Sistema de Gerenciamento de Tarefas

Plano de Gerenciamento de Configuração

Versão 1.0

[Nota: O gabarito a seguir é fornecido para utilização com o Rational Unified Process. O texto em azul exibido entre colchetes e em itálico (style=InfoBlue) foi incluído para orientar o autor e deve ser excluído antes da publicação do documento. Um parágrafo digitado após esse estilo será automaticamente definido como normal (style=Body Text).]

[Para personalizar campos automáticos no Microsoft Word (que exibem um segundo plano cinza quando selecionados), selecione File>Properties e substitua os campos Title, Subject e Company pelas informações apropriadas para este documento. Depois de fechar o diálogo, os campos automáticos podem ser atualizados no documento inteiro, selecionando Edit>Select All (ou Ctrl-A) e pressionando F9 ou simplesmente clique no campo e pressione F9. Esse procedimento deverá ser executado separadamente para os Cabeçalhos e Rodapés. Alt-F9 alterna entre a exibição de nomes de campos e do conteúdo dos campos. Consulte a Ajuda do Word para obter informações adicionais sobre como trabalhar com campos.]

Histórico da Revisão

Data	**Versão**	**Descrição**	**Autor**
02/12/2025	1.0	Criação inicial do documento	Equipe de Desenvolvimento

Índice

Plano de Gerenciamento de Configuração

1. Introdução

*[A introdução do **Plano de Gerenciamento de Configuração** fornece uma visão geral de todo o documento. Ela inclui o objetivo, o escopo, as definições, os acrônimos, as abreviações, as referências e a visão geral deste **Plano de Gerenciamento de Configuração**.]*

1.1. Objetivo

Este Plano de Gerenciamento de Configuração (PGC) estabelece os procedimentos, responsabilidades e ferramentas necessárias para gerenciar as configurações do Sistema de Gerenciamento de Tarefas ao longo de todo o seu ciclo de vida. O objetivo principal é garantir que todas as versões dos componentes do software sejam identificadas, controladas, rastreadas e auditadas de forma sistemática e eficiente.

1.2. Escopo

Este plano de gerenciamento de configuração aplica-se ao projeto Sistema de Gerenciamento de Tarefas, uma aplicação web desenvolvida em Python/Flask. O escopo inclui:

• Código-fonte: Todos os arquivos Python, HTML, CSS, JavaScript

• Documentação: README, documentação técnica, planos

• Artefatos de build: Dockerfiles, scripts de build

• Testes: Scripts de teste automatizados • Configurações: Arquivos de configuração de CI/CD

1.3. Definições, Acrônimos e Abreviações

Termo	**Definição**
CM	Configuration Management (Gerenciamento de Configuração)
CI/CD	Continuous Integration / Continuous Delivery
SCM	Source Control Management
Baseline	Versão estável e aprovada de um item de configuração
Branch	Ramificação no sistema de controle de versão
Commit	Operação de salvar mudanças no repositório
Tag	Marcação de uma versão específica no repositório
Build	Processo de compilação/construção do software
Deploy	Processo de implantação do software

1.4. Referências

• Rational Unified Process (RUP) - Template de Plano de Gerenciamento de Configuração

• IEEE Std 828-2012 - Standard for Configuration Management

• Git Documentation - https://git-scm.com/doc

• GitHub Actions Documentation

1.5. Visão Geral

Este documento está organizado da seguinte forma: Seção 1 - Introdução e contexto; Seção 2 - Estrutura organizacional e ferramentas; Seção 3 - Processos detalhados de gerenciamento de configuração; Seção 4 - Marcos e milestones; Seção 5 - Treinamento e recursos.

2. Gerenciamento de Configuração de Software

2.1. Organização, Responsabilidades e Interfaces

O projeto será gerenciado por uma equipe de desenvolvimento que seguirá as seguintes responsabilidades: Gerente de Projeto
• Aprovação de baselines principais
• Decisões sobre releases e versões Desenvolvedores
• Desenvolvimento e manutenção do código-fonte
• Criação de branches para features
• Realização de commits seguindo padrões estabelecidos Responsável por CM
• Manutenção do repositório • Criação de tags e releases
• Gerenciamento de branches principais

2.2. Ferramentas, Ambiente e Infra-estrutura

Ferramenta	**Versão**	**Propósito**
Git	2.x+	Controle de versão
GitHub	-	Repositório remoto
Python	3.11	Linguagem de programação
Flask	3.0.0	Framework web
pytest	7.4.3	Framework de testes
Docker	Latest	Containerização
GitHub Actions	-	CI/CD Pipeline

3. O Programa de Gerenciamento de Configuração

3.1. Identificação da Configuração

3.1.1. Métodos de Identificação

Sistema de Versionamento Semântico (SemVer)

Formato: MAJOR.MINOR.PATCH

• MAJOR: Mudanças incompatíveis na API

• MINOR: Novas funcionalidades compatíveis

• PATCH: Correções de bugs compatíveis

3.1.2. Linhas de Base do Projeto

Branch Principal: main ou master

• Contém código estável e testado

• Protegida contra pushes diretos

Branches de Desenvolvimento:

• feature/*: Novas funcionalidades

• bugfix/*: Correções de bugs

• hotfix/*: Correções urgentes

3.2. Configuração e Controle de Alterações

3.2.1. Processamento e Aprovação de Controles de Mudanças

Processo de Mudança:
1. Criação de Branch
2. Desenvolvimento com commits frequentes
3. Testes Locais
4. Push e Pull Request
5. Merge após aprovação e testes

3.2.2. CCB (Conselho de Controle de Mudanças)

O Conselho de Controle de Mudanças (CCB) é composto pelos seguintes membros:

• **Gerente de Projeto**: Responsável pela aprovação final de mudanças significativas • **Responsável por CM**: Coordena o processo de revisão e aprovação • **Desenvolvedor Líder**: Avalia o impacto técnico das mudanças

Procedimentos do CCB:

1. **Solicitação de Mudança**: Desenvolvedor cria Pull Request no GitHub
2. **Revisão Automática**: Pipeline CI/CD executa testes automaticamente
3. **Revisão de Código**: Pelo menos um desenvolvedor revisa o código
4. **Aprovação**: Após aprovação e testes bem-sucedidos, o merge é autorizado
5. **Registro**: Todas as mudanças são registradas no histórico do Git

Critérios de Aprovação: • Todos os testes devem passar (unidade, integração e aceitação) • Código revisado e aprovado por pelo menos um membro da equipe • Sem conflitos com a branch principal • Documentação atualizada quando necessário

3.3. Contabilidade do Status de Configuração

3.3.1. Armazenamento de Mídia do Projeto e Processo de Release

Políticas de Retenção:

• **Repositório Git**: Mantido online no GitHub permanentemente • **Backups**: GitHub mantém backup automático de todos os repositórios • **Tags de Release**: Versões estáveis são marcadas com tags Git (ex: v1.0.0) • **Artefatos de Build**: Imagens Docker são armazenadas e versionadas • **Documentação**: Mantida no repositório e em formato PDF

Plano de Backup e Recuperação:

• Backup automático pelo GitHub (redundância em múltiplos datacenters) • Clones locais mantidos pelos desenvolvedores • Imagens Docker podem ser reconstruídas a partir do código-fonte • Documentação versionada no repositório Git

Processo de Release:

1. **Preparação**: Código estável na branch main, todos os testes passando
2. **Versionamento**: Criação de tag seguindo SemVer (ex: v1.0.0)
3. **Build**: Pipeline CI/CD gera imagem Docker automaticamente
4. **Documentação**: Release notes descrevem mudanças e melhorias

5. **Distribuição**:
    - Código-fonte disponível via GitHub
    - Imagem Docker disponível para deploy
    - Documentação atualizada no repositório

Conteúdo do Release: • Código-fonte completo • Dockerfile para containerização • Documentação de instalação e uso • Changelog com todas as mudanças • Instruções de deploy

Público-alvo: • Desenvolvedores que irão contribuir ou fazer fork do projeto • Usuários finais que irão fazer deploy da aplicação • Equipe de manutenção e suporte

3.3.2. Relatórios e Auditorias

Relatórios de Configuração:

Os relatórios são gerados automaticamente pelo pipeline CI/CD e incluem:

• **Relatório de Cobertura de Testes**: Gerado pelo pytest-cov, mostra porcentagem de código testado • **Relatório de Build**: Status de compilação e build da aplicação • **Relatório de Testes**: Resultados de todos os testes (unidade, integração, aceitação) • **Histórico de Commits**: Log completo de todas as mudanças no repositório • **Relatório de Deploy**: Status da implantação e artefatos gerados

Formato dos Relatórios:

• **HTML**: Relatórios de cobertura disponíveis em htmlcov/ • **XML**: Relatórios de cobertura em formato XML para integração com ferramentas • **Terminal**: Saída formatada no console durante execução do pipeline • **GitHub Actions**: Interface web do GitHub mostra status de cada execução

Auditorias de Configuração:

Auditorias são realizadas através de:

1. **Revisão de Pull Requests**: Cada mudança é revisada antes do merge
2. **Execução Automática de Testes**: Pipeline valida todas as mudanças
3. **Verificação de Conformidade**: Código deve seguir padrões estabelecidos
4. **Rastreabilidade**: Histórico completo no Git permite rastrear todas as mudanças

Indicadores de Qualidade:

Relatórios Baseados em Tempo: • Tempo médio de resolução de issues: Monitorado via GitHub Issues • Tempo entre detecção e correção de bugs: Rastreado através de commits e tags • Tempo de ciclo de desenvolvimento: Desde criação de branch até merge

Relatórios Baseados em Contagem: • Número de testes: 43 testes automatizados (unidade, integração, aceitação) • Cobertura de código: Meta de 80%+ de cobertura • Defeitos por prioridade: Classificados como crítico, alto, médio, baixo • Status de correção: Aberto, em progresso, resolvido, fechado

Relatórios de Tendência: • Número cumulativo de defeitos detectados e corrigidos ao longo do tempo • Taxa de sucesso dos testes: Percentual de testes passando em cada execução • Intervalo de qualidade: Razão entre defeitos abertos e fechados • Velocidade de desenvolvimento: Commits por período, features entregues

Frequência de Relatórios: • **Tempo Real**: Status do pipeline disponível a cada commit • **Diário**: Resumo de atividades do dia via GitHub Insights • **Semanal**: Revisão de métricas de qualidade e progresso • **Por Release**: Relatório completo antes de cada release

4. Marcos

Marcos do Projeto:

Marco	**Data**	**Descrição**	**Entregáveis**
Início do Projeto	01/12/2025	Inicialização do repositório e estrutura base	Repositório Git criado
Baseline Inicial	02/12/2025	Primeira versão funcional com testes	Código-fonte, testes, Dockerfile
Pipeline CI/CD	02/12/2025	Configuração completa do pipeline	Script de implantação (.github/workflows/ci-cd.yml)
Release v1.0.0	02/12/2025	Primeira release estável	Tag v1.0.0, documentação completa

Marcos de CM (Gerenciamento de Configuração):

• **Baseline Inicial**: Primeira versão estável do código (02/12/2025) • **Configuração do Pipeline**: Implementação completa do CI/CD (02/12/2025) • **Primeira Release**: Tag v1.0.0

criada após validação completa (02/12/2025) • **Documentação Completa**: Plano de CM e documentação técnica finalizados (02/12/2025)

Atualização do Plano de Gerenciamento de Configuração:

Este plano deve ser atualizado nas seguintes situações:

• Mudanças significativas no processo de desenvolvimento • Adição de novas ferramentas ou tecnologias • Alterações na estrutura organizacional da equipe • Mudanças nos procedimentos de release • A cada release major (versão X.0.0) • Quando solicitado pelo Gerente de Projeto ou Responsável por CM

Próxima Revisão Planejada: Após primeira release ou quando houver mudanças significativas no processo.

5. Treinamento e Recursos

Treinamento para Desenvolvedores:

Uso Básico de Git: • Comandos essenciais: clone, add, commit, push, pull • Criação e gerenciamento de branches • Resolução de conflitos • Uso de tags para versionamento

Processo de Pull Request no GitHub: • Criação de branch para feature/bugfix • Desenvolvimento com commits frequentes e descritivos • Criação de Pull Request com descrição clara • Revisão de código e aprovação • Merge após aprovação e testes bem-sucedidos

Execução de Testes Localmente: • Instalação de dependências: `pip install -r requirements.txt` • Execução de todos os testes: `pytest` • Execução de testes específicos: `pytest tests/test_models.py` • Com cobertura: `pytest --cov=. --cov-report=html` • Verificação antes de commit: Sempre executar testes localmente

Uso do Docker: • Build da imagem: `docker build -t task-manager .` • Execução do container: `docker run -p 5000:5000 task-manager` • Verificação de logs: `docker logs <container_id>`

Recursos Disponíveis: • Documentação do Git: https://git-scm.com/doc • Documentação do GitHub: https://docs.github.com • Documentação do Flask: https://flask.palletsprojects.com • Documentação do pytest: https://docs.pytest.org • Documentação do Docker: https://docs.docker.com • Documentação do GitHub Actions: https://docs.github.com/en/actions

6. Controle de Software do Subfornecedor e do Fornecedor

Dependências Externas:

O projeto utiliza as seguintes bibliotecas e ferramentas de terceiros:

Bibliotecas Python (requirements.txt): • Flask 3.0.0 - Framework web • pytest 7.4.3 - Framework de testes • pytest-cov 4.1.0 - Cobertura de testes • requests 2.31.0 - Cliente HTTP • reportlab 4.0.7 - Geração de PDFs • python-pptx 0.6.23 - Manipulação de apresentações

Ferramentas e Serviços: • Git - Sistema de controle de versão • GitHub - Repositório remoto e CI/CD • Docker - Containerização • Python 3.11 - Linguagem de programação

Processo de Incorporação:

1. **Identificação**: Dependências são identificadas e documentadas no requirements.txt
2. **Versionamento**: Versões específicas são fixadas para garantir reprodutibilidade
3. **Validação**: Dependências são testadas durante o processo de build
4. **Atualização**: Atualizações de dependências seguem o mesmo processo de Pull Request
5. **Auditoria**: Dependências são revisadas periodicamente para segurança e atualizações

Políticas de Uso:

• Todas as dependências devem ser de código aberto ou ter licenças compatíveis • Versões devem ser fixadas no requirements.txt para garantir builds reproduzíveis • Atualizações de dependências devem passar por todos os testes antes do merge • Vulnerabilidades conhecidas devem ser corrigidas imediatamente • Licenças de dependências devem ser compatíveis com a licença do projeto

Monitoramento:

• GitHub Dependabot pode ser configurado para alertar sobre atualizações e vulnerabilidades • Revisão periódica de dependências para remover não utilizadas • Verificação de licenças antes de adicionar novas dependências