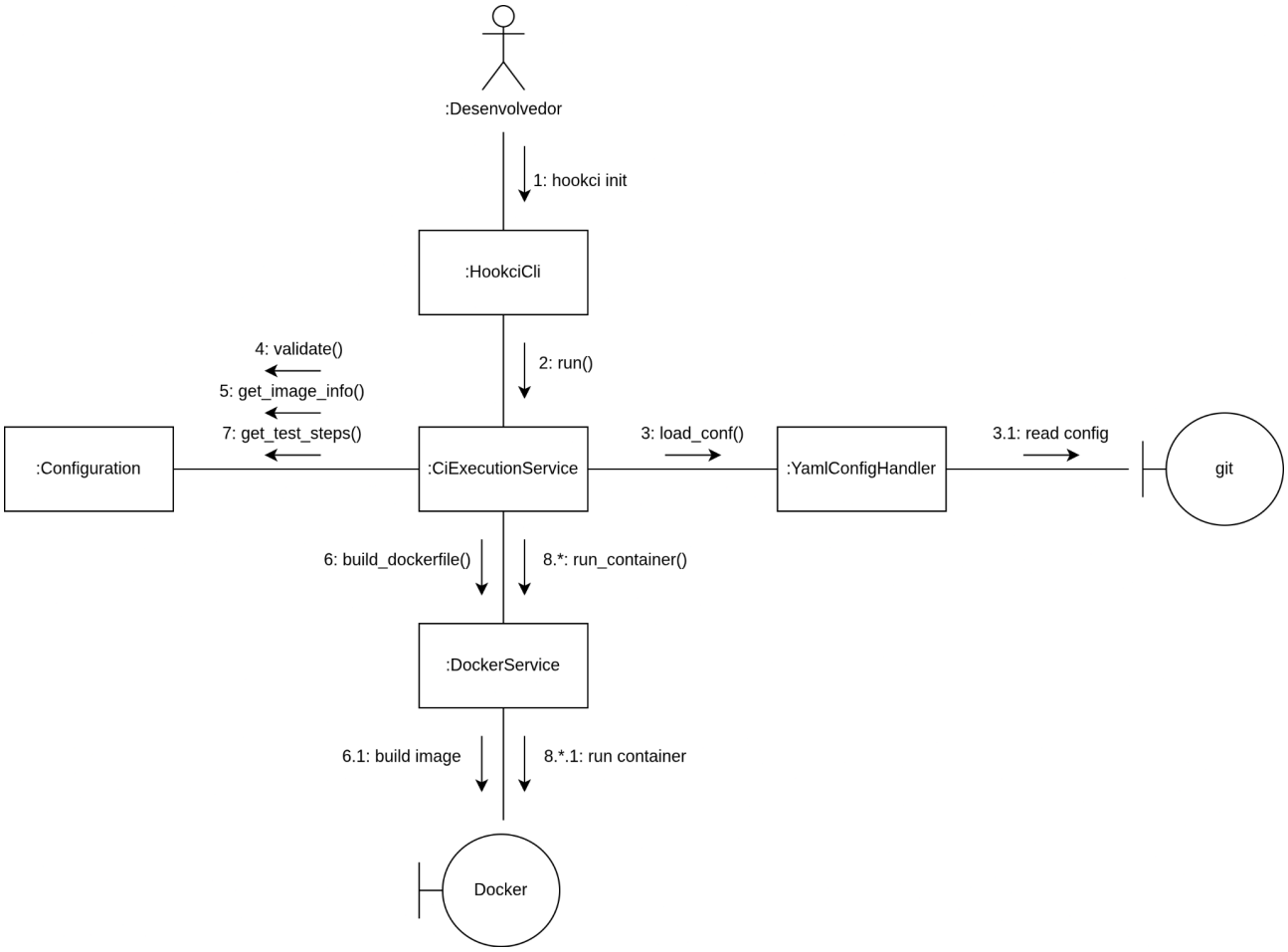

Figura 17. Diagrama de Comunicação para a operação "Executar CI"

3.4 Arquitetura

A arquitetura adota o padrão de camadas, visando a separação de responsabilidades e a manutenibilidade do código. A escolha pelo padrão de arquitetura em camadas fundamenta-se na natureza da aplicação e no equilíbrio entre estruturação e complexidade. O HookCI é uma ferramenta de CLI com um fluxo de execução linear e bem definido, onde a complexidade do domínio não justifica abstração imposta por arquiteturas mais complexas. A arquitetura em camadas oferece uma separação de responsabilidades clara e suficiente para garantir a testabilidade, isolando a lógica de negócios e os casos de uso das implementações concretas de IO, mantendo uma curva de aprendizado suave para a equipe e facilidade de manutenção. Além disso, a estrutura adotada facilita a evolução futura para padrões mais robustos caso o escopo do sistema se expanda, sem introduzir complexidade acidental prematura. A Figura 18 ilustra essa organização em quatro camadas principais: *Presentation*, *Application*, *Domain* e *Infrastructure*.

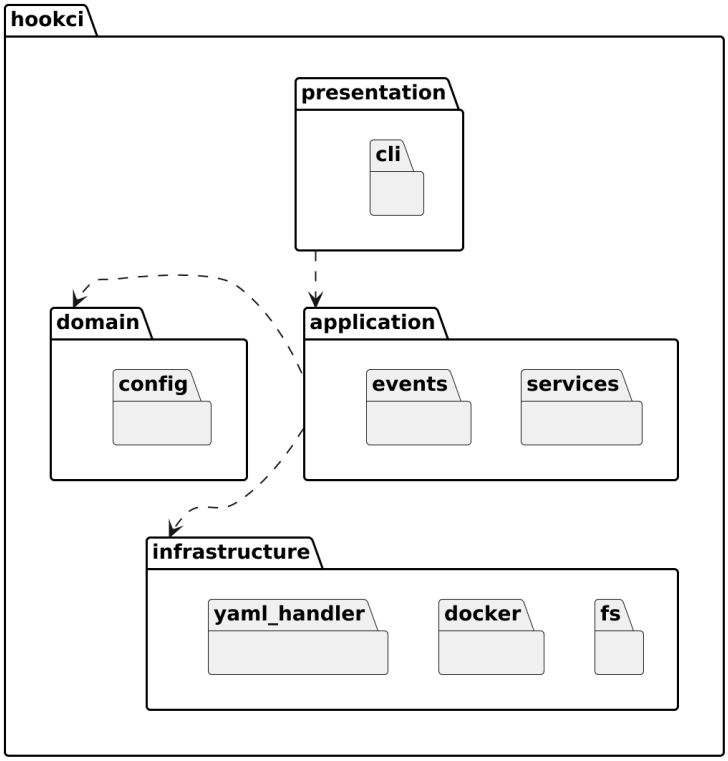

Figura 18. Diagrama de Pacotes da Arquitetura em Camadas

A camada de *Presentation* é responsável pela interação com o usuário, neste caso, através da CLI. Ela recebe os comandos do usuário e exibe os resultados. A camada de *Application* orquestra os casos de uso do sistema, coordenando as ações necessárias para atender às solicitações da camada de *Presentation*. Ela utiliza os serviços das camadas inferiores para executar suas tarefas, como gerenciamento de configuração e execução de fluxos de CI.

A camada de *Domain* encapsula a lógica de negócio central e as entidades do sistema, como os modelos de configuração e o processo de CI. Esta camada é independente das tecnologias externas e representa o núcleo do HookCI. Por fim, a camada de *Infrastructure* lida com detalhes técnicos e integrações externas, como a interação com Git, Docker, sistema de arquivos para leitura/escrita de configurações e a análise destas configurações. As dependências fluem das camadas superiores para as inferiores, garantindo que a lógica de negócio no *Domain* permaneça isolada das preocupações de infraestrutura e apresentação.

Para sustentar a integridade e a testabilidade deste modelo, a arquitetura emprega rigorosamente o Princípio da Inversão de Dependência (DIP) em conjunto com a Injeção de Dependência. As camadas de alto nível não dependem diretamente das implementações concretas de baixo nível; em vez disso, ambas dependem de abstrações. Isso se manifesta na definição de interfaces claras na camada de *Application* para serviços críticos, como o controle de versão e a orquestração de contêineres.

Consequentemente, as implementações reais desses serviços na camada de *Infrastructure* atuam como adaptadores intercambiáveis. Essa decisão estrutural isola a lógica de negócios da

complexidade das APIs do Docker e dos comandos do Git, permitindo que o sistema seja testado de forma isolada através de objetos simulados, garantindo que as regras de validação permaneçam consistentes e desacopladas das ferramentas externas.

3.5 Diagramas de Estados

Para modelar o comportamento dinâmico do sistema durante a execução de um ciclo de CI, o Diagrama de Estados apresentado pela Figura 19 apresenta os principais estados e transições do processo de execução do HookCI.

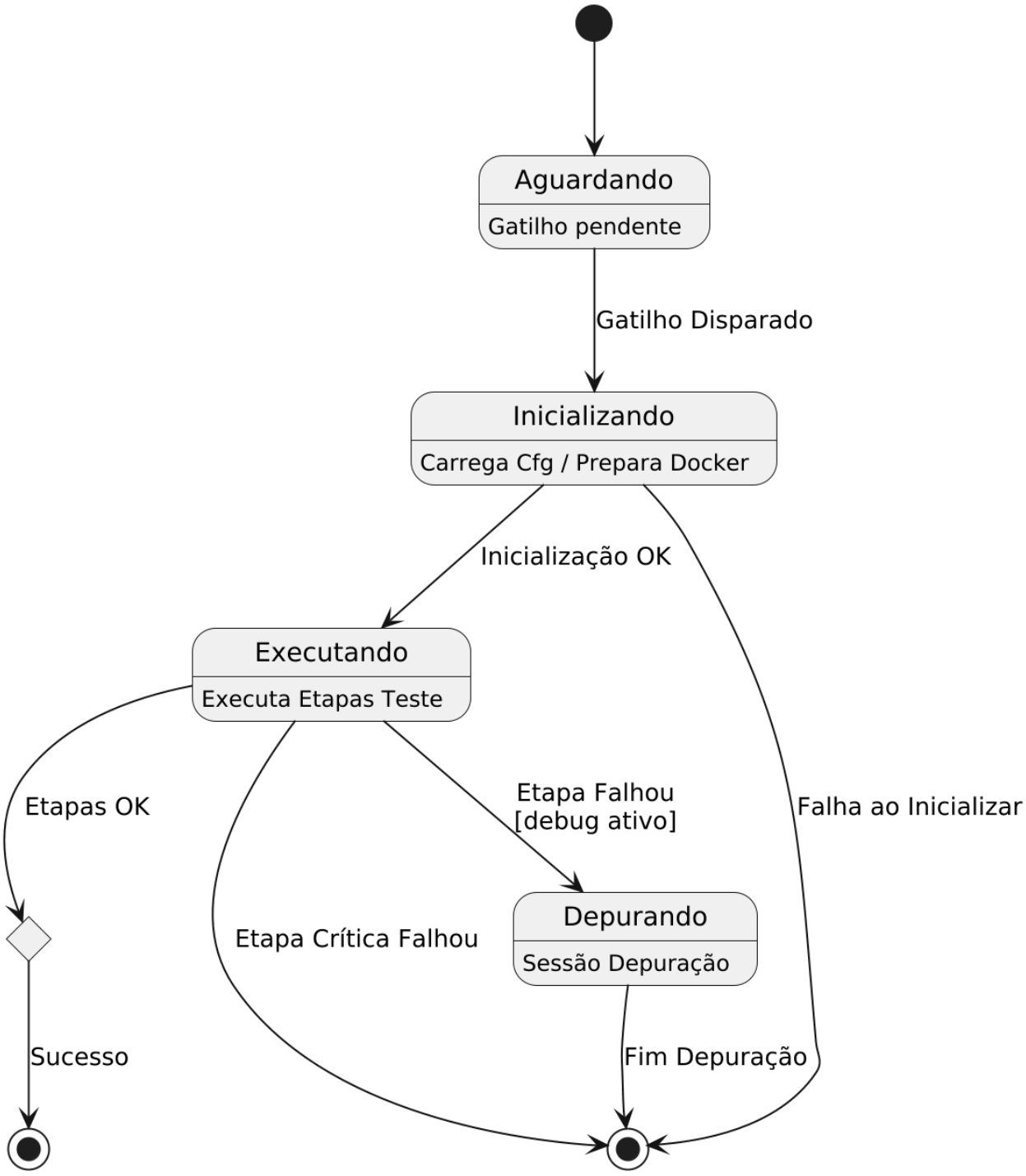

Figura 19. Diagrama de Estados da Execução de CI

O processo inicia no estado Aguardando, onde o sistema espera por um gatilho, seja um *hook* do Git ou um comando via CLI. Ao ser disparado, transita para o estado Inicializando, onde carrega a configuração do projeto e prepara o ambiente Docker. Se a inicialização for bem-sucedida, o sistema entra no estado Executando, processando sequencialmente cada etapa de teste definida na configuração. Durante a execução, se uma etapa falhar e o modo de depuração estiver ativo, o

sistema transitará para o estado Depurando. Caso contrário, ou após a depuração, se a etapa falha for crítica, o processo transita para o estado final Falha. Se uma etapa não crítica falhar, um aviso é gerado e a execução continua. Ao concluir todas as etapas, o sistema alcança um ponto de decisão que leva aos estados finais: OK, OK com Avisos ou Falha.

3.6 Diagrama de Componentes e Implantação.

Os diagramas de componentes e implantação detalham a estrutura modular do software e como ele é distribuído no ambiente de execução, respectivamente. A Figura 20 exibe os principais componentes de software do HookCI e suas interconexões.

Nesta arquitetura, o Domain Model atua como o núcleo do sistema, encapsulando as regras de negócio e a lógica de configuração sem depender de detalhes externos. Os Application Services orquestram a execução dos casos de uso, coordenando o fluxo de informações entre a apresentação e o domínio. A comunicação com o mundo externo é abstraída pelos componentes da camada de infraestrutura. O Git Adapter traduz solicitações internas em comandos para o Git, gerenciando o contexto do repositório e os gatilhos de execução, enquanto o Docker Adapter controla o Docker Engine, responsável pela construção das imagens e pelo ciclo de vida dos contêineres onde os testes ocorrem.

Esses sistemas externos, Git e Docker Engine, fornecem a base operacional necessária, sendo integrados ao HookCI estritamente através de seus respectivos adaptadores. Essa abordagem, que inclui também o YAML/File Handler para manipulação de arquivos, garante o desacoplamento entre a lógica de aplicação e as ferramentas de infraestrutura, assegurando a manutenibilidade e a robustez da solução.

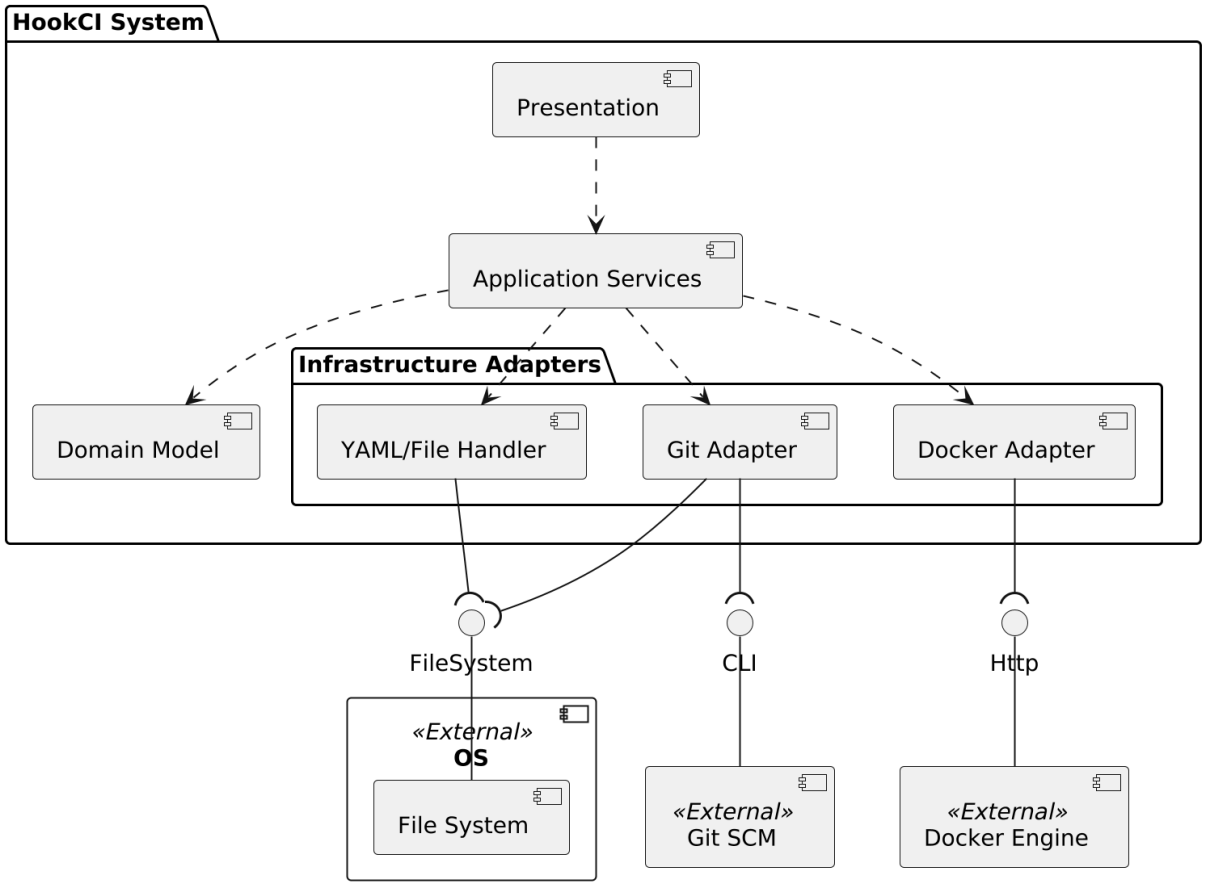

Figura 20. Diagrama de Componentes do Sistema HookCI

A Figura 21 ilustra o diagrama de implantação, mostrando como os artefatos de software são alocados nos nós do ambiente de execução típico, a estação de trabalho do desenvolvedor. Nesta estação residem o artefato principal HookCI CLI, o sistema de controle de versão Git, os *hooks* do Git configurados, o arquivo de configuração YAML e o código-fonte do projeto.

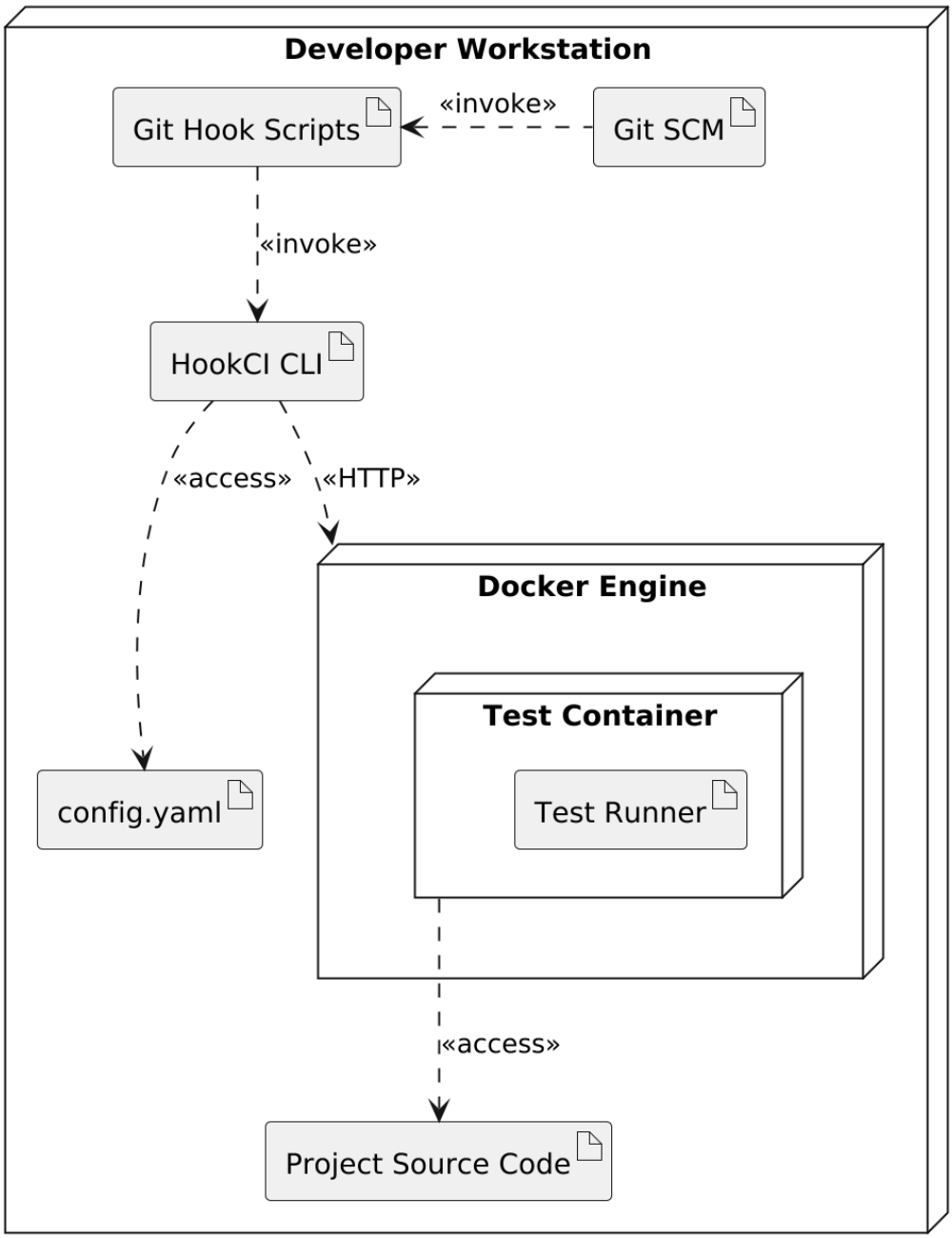

Figura 21. Diagrama de Implantação do Sistema HookCI

4. Projeto de Interface com Usuário

Esta seção apresenta as principais interações do usuário com o sistema HookCI por meio de sua CLI. Como o sistema opera exclusivamente via terminal, não são apresentados *mockups* ou *wireframes* gráficos. Em vez disso, são exibidas representações textuais das saídas geradas pelos comandos principais, ilustrando o fluxo de interação e o retorno fornecido ao usuário. Essas interações foram projetadas para cobrir as histórias de usuário definidas na Seção 2.3.

4.1 Esboço das Interfaces Comuns a Todos os Atores

A Figura 22 exibe a saída do comando de ajuda, invocado quando o usuário executa a ferramenta sem argumentos ou com o comando *help*. A interface lista os comandos disponíveis e fornece uma breve descrição de cada um, auxiliando o usuário a entender as funcionalidades da ferramenta. Esta tela contempla diretamente a história de usuário US01.

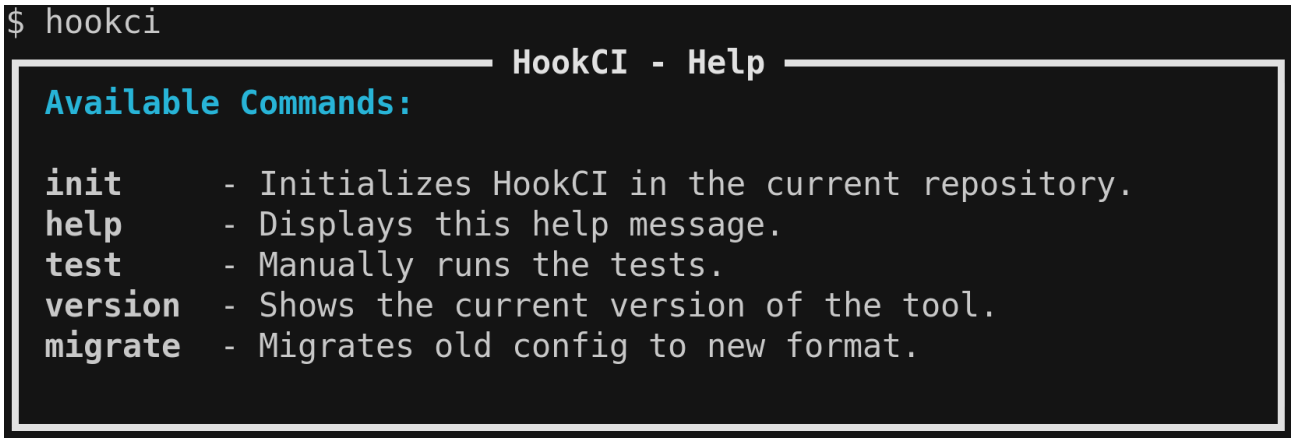

Figura 22. Descrição de comandos disponíveis

A Figura 23 mostra a resposta do sistema quando um comando inválido é fornecido pelo usuário. A interface apresenta uma mensagem de erro, indicando qual comando foi considerado desconhecido e sugerindo o uso do comando *help* para visualizar as opções válidas. Embora não corresponda diretamente a uma história de usuário funcional principal, esta interface é crucial para a usabilidade e orientação do usuário, relacionando-se indiretamente com US01 ao direcionar para a ajuda.

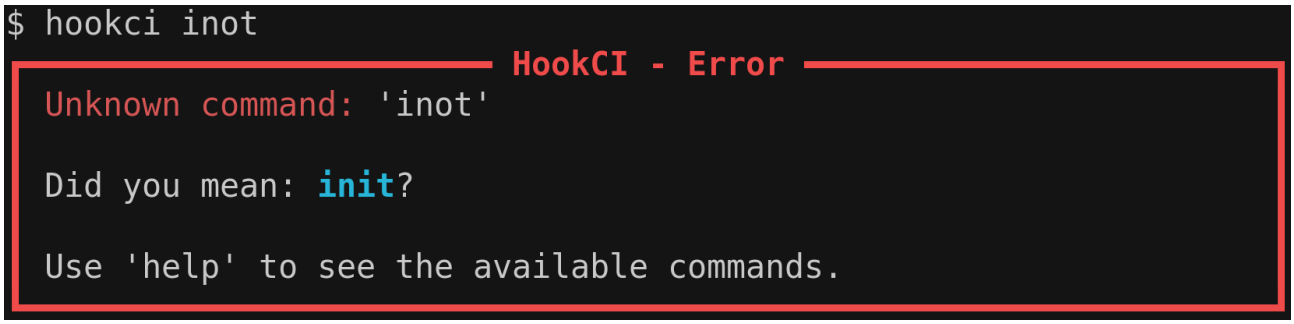

Figura 23. Comando inválido

4.2 Esboço das Interfaces Usadas pelo Desenvolvedor

A Figura 24 demonstra a saída da CLI durante a execução dos testes, acionada manualmente pelo comando *test* ou automaticamente por um Git *hook*. A interface exibe o progresso das etapas principais, como a inicialização, criação do contêiner Docker e execução da suíte de testes, utilizando indicadores visuais, sendo ⏳ para pendente/executando e ✔ para concluído. Assim pode-se fornecer retorno sobre o andamento.

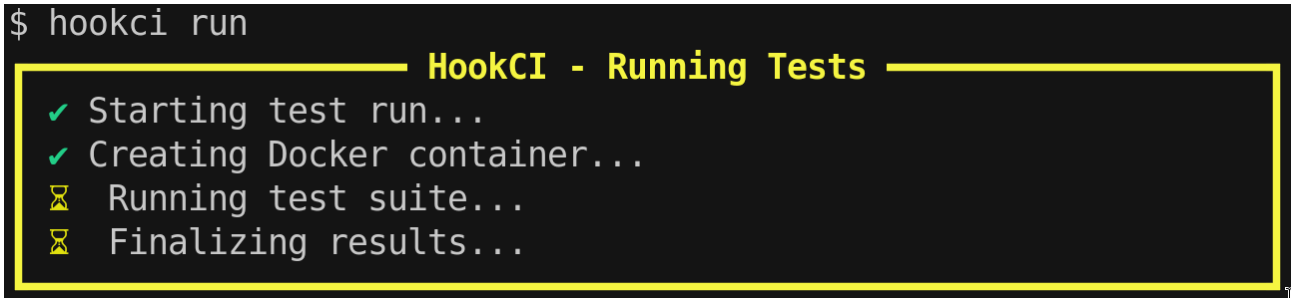

Figura 24. Execução de testes em andamento

A Figura 25 ilustra a conclusão bem-sucedida da execução dos testes. Esta interação está relacionada às histórias de usuário US05, US09, US10, US11, US12, US13, US14, US15, US16, US17 e US18.

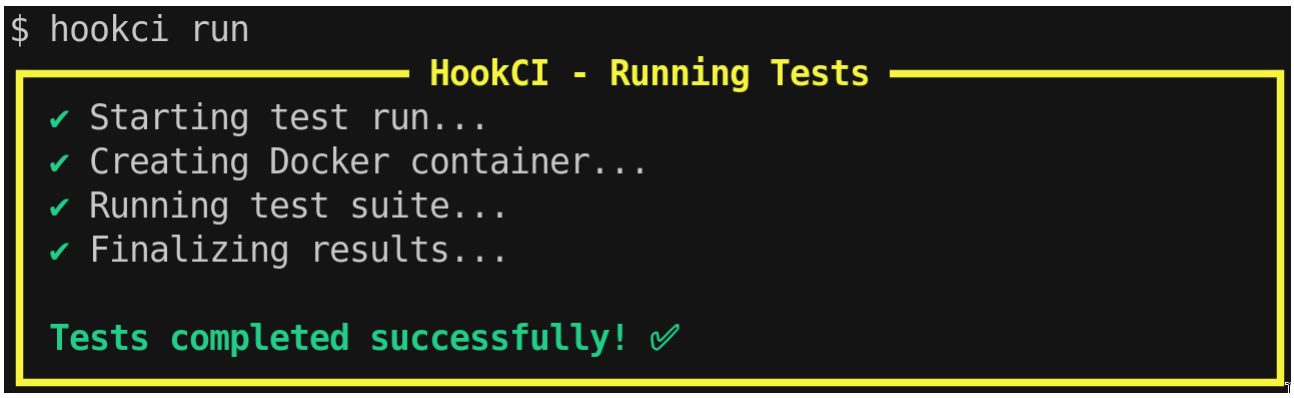

Figura 25. Execução de testes bem sucedida

4.3 Esboço das Interfaces Usadas pelo Administrador

A Figura 26 representa a saída do comando *init*, responsável por inicializar o HookCI em um repositório. A interface informa o início do processo e detalha as ações realizadas, como a criação do arquivo de configuração YAML, a instalação dos Git *hooks* necessários e a verificação de dependências. Esta interação abrange as histórias de usuário US02, US03, US04.

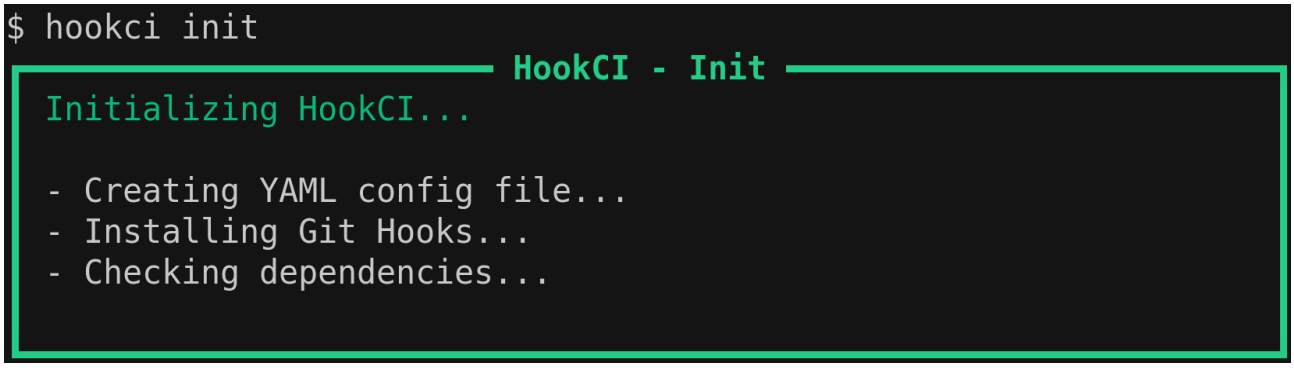

Figura 26. Inicialização de um repositório

A Figura 27 demonstra a saída do comando de migração quando bem sucedido, executado manualmente pelo usuário para alterar sua configuração de uma versão anterior da ferramenta para a versão atual. Esta interação contempla as histórias de usuário US06 e US07.

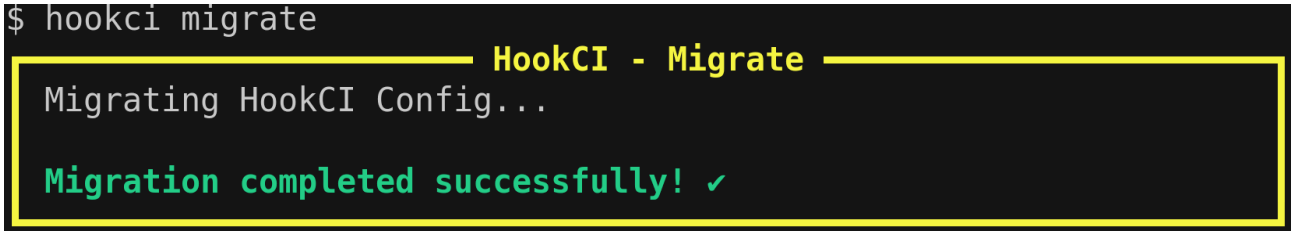

Figura 27. Migração de configuração bem sucedida

5. Glossário e Modelos de Dados

Esta seção apresenta o glossário dos termos utilizados no arquivo de configuração YAML do HookCI e o modelo de dados que descreve sua estrutura. Como o sistema não utiliza um banco de dados tradicional, o foco está na representação dos dados de configuração, que são armazenados e lidos a partir de um arquivo YAML. A Tabela 8 detalha os termos e os campos utilizados para configurar a ferramenta HookCI, servindo como glossário para a personalização do ambiente de um projeto.

Atributo	**Formato**	**Descrição**
version	string	Versão do esquema de configuração do HookCI. Utilizada para o gerenciamento de migrações entre diferentes versões da ferramenta.
log_level	string	Define o nível de verbosidade dos *logs* da CLI.
docker	object	Agrupa as configurações relacionadas ao ambiente Docker utilizado para a execução dos testes.
image	string	Nome e tag da imagem Docker pré-existente a ser utilizada. Se omitido, “dockerfile” deve ser especificado.
dockerfile	string	Caminho relativo para um *Dockerfile* no repositório, que será usado para construir a imagem Docker. Se omitido, “image_name” deve ser usado.
hooks	object	Define quais *hooks* do Git serão gerenciados e ativados pelo HookCI.
pre-commit	boolean	Se verdadeiro, ativa a execução do HookCI automaticamente durante o evento *pre-commit* do Git.

pre-push	boolean	Se verdadeiro, ativa a execução do HookCI automaticamente durante o evento *pre-push* do Git.
filters	object	Define filtros a serem aplicados em eventos git
branches	string	Filtros a serem aplicados sobre eventos por *branches*
commits	string	Filtros a serem aplicados sobre eventos evocados por *commits*
steps	object	Etapas a serem executadas sequencialmente durante o processo de CI.
name	string	Nome descritivo e único para a etapa de teste, utilizado para identificação nos *logs*.
depends_on	string	Lista de etapas previamente necessárias para a execução de uma etapa.
command	string	O comando a ser executado dentro do contêiner Docker para uma etapa.
critical	boolean	Se verdadeiro (padrão), uma falha nesta etapa interrompe a execução da CI e aborta a operação Git. Se falso, uma falha gera apenas um aviso.
env	object	Um mapa de pares chave-valor representando variáveis de ambiente a serem injetadas no contêiner Docker para uma etapa.
VAR	string	Valor de variável de ambiente a ser definida para uma etapa.

Tabela 8. Glossário do Arquivo de Configuração YAML

A Figura 28 a seguir ilustra, a estrutura hierárquica e os tipos de dados esperados para a configuração explicada acima, servindo como um esquema para sua validação e interpretação pela ferramenta. A opção pelo uso de um arquivo de configuração no formato YAML, em detrimento de uma solução baseada em banco de dados, fundamenta-se na busca por simplicidade operacional, portabilidade e integração direta com o sistema de versionamento do código-fonte do projeto.

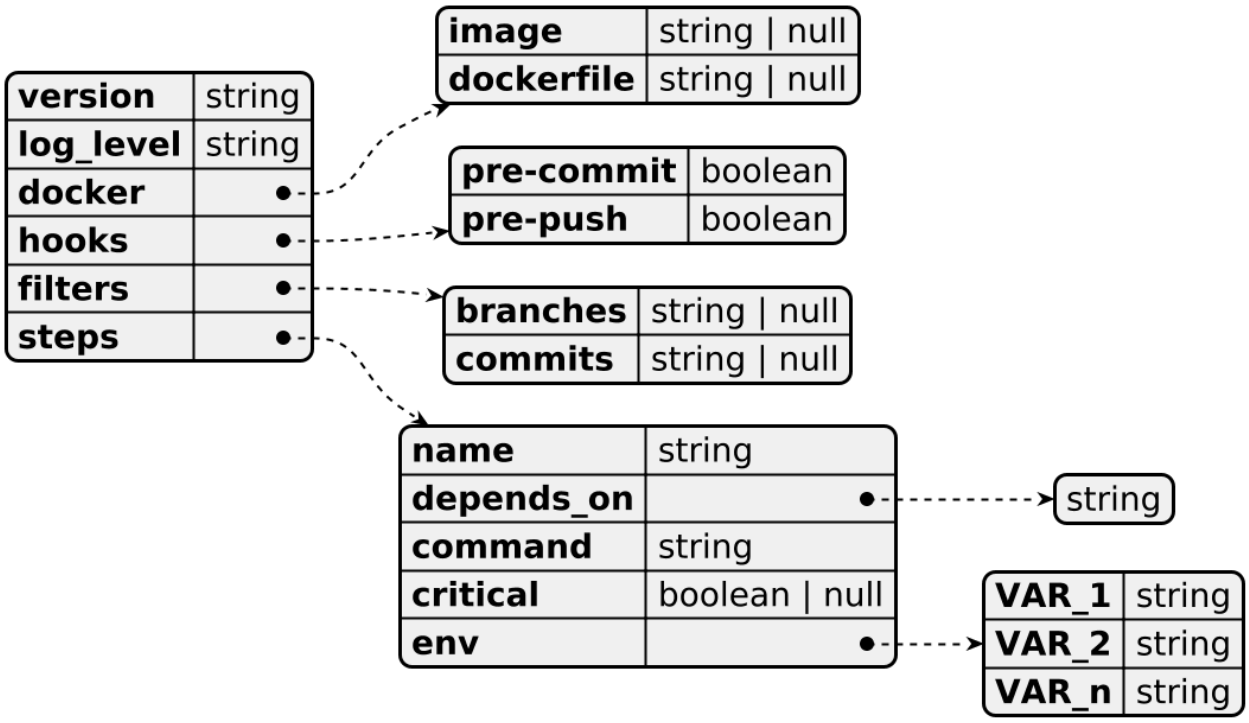

Figura 28. Mapeamento da Estrutura do Arquivo de Configuração YAML

6. Casos de Teste

Esta seção detalha os casos de teste projetados para validar as funcionalidades e integrações do sistema HookCI. Os testes são divididos em Testes de Aceitação, que verificam se os requisitos do usuário foram atendidos, e Testes de Integração, que focam na interação entre os diferentes componentes do sistema.

6.1 Testes de aceitação

Os Testes de Aceitação vistos abaixo são elaborados com base nas Histórias de Usuário
na seção 2.3.2 e nos Contratos de Operações na seção 2.4, visando garantir que o sistema se comporta conforme o esperado sob a perspectiva do usuário final. O caso de teste TA01, descrito na Tabela 9, valida a funcionalidade básica de ajuda da CLI, garantindo que o usuário possa consultar os comandos disponíveis e suas descrições, conforme a história de usuário US01. Refere-se à necessidade “Retorno Imediato aos Desenvolvedores”.

Identificador	**TA01**
Caso de Teste	Verificar a exibição da ajuda da CLI.
Pré-condições	HookCI instalado no ambiente.

Ações	Executar o comando “hookci help” ou “hookci” no terminal.
Resultados Esperados	A CLI exibe uma lista dos comandos disponíveis *init*, *run*, *migrate*, *help* com suas respectivas descrições, conforme Figura 22.

Tabela 9. Caso de teste TA01

O caso de teste TA02, descrito na Tabela 10, verifica o processo de inicialização do HookCI em um repositório Git novo para a ferramenta, assegurando a criação correta do arquivo de configuração e a instalação dos hooks, conforme as histórias de usuário. Refere-se às necessidades “Flexibilidade na Configuração e Execução” e “Pré-configuração na inicialização da ferramenta no repo”.

Identificador	**TA02**
Caso de Teste	Inicializar HookCI em um novo projeto Git.
Pré-condições	HookCI instalado; Docker instalado; Usuário está em um diretório de projeto Git que ainda não foi inicializado com HookCI.
Ações	Executar o comando “hookci init” no terminal.
Resultados Esperados	O sistema cria o arquivo YAML padrão na raiz do projeto, instala os scripts de *hook* no diretório local do Git, e exibe uma mensagem de sucesso, conforme Figura 26.

Tabela 10. Caso de teste TA02

O caso de teste TA03, descrito na Tabela 11, assegura que a tentativa de inicializar um projeto já configurado com HookCI resulte em um aviso, prevenindo a sobrescrita acidental de configurações. Refere-se à necessidade “Pré-configuração na inicialização da ferramenta no repo”.

Identificador	**TA03**
Caso de Teste	Tentar inicializar HookCI em um projeto já inicializado.
Pré-condições	HookCI instalado; Docker instalado; Usuário está em um diretório de projeto Git que já foi inicializado com HookCI.
Ações	Executar o comando “hookci init” no terminal.
Resultados Esperados	O sistema detecta que o projeto já está inicializado e exibe uma mensagem de aviso, sem sobrescrever configurações ou *hooks* existentes.

Tabela 11. Caso de teste TA03

O caso de teste TA04, descrito na Tabela 12, valida o cenário ideal da execução automática dos testes via *hooks* do Git, onde todos os testes passam e as operações Git são concluídas com sucesso.

Refere-se às necessidades “Validação Preventiva de Código”, “Pré-configuração na inicialização da ferramenta no repo”, “Otimização do Tempo de Execução” e “Retorno Imediato aos Desenvolvedores”.

Identificador	**TA04**
Caso de Teste	Execução automática de testes via *hook pre-commit* e *pre-push* com sucesso.
Pré-condições	HookCI inicializado no projeto; Configuração com uma etapa de teste válida sempre positiva; *Hook pre-commit* e *pre-push* habilitados na configuração; Docker em execução.
Ações	Usuário modifica um arquivo no repositório; Executa “git add .”; Executa “git commit -m "mensagem"”; Executa “git push”
Resultados Esperados	Um *hook pre-commit* dispara o HookCI. Os testes são executados em um contêiner Docker. Todos os testes passam. O *commit* é realizado com sucesso. Um *hook pre-push* dispara o HookCI. Os testes são executados em um contêiner Docker. Todos os testes passam. O *push* é realizado com sucesso.

Tabela 12. Caso de teste TA04

O caso de teste TA05, descrito na Tabela 13, verifica se uma falha em um teste crítico durante a execução automática via *hooks* e impede a conclusão da operação Git. Refere-se às necessidades “Validação Preventiva de Código”, “Pré-configuração na inicialização da ferramenta no repo”, “Otimização do Tempo de Execução” e “Retorno Imediato aos Desenvolvedores”.

Identificador	**TA05**
Caso de Teste	Execução automática de testes via *hook pre-commit* e *pre-push* com falha crítica.
Pré-condições	HookCI inicializado no projeto; Configuração com uma etapa de teste válida sempre positiva; *Hook pre-commit* e *pre-push* habilitados na configuração; Docker em execução.
Ações	Usuário modifica um arquivo no repositório; Executa “git add .”; Executa “git commit -m "mensagem"”; Executa “git push”
Resultados Esperados	Um *hook pre-commit* dispara o HookCI. Os testes são executados em um contêiner Docker. Um teste crítico falha. O *commit* é abortado. Um *hook pre-push* dispara o HookCI. Os testes são executados em um contêiner Docker. Um teste crítico falha. O *push* é abortado.

Tabela 13. Caso de teste TA05

O caso de teste TA06, descrito na Tabela 14, testa a execução manual dos testes com o comando "hookci run", garantindo que, em caso de sucesso, os *logs* apropriados sejam exibidos. Refere-se às necessidades "Flexibilidade na Configuração e Execução", "Otimização do Tempo de Execução" e "Retorno Imediato aos Desenvolvedores".

Identificador	**TA06**
Caso de Teste	Execução manual de testes "hookci run" com sucesso.
Pré-condições	HookCI inicializado no projeto; YAML configurado com uma etapa de teste válida sempre positiva; Docker em execução.
Ações	Usuário executa "hookci run" no terminal.
Resultados Esperados	HookCI executa todas as etapas de teste configuradas em containers Docker. Testes passam. *Logs* de sucesso são exibidos ao Usuário conforme Figura 24 e 25.

Tabela 14. Caso de teste TA06

O caso de teste TA07, descrito na Tabela 15, valida que modificações no arquivo de configuração são corretamente interpretadas e aplicadas na subsequente execução dos testes. Refere-se à necessidade "Flexibilidade na Configuração e Execução".

Identificador	**TA07**
Caso de Teste	Editar arquivo de configuração e executar testes.
Pré-condições	HookCI inicializado; Configuração existente; Docker em execução.
Ações	Usuário edita a configuração para adicionar um novo comando de teste; Usuário executa "hookci run".
Resultados Esperados	O novo comando de teste é executado corretamente durante a CI. O resultado reflete a alteração na configuração.

Tabela 15. Caso de teste TA07

O caso de teste TA08, descrito na Tabela 16, verifica a funcionalidade de migração de configuração, assegurando que configurações de versões anteriores possam ser atualizadas para o formato mais recente da ferramenta. Refere-se à necessidade "Migração das configurações entre versões".

Identificador	**TA08**
Caso de Teste	Migrar configuração de projeto para versão mais recente.

Pré-condições	HookCI instalado; Projeto Git com uma configuração de uma versão anterior do HookCI.
Ações	Usuário executa “hookci migrate” no terminal.
Resultados Esperados	O arquivo de configuração é atualizado para o esquema da versão mais recente. Uma mensagem de sucesso é exibida como na Figura 27. Se a migração automática não for possível, um alerta é emitido.

Tabela 16. Caso de teste TA08

O caso de teste TA09, descrito na Tabela 17, garante que o HookCI valide a estrutura do arquivo de configuração e reporte erros, impedindo a execução com uma configuração inválida. Refere-se à necessidade “Flexibilidade na Configuração e Execução”.

Identificador	**TA09**
Caso de Teste	Validar estrutura incorreta da configuração.
Pré-condições	HookCI inicializado; Configuração com um campo obrigatório ausente ou com tipo inválido.
Ações	Usuário executa “hookci run” ou Git dispara um *hook*.
Resultados Esperados	O HookCI detecta a invalidade da configuração antes de iniciar a execução dos testes e exibe uma mensagem de erro informativa, indicando o problema na configuração. A execução da CI e a operação Git são abortadas.

Tabela 17. Caso de teste TA09

O caso de teste TA10, descrito na Tabela 18, valida a configuração de testes não essenciais, onde uma falha gera um aviso, mas não impede o fluxo da CI ou a operação Git. Refere-se à necessidade “Validação Preventiva de Código”.

Identificador	**TA10**
Caso de Teste	Configurar teste não essencial.
Pré-condições	HookCI inicializado no projeto; Configuração com uma etapa de teste válida sempre negativa; Hook pre-commit e pre-push habilitados na configuração; Docker em execução.
Ações	Usuário executa “hookci run” ou Git dispara um *hook*.
Resultados Esperados	A etapa de teste não crítica falha, um aviso é gerado nos logs, mas a execução da CI continua ou a operação Git é permitida.

Tabela 18. Caso de teste TA10

O caso de teste TA11, descrito na Tabela 19, verifica a capacidade do sistema de utilizar uma imagem Docker definida na configuração por nome e versão para a execução dos testes. Refere-se às necessidades “Isolamento do Ambiente de CI” e “Especificar imagens docker para runtime da CI”.

Identificador	**TA11**
Caso de Teste	Utilizar imagem Docker personalizada por nome e versão.
Pré-condições	HookCI inicializado; Configuração com imagem definida para uma imagem:tag existente no Docker Hub ou localmente; Docker em execução.
Ações	Usuário executa “hookci run” ou Git dispara um *hook*.
Resultados Esperados	O HookCI utiliza a imagem Docker especificada para executar os testes. Os testes são executados no ambiente dessa imagem.

Tabela 19. Caso de teste TA11

O caso de teste TA12, descrito na Tabela 20, assegura que o HookCI possa construir e utilizar uma imagem Docker a partir de um *Dockerfile* especificado na configuração. Refere-se às necessidades “Isolamento do Ambiente de CI” e “Especificar imagens docker para runtime da CI”.

Identificador	**TA12**
Caso de Teste	Utilizar *Dockerfile* para construir imagem Docker.
Pré-condições	HookCI inicializado; Configuração com “dockerfile” apontando para um *Dockerfile* válido no repositório; Docker em execução.
Ações	Usuário executa “hookci run” ou Git dispara um *hook*.
Resultados Esperados	O HookCI utiliza a imagem Docker especificada para executar os testes. Os testes são executados no ambiente dessa imagem.

Tabela 20. Caso de teste TA12

O caso de teste TA13, descrito na Tabela 21, valida a funcionalidade de ajuste do nível de verbosidade dos *logs*, permitindo ao usuário controlar a quantidade de informações exibidas durante a execução. Refere-se à necessidade “Depuração da CI”.

Identificador	**TA13**
Caso de Teste	Ajustar nível de verbosidade dos *logs.*

Pré-condições	HookCI inicializado; Configuração com “log_level” definido.
Ações	Executar o comando “hookci run”.
Resultados Esperados	Os *logs* exibidos na CLI correspondem ao nível de verbosidade configurado.

Tabela 21. Caso de teste TA13

O caso de teste TA14, descrito na Tabela 22, valida a funcionalidade de filtro por *branch* e *commit*, garantindo que a CI seja executada quando as condições são atendidas. Refere-se à necessidade “Suporte ao *workflow* de desenvolvimento”.

Identificador	**TA14**
Caso de Teste	Filtrar execução de CI por *branch* e *commit.*
Pré-condições	HookCI inicializado; Configuração com “filters: branches: "feature/.*"”; Usuário está no branch 'feature/login'. Configuração com “filters: commits: "docs:*"'"; Usuário faz *commit* 'docs: mycommit'.
Ações	Git dispara *hook.*
Resultados Esperados	O HookCI identifica que o *branch*/*commit* corresponde ao filtro e executa a CI normalmente.

Tabela 22. Caso de teste TA14

O caso de teste TA15, descrito na Tabela 23, valida a funcionalidade de filtro por *branch* e *commit*, garantindo que a CI seja pulada quando as condições são atendidas. Refere-se à necessidade “Suporte ao *workflow* de desenvolvimento”.

Identificador	**TA15**
Caso de Teste	Pular execução de CI por *branch* e *commit.*
Pré-condições	HookCI inicializado; Configuração com “filters: branches: "feature/.*"”; Usuário está no branch bugfix/login'. Configuração com “filters: commits: "docs:*"'"; Usuário faz *commit* 'feat: mycommit'.
Ações	Git dispara *hook.*
Resultados Esperados	O HookCI identifica que o *branch*/*commit* não corresponde ao filtro e permite a operação git sem executar a CI.

Tabela 23. Caso de teste TA15

O caso de teste TA16, descrito na Tabela 24, verifica a funcionalidade do modo de depuração interativo, uma ferramenta para investigar falhas durante a execução da CI. Refere-se à necessidade "Depuração da CI".

Identificador	**TA16**
Caso de Teste	Acessar console de depuração em caso de falha.
Pré-condições	HookCI inicializado; Configuração com uma etapa de teste que falha; CI executada manualmente com a opção de depuração ativa.
Ações	Usuário executa "hookci run" com depuração ativa.
Resultados Esperados	A etapa de teste falha. Em vez de encerrar, o HookCI mantém o contêiner Docker em execução e anexa o terminal do usuário a um shell interativo dentro do contêiner, permitindo a investigação do ambiente no momento da falha.

Tabela 24. Caso de teste TA16

O caso de teste TA17, descrito na Tabela 25, verifica o comportamento do sistema quando não possui permissões de escrita no diretório para inicialização. Refere-se à necessidade "Robustez do sistema".

Identificador	**TA17**
Caso de Teste	Falha ao inicializar sem permissão de escrita.
Pré-condições	HookCI instalado; Repositório Git com permissão de leitura apenas.
Ações	Executar "hookci init".
Resultados Esperados	Mensagem de erro indicando falha de permissão ao criar arquivos ou diretórios. Código de saída de erro.

Tabela 25. Caso de teste TA17

O caso de teste TA18, descrito na Tabela 26, assegura que a ferramenta informe corretamente o usuário quando executada fora do contexto esperado, um repositório Git. Refere-se à necessidade "Robustez do sistema".

Identificador	**TA18**
Caso de Teste	Falha ao executar fora de repositório Git.
Pré-condições	HookCI instalado; Diretório atual não é um repositório git.

Ações	Executar “hookci init” ou “hookci run”.
Resultados Esperados	Mensagem de erro “Not inside a Git repository”. Código de saída de erro.

Tabela 26. Caso de teste TA18

O caso de teste TA19, descrito na Tabela 27, valida que a ferramenta lida corretamente com solicitações redundantes de migração. Refere-se à necessidade “Migração das configurações entre versões”.

Identificador	**TA19**
Caso de Teste	Migração em configuração já atualizada.
Pré-condições	Configuração válida na versão atual.
Ações	Executar “hookci migrate”.
Resultados Esperados	Mensagem informativa “Configuration is already up-to-date”. Código de saída 0.

Tabela 27. Caso de teste TA19

O caso de teste TA20, descrito na Tabela 28, verifica a integração com arquivos .env para injeção de variáveis de ambiente. Refere-se à necessidade “Suporte ao *workflow* de desenvolvimento”.

Identificador	**TA20**
Caso de Teste	Injeção de variáveis via .env
Pré-condições	Arquivo .env na raiz com “TEST_VAR=hello”. Step configurado para imprimir essa var.
Ações	Executar “hookci run”.
Resultados Esperados	O log do step exibe “hello”.

Tabela 28. Caso de teste TA20

O caso de teste TA21, descrito na Tabela 29, assegura que as configurações específicas de variáveis de ambiente no YAML tenham precedência sobre valores genéricos. Refere-se à necessidade “Flexibilidade na Configuração e Execução”.

Identificador	**TA21**
Caso de Teste	Precedência de variáveis.
Pré-condições	.env com “VAR=file”. Configuração de passo com “VAR=yaml”.
Ações	Executar “hookci run”.
Resultados Esperados	O log do step exibe “yaml”.

Tabela 29. Caso de teste TA21

O caso de teste TA22, descrito na Tabela 30, valida a detecção de configurações conflitantes para o ambiente Docker. Refere-se à necessidade “Robustez do sistema”.

Identificador	**TA22**
Caso de Teste	Erro configuração Docker conflitante
Pré-condições	YAML com “image” e “dockerfile” definidos.
Ações	Executar “hookci run”.
Resultados Esperados	Erro de validação indicando conflito. CI abortada.

Tabela 30. Caso de teste TA22

O caso de teste TA23, descrito na Tabela 31, valida a obrigatoriedade de configuração do ambiente Docker. Refere-se à necessidade “Robustez do sistema”.

Identificador	**TA23**
Caso de Teste	Erro configuração Docker ausente.
Pré-condições	YAML sem “image” e sem “dockerfile”.
Ações	Executar “hookci run”.
Resultados Esperados	Erro de validação indicando necessidade de configuração Docker. CI abortada.

Tabela 31. Caso de teste TA23

O caso de teste TA24, descrito na Tabela 32, verifica a detecção de ciclos no grafo de dependências das etapas. Refere-se à necessidade “Robustez do sistema”.

Identificador	**TA24**
Caso de Teste	Erro dependência circular.
Pré-condições	YAML com Step A dependendo de B, e B de A.
Ações	Executar “hookci run”.
Resultados Esperados	Erro de validação “Circular dependency detected”. CI abortada.

Tabela 32. Caso de teste TA24

O caso de teste TA25, descrito na Tabela 33, assegura que dependências apontem para etapas existentes. Refere-se à necessidade “Robustez do sistema”.

Identificador	**TA25**
Caso de Teste	Erro dependência inexistente.
Pré-condições	YAML com Step A dependendo de Z, em que Z não existe.
Ações	Executar “hookci run”.
Resultados Esperados	Erro de validação “Step 'A' depends on unknown step 'Z'”. CI abortada.

Tabela 33. Caso de teste TA25

O caso de teste TA26, descrito na Tabela 34, valida a proibição de auto-dependência em etapas. Refere-se à necessidade “Robustez do sistema”.

Identificador	**TA26**
Caso de Teste	Erro auto-dependência
Pré-condições	YAML com Step A dependendo de A.
Ações	Executar “hookci run”.
Resultados Esperados	Erro de validação “Step 'A' cannot depend on itself”. CI abortada.

Tabela 34. Caso de teste TA26

O caso de teste TA27, descrito na Tabela 35, valida o comportamento de bloqueio de execução para etapas dependentes de uma etapa falha.

Identificador	**TA27**
Caso de Teste	Falha bloqueia dependentes.
Pré-condições	Step A falha. Step B depende de A.
Ações	Executar “hookci run”.
Resultados Esperados	Step A falha. Step B não é executado. Status final FALHA.

Tabela 35. Caso de teste TA27

O caso de teste TA28, descrito na Tabela 36, verifica a funcionalidade básica de identificação da versão da ferramenta. Refere-se à necessidade “Retorno Imediato aos Desenvolvedores”.

Identificador	**TA28**
Caso de Teste	Exibir versão.
Pré-condições	Ferramenta instalada no ambiente.
Ações	Executar “hookci --version”.
Resultados Esperados	A versão atual da ferramenta é exibida.

Tabela 36. Caso de teste TA28

O caso de teste TA29, descrito na Tabela 37, assegura que a ferramenta lide robustamente com arquivos de configuração malformados. Refere-se à necessidade “Robustez do sistema”.

Identificador	**TA29**
Caso de Teste	Configuração YAML malformada
Pré-condições	Arquivo YAML com erro de indentação.
Ações	Executar “hookci run”.
Resultados Esperados	Erro de parsing do YAML exibido. Execução abortada.

Tabela 37. Caso de teste TA29

O caso de teste TA30, descrito na Tabela 38, verifica a execução correta de um fluxo de trabalho com múltiplas etapas e dependências.

Identificador	**TA30**
Caso de Teste	Sucesso em pipeline de dependências.
Pré-condições	Pipeline com steps paralelos e dependências. Todos os comandos retornam com sucesso.
Ações	Executar “hookci run”.
Resultados Esperados	Todos steps executados na ordem correta. Status final SUCESSO.

Tabela 38. Caso de teste TA30

6.2 Testes de integração

Os Testes de Integração focam em verificar as interações de alto nível entre o HookCI e os sistemas externos com os quais ele se comunica, como o Git, o Docker Engine e o Sistema de Arquivos. Estes testes garantem que os fluxos de dados e controle entre o HookCI e esses sistemas ocorram conforme o esperado. O caso de teste TI01, descrito na Tabela 39, verifica a robustez do HookCI ao lidar com a indisponibilidade do Docker Engine, informando o erro ao usuário e interrompendo a execução..

Identificador	**TI01**
Caso de Teste	Falha na comunicação com Docker Engine
Interface	CLI ou Git *Hooks*
Sistemas Envolvidos	Docker Engine, Git *Hooks*
Pré-condições	Docker Engine não está em execução ou inacessível.
Ações	Usuário executa “hookci run” ou Git dispara *hook*.
Resultados Esperados	O HookCI detecta a falha de comunicação com o Docker Engine; Exibe uma mensagem de erro ao usuário indicando o problema com o Docker; Aborta a execução dos testes e da operação Git; Retorna um código de saída diferente de zero.

Tabela 39. Caso de teste TI01

O caso de teste TI02, descrito na Tabela 40, garante que a inicialização falhe quando executado fora de um repositório Git, informando o usuário sobre a condição necessária.

Identificador	**TI02**
Caso de Teste	Tentativa de inicialização fora de um repositório Git válido
Interface	CLI
Sistemas Envolvidos	Sistema de Arquivos, Git
Pré-condições	HookCI instalado; Usuário está em um diretório ou subdiretório que não é um repositório Git.
Ações	Usuário executa “hookci init”.
Resultados Esperados	O HookCI detecta que não está em um repositório Git válido; Exibe uma mensagem de erro ao usuário; Não se inicializa; Retorna um código de saída diferente de zero.

Tabela 40. Caso de teste TI02

O caso de teste TI03, descrito na Tabela 41, verifica o comportamento do sistema durante a inicialização em um cenário onde existem restrições de permissão de escrita no sistema de arquivos, informando o usuário e interrompendo a execução.

Identificador	**TI03**
Caso de Teste	Permissões de arquivo insuficientes para instalar-se em repositório
Interface	CLI
Sistemas Envolvidos	Sistema de Arquivos
Pré-condições	HookCI instalado; Usuário está em um repositório Git; O diretório do projeto não tem permissão de escrita para o usuário atual.
Ações	Usuário executa “hookci init”.
Resultados Esperados	O HookCI detecta que não tem permissões de escrita no repositório Git; Exibe uma mensagem de erro ao usuário; Não se inicializa; Retorna um código de saída diferente de zero.

Tabela 41. Caso de teste TI03

O caso de teste TI04, descrito na Tabela 42, valida a capacidade do HookCI de tratar problemas com o arquivo de configuração, como sua ausência, impossibilidade de leitura ou erros de sintaxe, informando o usuário e interrompendo a execução.

Identificador	**TI04**
Caso de Teste	Ilegibilidade de Configuração
Interface	CLI ou Git *Hooks*
Sistemas Envolvidos	Sistema de Arquivos
Pré-condições	HookCI instalado; Usuário está em um repositório Git inicializado; A configuração não pode ser encontrada, ou não pode ser lida ou é inválida.
Ações	Usuário executa "hookci run" ou Git dispara *hook*.
Resultados Esperados	O HookCI tenta parsear configuração e detecta o erro de sintaxe; Exibe uma mensagem de erro, indicando a linha ou natureza do erro de parsing ou leitura do arquivo; Aborta a execução de testes e da operação Git; Retorna um código de saída diferente de zero.

Tabela 42. Caso de teste TI04

O caso de teste TI05, descrito na Tabela 43, verifica como o HookCI interage com o Docker Engine em caso de falha ao obter ou construir a imagem Docker especificada, informando o erro ao usuário e interrompendo a execução.

Identificador	**TI05**
Caso de Teste	Erro ao baixar ou construir imagem docker
Interface	CLI
Sistemas Envolvidos	Docker Engine
Pré-condições	HookCI inicializado; Configuração especifica imagem que não existe localmente nem em nenhum registry configurado ou *Dockerfile* inexistente ou que retorne erro ao ser construído.
Ações	Usuário executa "hookci run" ou Git dispara *hook*.
Resultados Esperados	O HookCI instrui o Docker Engine a usar a imagem/*dockerfile*; O Docker Engine falha ao tentar puxar/construir a imagem; O HookCI reporta o erro do Docker Engine ao usuário; Aborta a execução de testes ou operação Git; Retorna um código de saída diferente de zero.

Tabela 43. Caso de teste TI05

7. Cronograma e Processo de Implementação

Esta seção detalha o cronograma de atividades planejado para a implementação do sistema HookCI e o processo de desenvolvimento que será seguido. O cronograma abrange o período do segundo semestre de 2025, dividido em quinzenas, com atribuições específicas para cada membro da equipe.

7.1 Cronograma

O cronograma a seguir estabelece as metas quinzenais para cada membro da equipe, Gabriel Dolabela Marques (GDM) e Henrique Carvalho Almeida (HCA), cobrindo o desenvolvimento de requisitos funcionais, não funcionais e aspectos de CI e Entrega contínua (CD do inglês *Continuous Deployment*) do projeto.

Período	**Aluno**	**Atividade**	**Requisitos associados**
Agosto/2025 Q1 Sprint 1	GDM	Implementação da inicialização pela criação do YAML padrão e estrutura inicial.	US02, US03
	HCA	Implementação da estrutura base da CLI com *parsing*, do comando *help*. Configuração do ambiente de dev Docker.	US01, NF: Usabilidade da CLI
Agosto/2025 Q2 Sprint 2	GDM	Desenvolvimento do parser YAML para carregar a configuração e validação básica da estrutura.	US05, US14
	HCA	Implementação da instalação automática dos Git *hooks* durante o init.	US04
Setembro/2025 Q1 Sprint 3	GDM	Implementação da execução local de testes ao chamar “hookci run”, lendo *steps* do YAML.	US09
	HCA	Integração inicial com Docker através da execução de comandos de *steps* em contêineres. Montagem do repositório no contêiner.	US10, US18
Setembro/2025 Q2 Sprint 4	GDM	Implementação da execução de testes via Git *hooks*, integrando com a lógica de run.	US08

	HCA	Implementação do sistema de *logging*, verbosidade e suporte para imagem Docker/*Dockerfile*.	US11, US12, US16, US17
Outubro/2025 Q1 Sprint 5	GDM	Implementação de testes não essenciais na CI.	US15
	HCA	Implementação do modo de depuração em caso de falha nos testes.	US13
Outubro/2025 Q2 Sprint 6	GDM	Implementação da funcionalidade de migração de configuração através do comando “hookci migrate”.	US06, US07
	HCA	Criação da documentação com GNU Roff.	Documentação
Novembro/2025 Q1 Sprint 7	GDM	Configuração de pipeline CI/CD para o projeto HookCI. Criação de testes unitários.	CI/CD NF: Testes
	HCA	Pipeline de CI auto-testada pelo HookCI.	CI/CD
Novembro/2025 Q2 Sprint 8	GDM	Criação de Casos de Teste de Aceitação.	NF: Testes
	HCA	Empacotamento do sistema via PyInstaller e .deb.	CI/CD

Tabela 44. Cronograma de implementação

7.2 Processo de Implementação

O desenvolvimento do sistema HookCI será conduzido seguindo um processo iterativo e incremental, com inspiração em metodologias ágeis, para garantir flexibilidade e entregas contínuas de valor. A linguagem de programação primária escolhida para a implementação é Python, dada sua versatilidade e amplo suporte de bibliotecas para as funcionalidades planejadas.

Para assegurar a consistência do ambiente de desenvolvimento entre os membros da equipe e facilitar a replicação do ambiente para testes e integração, será utilizado Docker, permitindo a criação de um ambiente de desenvolvimento padronizado e isolado. O gerenciamento das dependências no desenvolvimento local do projeto Python será feito utilizando Poetry.

O trabalho será estruturado em sprints quinzenais. Cada sprint visa entregar um incremento funcional e testável do software. Reuniões curtas de planejamento e alinhamento serão realizadas no início e ao final de cada ciclo para discutir o progresso, os impedimentos e as próximas etapas. Esta gestão será organizada através do GitHub Projects. A utilização de um quadro Kanban, combinada com a definição de *sprints*, permite o controle das tarefas e do tempo, garantindo o cumprimento do

cronograma e a gestão adequada do escopo entre os membros da equipe. A Figura 31 demonstra o cronograma do projeto em seu fim.

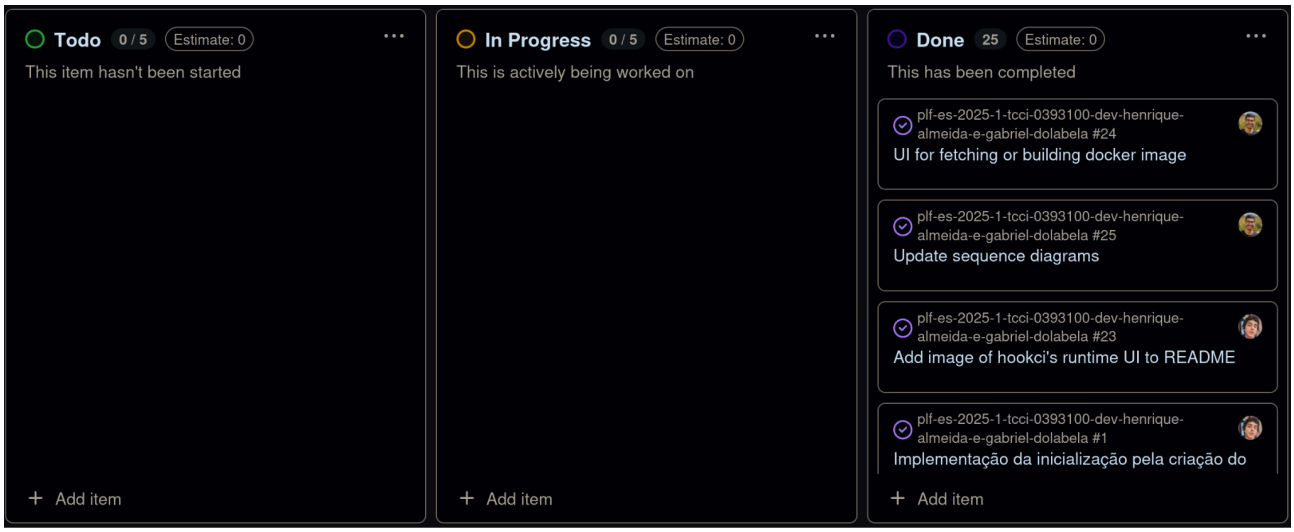

Figura 31. Quadro Kanban no GitHub Projects

A qualidade do produto será assegurada por uma estratégia de testes multifacetada: Testes Unitários serão implementados com pytest para validar o sistema de forma isolada; Testes de Integração verificarão as interações entre os módulos internos do HookCI e com os sistemas externos, como o Git, o Docker Engine e o sistema de arquivos. Para monitoramento da qualidade do código fonte, utilizou-se o SonarQube Cloud. A ferramenta foi integrada ao fluxo de CI, realizando análises estáticas a cada Pull Request para identificar *code smells*, bugs potenciais e duplicidade de código. A *Quality Gate* definida impõe zero novos problemas e cobertura mínima de 80%. Esta abordagem possibilitou alcançar uma cobertura de código de 100% sobre o total de 1.600 linhas de código fonte da aplicação, conforme ilustrado na Figura 29.

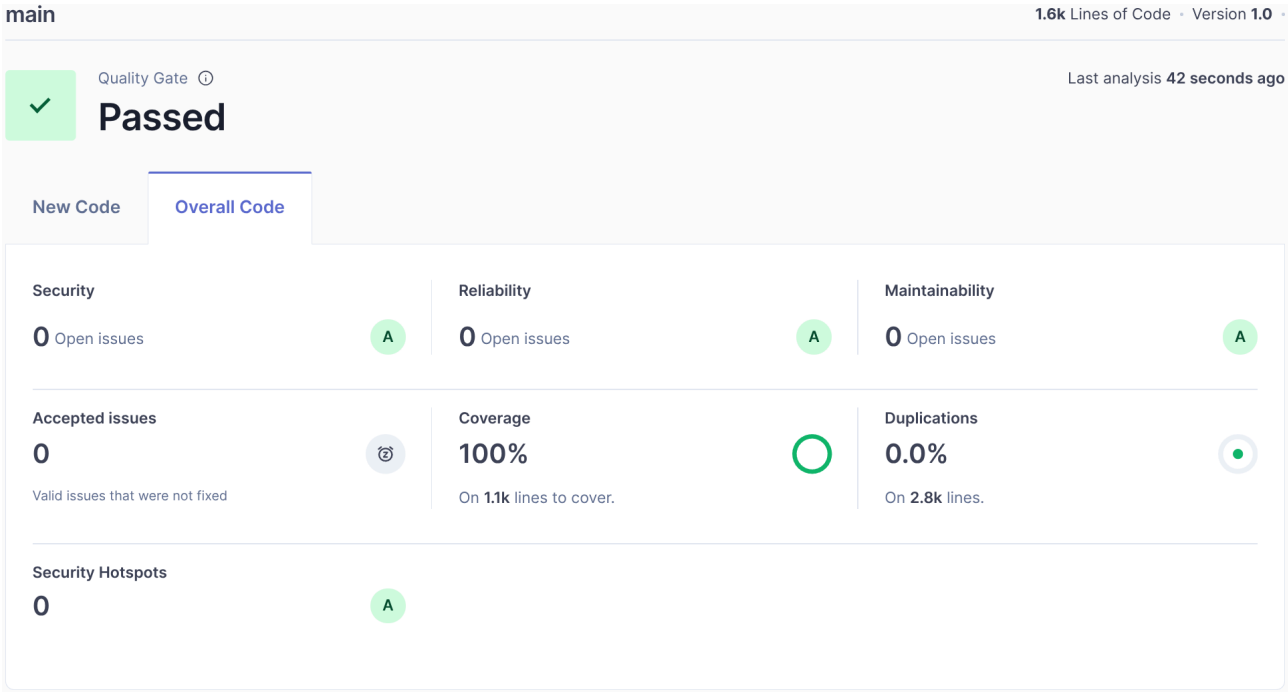

Figura 29. Captura de tela de uma análise SonarQube

Para automatizar e otimizar o ciclo de desenvolvimento do HookCI, será configurado uma *pipeline* de CI/CD com Github Actions. Esta será responsável pela execução automática dos testes, análise estática de código para garantir a conformidade com padrões de codificação, e a automação do processo de build e empacotamento a cada release. Ao decorrer do projeto, foram estabelecidas esteiras automatizadas que executam a suíte de testes e verificações de estilo a cada *push*, garantindo a integridade da *branch* principal. Adicionalmente, a automação do processo de *release* a cada *tag* permitiu a geração dos artefatos de distribuição, agilizando a entrega de novas versões. As Figuras 30 e 31 revelam, respectivamente, um exemplo de execução da esteira e uma *release* do Github criada automaticamente.

Figura 30. *Pipeline* de CI/CD no GitHub Actions.

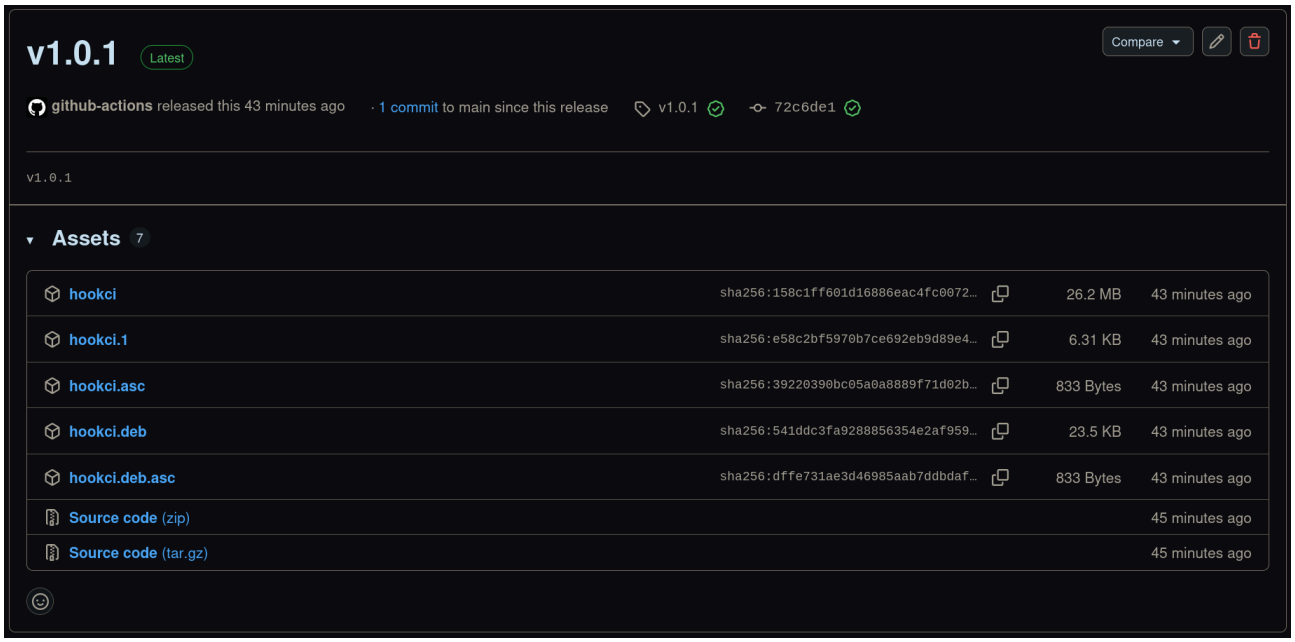

Figura 31. *Release* criada por esteira CI/CD com artefatos do sistema.

A documentação do projeto, incluindo este documento de especificação, será um artefato vivo, mantido atualizado ao longo de todo o ciclo de desenvolvimento. Adicionalmente, será elaborada uma documentação abrangente para o usuário final, através de GNU Roff, um arquivo README com instruções simples e mensagens de ajuda na CLI para facilitar a utilização da ferramenta.

Cada *release* do sistema incluirá três Artefatos. O primeiro é um arquivo ELF, gerado via PyInstaller, que encapsula a aplicação e todas as suas dependências, permitindo sua execução direta

em outros sistemas sem a necessidade de uma instalação prévia do interpretador Python nem de dependências. O segundo é um pacote DEB, destinado a distribuições baseadas em Debian, que facilita a instalação, atualização e remoção do HookCI. Por fim, uma manpage que constitui a documentação de referência da ferramenta, integrada ao sistema para consulta via terminal.

Além disso, foi desenvolvida uma extensão para o editor de código Visual Studio Code, com o objetivo de simplificar a instalação e o uso da ferramenta. A extensão encapsula o binário gerado pelo PyInstaller e realiza a instalação automática da ferramenta no ambiente assim que é ativada. Dessa forma, a configuração dos *hooks* do Git e a preparação do ambiente ocorrem de maneira transparente, eliminando passos manuais de configuração e garantindo que a validação local esteja operacional desde o primeiro momento.

8. Post-mortem

Esta seção apresenta uma análise reflexiva sobre o processo de desenvolvimento do sistema HookCI, destacando os pontos fortes, as dificuldades encontradas e os aprendizados técnicos obtidos pela equipe durante a execução do projeto.

8.1 Experiências Positivas

A adoção de uma arquitetura em camadas demonstrou-se uma decisão acertada. A separação clara entre *Presentation*, *Application*, *Domain* e *Infrastructure* facilitou a manutenção do código e permitiu que a lógica de negócios permanecesse isolada das complexidades das bibliotecas externas, como o cliente do Docker e os comandos do Git. Isso resultou em um código modular e de fácil leitura.

No que tange à interface, a escolha das bibliotecas Typer e Rich para a construção da CLI foi fundamental. Estas ferramentas permitiram a criação de uma interface com feedbacks visuais claros e tratamento de erros elegante, elevando a percepção de qualidade do produto final sem demandar esforço excessivo no desenvolvimento de componentes visuais.

8.2 Experiências Negativas

Um dos maiores desafios técnicos enfrentados foi a manipulação de *streams* de entrada e saída da API do Docker. A necessidade de demultiplexar os canais de saída e erro em tempo real, mantendo a sincronia com a interface do usuário, revelou-se mais complexa do que o estimado inicialmente. Houve dificuldades em tratar caracteres de controle e garantir que a saída dos testes fosse apresentada ao usuário de forma legível e imediata.

Além disso, a complexidade de criar testes automatizados para a camada de infraestrutura foi elevada. Simular o comportamento do *daemon* do Docker e subprocessos do sistema operacional exigiu a criação de *fixtures* complexas, o que, em alguns momentos, reduziu a velocidade de desenvolvimento.

8.3 Lições Aprendidas

O projeto reforçou a importância do princípio de Inversão de Dependência e do uso de Injeção de Dependência. A estruturação do código de forma que as camadas de alto nível não dependessem das implementações concretas de baixo nível foi crucial para viabilizar os testes unitários, permitindo a substituição dos serviços reais de Docker e Git por objetos simulados durante a validação da lógica de aplicação.

Por fim, ficou evidente que o tratamento de erros em ferramentas de CLI deve ser proativo e informativo. Diferente de sistemas *backend* onde *logs* de erro são suficientes, uma ferramenta de uso local precisa traduzir exceções técnicas em mensagens de ação claras para o usuário, orientando-o sobre como resolver o problema.

8.4 Repositório do Trabalho

O código fonte completo do sistema HookCI, incluindo a documentação técnica, os testes automatizados e os scripts de instalação, encontra-se disponível publicamente no seguinte repositório:
github.com/ICEI-PUC-Minas-PPLES-TI/plf-es-2025-1-tcci-0393100-dev-henrique-almeida-e-gabriel-dolabela