

---

**Gabriel Dolabela e Henrique Almeida**

gabriel.marques.1203633@sga.pucminas.br  
me@h3nc4.com

# Documento de Visão para o Sistema HookCI

**16 de Março de 2025**

*Proposta do(s) aluno(s) Gabriel Dolabela Marques e Henrique Carvalho Almeida ao curso de Engenharia de Software como projeto de Trabalho de Conclusão de Curso (TCC) sob orientação de conteúdo da professora Joana Gabriela Ribeiro de Souza e orientação acadêmica do professor Cleiton Silva Tavares.*

---

## OBJETIVOS

Criar uma ferramenta que integre a prática de Integração Contínua (CI, do inglês *Continuous Integration*) ao ambiente local de desenvolvimento. A solução permitirá que os testes na CI de um sistema sejam executados antes dos *commits* ou *pushes*, evitando que falhas e inconsistências sejam propagadas para o repositório remoto. Desta forma, o sistema visa aumentar a confiabilidade do código, reduzir o retrabalho e otimizar o uso dos recursos, contribuindo para um ciclo de desenvolvimento mais ágil e eficaz.

## ESCOPO

HookCI consiste em uma ferramenta para a execução da CI de forma local, integrando Docker e Git *hooks* para validar *commits* e *pushes*. Suas principais características são:

- Integração com Git: A ferramenta será instalada como parte do fluxo de trabalho do Git, utilizando *hooks* como *pre-commit* e *pre-push* para disparar a execução da CI.
- Execução da CI localmente: Em vez de esperar que os testes da CI sejam realizados em servidores remotos, o sistema permite que as validações ocorram na máquina do desenvolvedor, promovendo retorno imediato.
- Isolamento via Docker: Para garantir a consistência do ambiente de CI e evitar a interferência do ambiente local, a execução ocorrerá em containers Docker configurados conforme as necessidades do projeto.

- 
- Configuração flexível: Através de um arquivo do tipo YAML, os usuários poderão definir variáveis de ambiente, comandos para executar seus testes e parâmetros de execução, adaptando a ferramenta a diferentes contextos de desenvolvimento.
  - Interface de linha de comando (CLI, do inglês *Command Line Interface*): A interação com o sistema será realizada via CLI, possibilitando uma integração simples e eficiente ao fluxo de trabalho dos desenvolvedores.

A proposta se diferencia dos serviços de CI tradicionais, como GitHub Actions ou GitLab CI, pois traz o conceito de CI para o ambiente local, reduzindo o tempo de retorno e evitando a sobrecarga em servidores remotos. Assim, a ferramenta atua como uma camada preventiva que assegura que somente códigos que passam na CI sejam integrados ao repositório central.

Outra ferramenta semelhante é o Husky, que facilita a execução de *scripts* via Git *hooks*, mas roda os comandos diretamente no ambiente local. Nossa ferramenta pretende isolar a CI em Docker para garantir a consistência, oferecendo configuração via YAML e uma CLI dedicada, tornando o processo de CI local mais estruturado.

## FORA DO ESCOPO

Para manter o foco na solução do problema principal e evitar complexidades desnecessárias, foram definidos os seguintes itens como fora do escopo:

- Execução remota de CI: O sistema não prevê a criação ou o gerenciamento de servidores remotos para CI; a execução dos testes se restringe ao ambiente local.
- Interface gráfica: O foco será em uma interface YAML e CLI, não havendo, nesta versão, uma interface gráfica interativa para configuração ou monitoramento.
- Integração com múltiplos sistemas de versionamento: O sistema foi projetado para funcionar com Git, não sendo contemplada a integração com outros sistemas de versionamento.
- Geração de relatórios entre operações: Embora retornos de erro e estado sejam apresentados, a geração de relatórios e *dashboards* interativos está fora do escopo inicial.

## GESTORES, USUÁRIOS E OUTROS INTERESSADOS

Nome	Gabriel Dolabela Marques
Qualificação	Estudante
Responsabilidades	Desenvolvedor

---

Nome	Henrique Carvalho Almeida
Qualificação	Estudante
Responsabilidades	Desenvolvedor

## LEVANTAMENTO DE NECESSIDADES

Enumere as necessidades identificadas no ambiente de negócio e que justifiquem a criação do sistema. As necessidades devem ser numeradas e justificadas.

1. Validação Preventiva de Código. Evitar que *commits* e *pushes* com falhas ou erros sejam enviados para o repositório remoto, garantindo a integridade do código e diminuindo retrabalhos.
2. Retorno Imediato aos Desenvolvedores. Proporcionar uma resposta rápida quanto à qualidade do código, possibilitando correções imediatas e maior agilidade no processo de desenvolvimento.
3. Isolamento do Ambiente de CI. Utilizar Docker para garantir que a CI seja executada em um ambiente controlado e padronizado, independente das configurações locais dos desenvolvedores.
4. Flexibilidade na Configuração e Execução. Permitir que diferentes projetos e equipes customizem os parâmetros de execução da CI via arquivo YAML, variáveis de ambiente e interface CLI.
5. Especificar imagens docker para runtime. Permitir que a ferramenta utilize uma imagem Docker pré-definida ou um Dockerfile para criar o ambiente de testes.
6. Pré-configuração na inicialização da ferramenta no repositório. Automatizar a instalação de Git *hooks*, a geração de arquivos de configuração e a verificação de dependências essenciais para facilitar o setup inicial.
7. Suporte ao *workflow* de desenvolvimento. Adaptar o comportamento da ferramenta ao fluxo de trabalho de times, incluindo regras por tipo de branch, prefixos de *commits* e controle mais refinado sobre o que e quando será testado.
8. Migração das configurações entre versões. Viabilizar a detecção de versões antigas da configuração e oferecer migração automática ou manual, garantindo a continuidade do uso da ferramenta em projetos que evoluem.
9. Otimização do Tempo de Execução. Reduzir o tempo necessário para executar a CI com paralelismo, ordenação de testes e limites de execução, otimizando o retorno ao usuário.

- 
10. Depuração da CI. Fornecer mecanismos de depuração detalhados para auxiliar na investigação de falhas, incluindo logs, console interativo e diferentes níveis de verbosidade.

## FUNCIONALIDADES DO PRODUTO

Necessidade: Validação Preventiva de Código	
Funcionalidade	Categoria
1. Integração com Git <i>hooks</i> ( <i>pre-commit</i> e <i>pre-push</i> ) para disparo automático da CI.	Crítico
2. A ferramenta deve capturar e armazenar o código de saída, <i>stdout</i> e <i>stderr</i> de cada comando executado..	Crítico
3. Bloqueio de <i>commits</i> ou <i>pushes</i> em caso de falha na CI.	Crítico
4. Mecanismo de bloqueio condicional para permitir testes não essenciais.	Útil

Necessidade: Flexibilidade na Configuração e Execução	
Funcionalidade	Categoria
1. Utilização de arquivo YAML para definição de testes e ambientes.	Crítico
2. Permitir evocação da CLI pelo usuário para execução da CI.	Importante
3. Permitir configuração para grupos de testes que não bloqueiam a <i>commit</i> ou <i>push</i> .	Útil
4. Permitir a desativação temporária da CI via CLI.	Útil
5. Permitir injeção de variáveis de ambiente via ambiente, YAML e <i>.env</i>	Útil

Necessidade: Retorno Imediato aos Desenvolvedores	
Funcionalidade	Categoria
1. Log dos resultados da CI apresentados na interface CLI em caso de Falha	Crítico

2. Arquivo de logs completo com todo o output dos testes executados	Importante
3. Adicionar cores e símbolos para destacar erros e avisos.	Útil

<b>Necessidade:</b> Isolamento do Ambiente de CI	
<b>Funcionalidade</b>	<b>Categoria</b>
1. Execução da CI em containers Docker, garantindo ambiente padronizado.	Crítico
2. Montagem do diretório atual do repositório no container para execução da CI.	Crítico

<b>Necessidade:</b> Especificar imagens docker para runtime da CI	
<b>Funcionalidade</b>	<b>Categoria</b>
1. Usuário especifica uma imagem docker através de nome e versão.	Crítico
2. Usuário especifica uma imagem docker através de Dockerfile.	Importante

<b>Necessidade:</b> Pré-configuração na inicialização da ferramenta no repo	
<b>Funcionalidade</b>	<b>Categoria</b>
1. Chamada por comando para se inicializar no repositório.	Crítico
2. Instalação automatizada de Git <i>hooks</i> durante a configuração inicial do projeto.	Crítico
3. Gerar automaticamente um arquivo de configuração YAML padrão.	Crítico
4. Verificar dependências essenciais e alertar sobre falhas e faltas no ambiente.	Importante
5. Customização de configurações iniciais através de opções CLI.	Importante

<b>Necessidade:</b> Suporte ao <i>workflow</i> de desenvolvimento	
Funcionalidade	Categoria
1. Definir regras diferentes de bloqueio para diferentes tipos de branch.	Importante
2. Permitir regras de filtro para disparo de testes por <i>branches</i> . (ex: <i>feature/*</i> , <i>bugfix/*</i> )	Importante
3. Permitir regras de filtro para disparo de testes por <i>commits</i> . (ex: <i>docs:*</i> , <i>feat:*</i> )	Importante

<b>Necessidade:</b> Migração das configurações caso uma nova versão (compatível com a anterior) da ferramenta seja detectada	
Funcionalidade	Categoria
1. Verificar a versão atual da configuração YAML.	Útil
2. Validar compatibilidade com a nova versão.	Útil
3. Aplicar migração automática ou sugerir intervenção manual, caso a versão seja incompatível.	Útil

<b>Necessidade:</b> Otimização do Tempo de Execução	
Funcionalidade	Categoria
1. Executar CI de forma paralela.	Importante
2. Permitir definição da ordem de execução dos testes via configuração.	Útil
3. Permitir definição de tempo máximo para um teste	Útil

<b>Necessidade:</b> Depuração da CI	
Funcionalidade	Categoria
1. Permitir parâmetros de verbosidade na CLI.	Útil
2. Console de depuração aberto para o usuário em caso de erro ou fim de execução quando a depuração está ativa.	Útil

---

3. Log de ambiente com variáveis, caminho de execução e comandos executados.	Útil
--	------

## INTERLIGAÇÃO COM OUTROS SISTEMAS

O sistema será interligado com Git por meio de sua ferramenta de Git *hooks* e com Docker para a execução da CI.

## RESTRIÇÕES

- Restrições do Produto
  - O sistema deverá ser implementado utilizando Python e Shell.
  - O sistema deverá ser entregue via um único binário executável.
  - O retorno dos testes deverá ser otimizado, com um limite de 2 segundos após sua execução.
- Requisitos de Portabilidade
  - O sistema deverá ser executado em uma distribuição GNU/Linux com Docker e Git.
  - O sistema deverá ter, ao menos, a versão 27 do Docker Engine e 2 do Git.
- Requisitos de Segurança
  - Os binários distribuídos pelo projeto devem ser assinados com chaves criptográficas OpenPGP.
- Requisitos de Licenciamento
  - O usuário estará protegido pela *GNU General Public License v3* ou versão posterior.
  - O código do sistema será disponibilizado de forma aberta.

## DOCUMENTAÇÃO

Toda a documentação detalhada, incluindo instruções avançadas, exemplos de uso, opções de configuração e manutenção, estará disponível por meio de *manpages* escritas em GNU roff. Estas servirão como referência robusta para usuários que desejem explorar todas as funcionalidades do sistema.

No repositório, um arquivo README.md conterá uma documentação simplificada e objetiva. Este guia oferecerá as informações essenciais sobre instalação, configuração básica e utilização imediata da ferramenta, facilitando o acesso rápido e intuitivo aos principais recursos do sistema.