PROJETO A3: DOCUMENTAÇÃO DE DESENVOLVIMENTO DE SOFTWARE

GERENCIADOR DE TAREFAS

Universidade: Universidade São Judas Tadeu

Disciplina: UC – Gestão e Qualidade de Software

Professor: Prof. Calvetti

Data de Entrega: 11/06/2025

Integrantes: Bruno Rodrigues Reis

RA: 8222243147

RESUMO EXECUTIVO

Este documento apresenta a documentação completa do desenvolvimento de um sistema de gerenciamento de tarefas, desenvolvido como projeto hipotético para demonstrar a aplicação de conceitos de engenharia de software, gestão de qualidade e planejamento de testes. O projeto abrange desde o planejamento inicial até a implementação e validação do sistema, incluindo cronograma detalhado de testes, casos de teste abrangentes e estratégias de gestão de configuração.

O sistema "Gerenciador de Tarefas" foi desenvolvido utilizando Flask como framework backend e tecnologias web modernas para o frontend, demonstrando a aplicação prática dos conceitos estudados na disciplina. A documentação segue as melhores práticas da engenharia de software.

SUMÁRIO

---

1. INTRODUÇÃO

1.1. Contexto e Justificativa

O desenvolvimento de software de qualidade requer a aplicação sistemática de metodologias, processos e ferramentas que garantam não apenas a funcionalidade do produto final, mas também sua confiabilidade, manutenibilidade e adequação aos requisitos estabelecidos. Conforme destacado por Pressman e Maxim [1], a engenharia de software é uma disciplina que aplica princípios científicos e de engenharia ao desenvolvimento de sistemas de software, visando a criação de produtos de alta qualidade de forma econômica e eficiente.

Este projeto tem como objetivo demonstrar a aplicação prática dos conceitos fundamentais de gestão e qualidade de software através do desenvolvimento de um sistema hipotético de gerenciamento de tarefas. A escolha deste domínio de aplicação se justifica pela sua simplicidade conceitual, que permite focar nos aspectos metodológicos e de qualidade sem a complexidade de regras de negócio elaboradas, e pela sua relevância prática, uma vez que sistemas de gerenciamento de tarefas são amplamente utilizados em diversos contextos organizacionais e pessoais.

1.2. Objetivos do Projeto

O objetivo principal deste projeto é elaborar uma documentação completa de desenvolvimento de software que demonstre a aplicação dos conceitos estudados na disciplina de Gestão e Qualidade de Software. Os objetivos específicos incluem:

Objetivo Geral: Desenvolver uma documentação abrangente que demonstre a aplicação de metodologias de engenharia de software, desde o planejamento até a validação, utilizando um sistema de gerenciamento de tarefas como caso de estudo.

Objetivos Específicos: - Elaborar um plano de projeto detalhado que inclua planejamento, escopo, recursos e estimativas - Desenvolver um documento de requisitos que especifique claramente as funcionalidades do sistema - Criar um plano de testes abrangente com cronograma, estratégias e casos de teste - Implementar um protótipo funcional do sistema para validação dos conceitos - Estabelecer estratégias de

gestão de configuração adequadas ao projeto - Demonstrar a aplicação prática das referências bibliográficas da disciplina

1.3. Metodologia Aplicada

A metodologia adotada neste projeto baseia-se nos princípios da engenharia de software moderna, conforme descrito por Sommerville [2], que enfatiza a importância de processos bem definidos e documentados. O desenvolvimento seguiu uma abordagem iterativa e incremental, permitindo refinamentos contínuos da documentação e do produto.

As fases do projeto foram organizadas de acordo com os marcos principais do desenvolvimento de software: análise e planejamento, especificação de requisitos, design e implementação, testes e validação, e gestão de configuração. Cada fase foi documentada de forma detalhada, seguindo as melhores práticas estabelecidas na literatura especializada.

1.4. Estrutura do Documento

Este documento está organizado de forma a apresentar todos os artefatos exigidos pelo Projeto A3, seguindo uma sequência lógica que reflete o ciclo de vida do desenvolvimento de software. Cada seção aborda aspectos específicos do projeto, desde o planejamento inicial até a gestão de configuração, proporcionando uma visão completa e integrada do processo de desenvolvimento.

A estrutura adotada permite tanto a leitura sequencial quanto a consulta específica de seções individuais, facilitando o uso do documento como referência para projetos futuros. Todas as seções incluem referências às fontes bibliográficas relevantes, estabelecendo conexões claras entre a teoria estudada e a prática aplicada.

---

2. PLANO DE PROJETO

2.1. Planejamento do Projeto

O planejamento do projeto "Gerenciador de Tarefas" foi desenvolvido com base nos princípios estabelecidos por Pressman e Maxim [1], que destacam a importância de um planejamento cuidadoso como fundamento para o sucesso de qualquer projeto de software. O planejamento abrange não apenas os aspectos técnicos do desenvolvimento, mas também a gestão de recursos, cronogramas e riscos associados ao projeto.

A abordagem de planejamento adotada segue o modelo de processo incremental, que permite entregas parciais funcionais ao longo do desenvolvimento, facilitando a validação contínua dos requisitos e a identificação precoce de problemas. Esta estratégia é particularmente adequada para projetos de pequeno a médio porte, como o sistema proposto, onde a flexibilidade e a capacidade de adaptação são fundamentais.

O projeto foi estruturado em fases bem definidas, cada uma com objetivos específicos e critérios de conclusão claros. Esta organização facilita o controle do progresso e permite ajustes no planejamento conforme necessário, mantendo o foco na qualidade do produto final. As fases principais incluem análise e design, implementação, testes e documentação, cada uma contribuindo para a construção gradual de um sistema robusto e bem documentado.

2.2. Escopo do Projeto

O escopo do projeto "Gerenciador de Tarefas" foi definido de forma a equilibrar a simplicidade necessária para um projeto acadêmico com a complexidade suficiente para demonstrar os conceitos fundamentais de engenharia de software. Conforme recomendado por Sommerville [2], a definição clara do escopo é essencial para evitar o crescimento descontrolado dos requisitos e manter o projeto dentro dos limites estabelecidos.

Funcionalidades Incluídas no Escopo:

O sistema incluirá funcionalidades básicas de gerenciamento de tarefas que são essenciais para demonstrar os conceitos de desenvolvimento e teste de software. A funcionalidade de criação de tarefas permite aos usuários adicionar novas atividades ao sistema, especificando um título obrigatório e uma descrição opcional. Esta funcionalidade demonstra conceitos importantes como validação de entrada, persistência de dados e feedback ao usuário.

A visualização de tarefas proporciona uma interface clara e organizada para que os usuários possam consultar todas as suas atividades. Esta funcionalidade envolve aspectos de design de interface, organização de dados e experiência do usuário, elementos fundamentais em qualquer sistema de software moderno.

A marcação de conclusão de tarefas permite aos usuários indicar quando uma atividade foi completada, incluindo a possibilidade de reabrir tarefas previamente marcadas como concluídas. Esta funcionalidade demonstra conceitos de atualização de estado, persistência de mudanças e feedback visual ao usuário.

A exclusão de tarefas oferece aos usuários a capacidade de remover permanentemente atividades que não são mais necessárias. Esta funcionalidade inclui mecanismos de

confirmação para evitar exclusões acidentais, demonstrando princípios de design defensivo e experiência do usuário.

Limitações e Exclusões do Escopo:

Para manter o projeto dentro de limites gerenciáveis, algumas funcionalidades avançadas foram deliberadamente excluídas do escopo. Não serão implementadas funcionalidades de autenticação e autorização de usuários, mantendo o sistema como uma aplicação de usuário único. Esta decisão simplifica significativamente a arquitetura e permite focar nos aspectos fundamentais de desenvolvimento e teste.

Funcionalidades de colaboração, como compartilhamento de tarefas entre usuários ou atribuição de responsabilidades, também foram excluídas. Embora sejam valiosas em sistemas reais, estas funcionalidades introduziriam complexidade adicional que não é necessária para os objetivos educacionais do projeto.

Recursos avançados de organização, como categorização de tarefas, definição de prioridades ou estabelecimento de prazos, não fazem parte do escopo inicial. Estas funcionalidades poderiam ser consideradas em versões futuras do sistema, mas não são essenciais para demonstrar os conceitos fundamentais abordados na disciplina.

2.3. Recursos Necessários

A identificação e alocação adequada de recursos é fundamental para o sucesso de qualquer projeto de software. Conforme destacado por Pressman e Maxim [1], a gestão eficaz de recursos envolve não apenas a identificação das necessidades técnicas, mas também a consideração de fatores humanos, temporais e organizacionais.

Recursos Humanos:

O projeto requer uma equipe multidisciplinar com competências específicas em diferentes áreas do desenvolvimento de software. O papel de desenvolvedor backend é responsável pela implementação da lógica de negócio, integração com banco de dados e criação das APIs necessárias para o funcionamento do sistema. Este profissional deve ter conhecimento sólido em Python e Flask, além de experiência com bancos de dados relacionais.

O desenvolvedor frontend é responsável pela criação da interface do usuário, implementação da experiência do usuário e integração com as APIs backend. Este profissional deve dominar tecnologias web modernas, incluindo HTML5, CSS3 e JavaScript, além de ter conhecimento em design responsivo e acessibilidade.

O analista de testes desempenha papel crucial na garantia da qualidade do software, sendo responsável pela elaboração de casos de teste, execução de testes funcionais e

não funcionais, e documentação dos resultados. Este profissional deve ter conhecimento profundo em metodologias de teste e ferramentas de automação.

Recursos Tecnológicos:

A infraestrutura tecnológica necessária para o projeto inclui ferramentas de desenvolvimento, ambientes de teste e sistemas de controle de versão. O ambiente de desenvolvimento deve incluir editores de código modernos com suporte a debugging e integração com sistemas de controle de versão.

O framework Flask foi escolhido como base para o desenvolvimento backend devido à sua simplicidade, flexibilidade e ampla documentação disponível. Flask permite desenvolvimento rápido de aplicações web sem impor estruturas rígidas, sendo ideal para projetos de pequeno a médio porte.

Para o frontend, foram selecionadas tecnologias web padrão (HTML, CSS, JavaScript) que garantem compatibilidade ampla e facilidade de manutenção. Esta escolha evita dependências complexas e permite foco nos conceitos fundamentais de desenvolvimento.

O banco de dados SQLite foi escolhido para persistência de dados devido à sua simplicidade de configuração e adequação para projetos de desenvolvimento e teste. SQLite não requer instalação de servidor separado e oferece todas as funcionalidades necessárias para o projeto.

Recursos de Infraestrutura:

A infraestrutura de desenvolvimento inclui sistemas de controle de versão para gerenciamento de código fonte, ambientes de desenvolvimento isolados para cada desenvolvedor, e ambientes de teste que simulem as condições de produção.

O sistema de controle de versão Git, integrado com plataforma GitHub, proporciona rastreabilidade completa das mudanças no código, facilitando a colaboração entre desenvolvedores e a gestão de diferentes versões do sistema.

2.4. Estimativas de Projeto

A estimativa precisa de esforço e cronograma é um dos aspectos mais desafiadores do gerenciamento de projetos de software. Conforme observado por Pressman e Maxim [1], as estimativas devem ser baseadas em dados históricos quando disponíveis, e em técnicas de decomposição e analogia quando dados históricos não estão disponíveis.

Estimativas de Esforço:

A fase de análise e design foi estimada em 8 horas de trabalho, incluindo a definição detalhada dos requisitos, criação da arquitetura do sistema e especificação das interfaces. Esta estimativa considera a necessidade de documentação detalhada e revisões para garantir a qualidade das especificações.

A implementação do sistema foi estimada em 24 horas, distribuídas entre desenvolvimento backend (14 horas) e frontend (10 horas). A estimativa do backend inclui criação das APIs, implementação da lógica de negócio e integração com banco de dados. A estimativa do frontend abrange criação da interface do usuário, implementação da experiência do usuário e integração com as APIs.

A fase de testes foi estimada em 16 horas, incluindo criação de casos de teste (6 horas), execução de testes funcionais (8 horas) e testes de integração (2 horas). Esta estimativa considera tanto testes manuais quanto automatizados, garantindo cobertura adequada das funcionalidades.

A documentação foi estimada em 12 horas, incluindo elaboração da documentação técnica, manuais do usuário e relatórios de teste. Esta estimativa reflete a importância da documentação para a manutenibilidade e transferência de conhecimento.

Estimativas de Cronograma:

O cronograma total do projeto foi estimado em 4 semanas, considerando dedicação parcial da equipe e necessidade de revisões e refinamentos. A primeira semana é dedicada à análise e design, estabelecendo as bases sólidas para o desenvolvimento.

A segunda e terceira semanas são focadas na implementação, permitindo desenvolvimento iterativo com validações contínuas. Esta abordagem facilita a identificação precoce de problemas e permite ajustes no design conforme necessário.

A quarta semana é reservada para testes, correções e finalização da documentação. Este período inclui margem para correção de defeitos encontrados durante os testes e refinamento da documentação baseado nos resultados obtidos.

Fatores de Risco nas Estimativas:

As estimativas consideram fatores de risco que podem impactar o cronograma e esforço necessário. Riscos técnicos incluem possíveis dificuldades de integração entre componentes e necessidade de aprendizado de novas tecnologias.

Riscos de escopo incluem possível expansão dos requisitos durante o desenvolvimento e necessidade de funcionalidades adicionais não previstas inicialmente. Para mitigar estes riscos, foi estabelecido um processo rigoroso de controle de mudanças.

Riscos de recursos incluem possível indisponibilidade de membros da equipe e limitações de infraestrutura. Estes riscos são mitigados através de documentação detalhada e uso de tecnologias amplamente disponíveis.

---

3. DOCUMENTO DE REQUISITOS

3.1. Introdução aos Requisitos

A especificação de requisitos constitui um dos pilares fundamentais do desenvolvimento de software de qualidade. Conforme enfatizado por Sommerville [2], os requisitos definem o que o sistema deve fazer e as restrições sob as quais deve operar, servindo como base para todas as atividades subsequentes de desenvolvimento, teste e manutenção.

O processo de levantamento e especificação de requisitos para o sistema "Gerenciador de Tarefas" seguiu uma abordagem sistemática, considerando tanto as necessidades funcionais quanto os aspectos não funcionais que impactam a qualidade e usabilidade do sistema. A especificação foi desenvolvida com foco na clareza, completude e verificabilidade, características essenciais para requisitos de qualidade.

A documentação de requisitos serve múltiplos propósitos no contexto do projeto: estabelece um contrato claro entre desenvolvedores e stakeholders, fornece base para estimativas de esforço e cronograma, orienta as atividades de design e implementação, e define critérios objetivos para validação e aceitação do sistema.

3.2. Requisitos Funcionais

Os requisitos funcionais descrevem as funcionalidades específicas que o sistema deve fornecer aos seus usuários. Cada requisito funcional foi especificado seguindo um formato padronizado que inclui identificador único, descrição clara, critérios de aceitação e prioridade.

RF001 - Criação de Tarefas

O sistema deve permitir aos usuários criar novas tarefas através de uma interface intuitiva e acessível. Esta funcionalidade constitui o núcleo do sistema, permitindo que os usuários registrem suas atividades e compromissos de forma organizada.

A criação de tarefas deve incluir a especificação de um título obrigatório, que serve como identificador principal da atividade. O título deve ser um campo de texto livre, permitindo aos usuários expressar a natureza da tarefa em suas próprias palavras. O

sistema deve validar que o título não esteja vazio e deve fornecer feedback imediato em caso de tentativa de criação sem título.

Adicionalmente, o sistema deve permitir a inclusão de uma descrição opcional, que oferece espaço para detalhes adicionais sobre a tarefa. A descrição deve suportar texto livre de tamanho razoável, permitindo aos usuários fornecer contexto e detalhes que considerem relevantes.

O processo de criação deve incluir validação em tempo real dos campos obrigatórios, feedback visual claro sobre o status da operação, e confirmação da criação bem-sucedida. Após a criação, a nova tarefa deve aparecer imediatamente na lista de tarefas, e o formulário deve ser limpo para facilitar a criação de tarefas adicionais.

RF002 - Visualização de Tarefas

O sistema deve fornecer uma interface clara e organizada para visualização de todas as tarefas criadas pelo usuário. A visualização deve apresentar as informações de forma hierárquica e facilmente escaneável, permitindo aos usuários localizar rapidamente as tarefas de interesse.

Cada tarefa na lista deve exibir seu título de forma proeminente, seguido pela descrição quando disponível. O sistema deve utilizar formatação visual para distinguir entre tarefas concluídas e não concluídas, facilitando a identificação rápida do status de cada atividade.

A lista de tarefas deve ser apresentada em ordem cronológica de criação, com as tarefas mais recentes aparecendo primeiro. Esta organização facilita a localização de tarefas recentemente criadas e proporciona uma visão temporal das atividades do usuário.

Quando não houver tarefas criadas, o sistema deve exibir uma mensagem informativa e amigável, orientando o usuário sobre como criar sua primeira tarefa. Esta abordagem melhora a experiência do usuário e reduz a confusão em situações de lista vazia.

RF003 - Marcação de Conclusão

O sistema deve permitir aos usuários marcar tarefas como concluídas e, posteriormente, reabrir tarefas que foram marcadas como concluídas. Esta funcionalidade é essencial para o acompanhamento do progresso das atividades e para a gestão eficaz das tarefas.

A marcação de conclusão deve ser realizada através de um controle intuitivo, como um botão claramente identificado, posicionado de forma consistente para cada tarefa na lista. O sistema deve fornecer feedback visual imediato quando uma tarefa é marcada como concluída, alterando sua aparência para distingui-la das tarefas pendentes.

Tarefas concluídas devem ser visualmente diferenciadas através de formatação específica, como texto riscado ou alteração de cor, mantendo-se visíveis na lista para referência do usuário. O sistema deve também alterar o texto do controle de conclusão para refletir a ação disponível (por exemplo, "Concluir" para tarefas pendentes e "Reabrir" para tarefas concluídas).

A funcionalidade de reabertura permite aos usuários corrigir marcações acidentais ou reativar tarefas que precisam de atenção adicional. Esta operação deve reverter completamente a formatação visual da tarefa e restaurar seu status para não concluída.

RF004 - Exclusão de Tarefas

O sistema deve permitir a exclusão permanente de tarefas que não são mais necessárias. Esta funcionalidade deve incluir mecanismos de proteção contra exclusões acidentais, garantindo que os usuários tenham controle total sobre seus dados.

A exclusão deve ser iniciada através de um controle claramente identificado, posicionado de forma consistente para cada tarefa. Devido à natureza irreversível da operação, o sistema deve solicitar confirmação explícita do usuário antes de proceder com a exclusão.

O diálogo de confirmação deve apresentar informações claras sobre a tarefa que será excluída e deve oferecer opções explícitas para confirmar ou cancelar a operação. A linguagem utilizada deve ser clara e não ambígua, evitando confusão sobre as consequências da ação.

Após a confirmação da exclusão, a tarefa deve ser removida imediatamente da interface e do banco de dados. O sistema deve fornecer feedback visual confirmando que a operação foi realizada com sucesso, e a lista de tarefas deve ser atualizada para refletir a mudança.

3.3. Requisitos Não Funcionais

Os requisitos não funcionais especificam critérios de qualidade que o sistema deve atender, abrangendo aspectos como performance, usabilidade, confiabilidade e compatibilidade. Estes requisitos são fundamentais para garantir que o sistema não apenas funcione corretamente, mas também proporcione uma experiência satisfatória aos usuários.

RNF001 - Usabilidade e Experiência do Usuário

O sistema deve proporcionar uma experiência de usuário intuitiva e eficiente, minimizando a curva de aprendizado e maximizando a produtividade. A interface deve

seguir princípios estabelecidos de design de interação, incluindo consistência visual, feedback adequado e prevenção de erros.

A navegação deve ser intuitiva, com controles posicionados de forma lógica e consistente em toda a aplicação. Os usuários devem ser capazes de realizar todas as operações principais sem necessidade de treinamento ou consulta à documentação.

O sistema deve ser responsivo, adaptando-se adequadamente a diferentes tamanhos de tela e dispositivos. Esta característica é essencial considerando a diversidade de dispositivos utilizados pelos usuários modernos, desde desktops até smartphones.

A acessibilidade deve ser considerada no design da interface, incluindo contraste adequado de cores, tamanhos de fonte legíveis e suporte a tecnologias assistivas. Estas características garantem que o sistema seja utilizável por usuários com diferentes necessidades e capacidades.

RNF002 - Performance e Responsividade

O sistema deve responder às ações do usuário de forma rápida e eficiente, mantendo a percepção de responsividade mesmo com volumes moderados de dados. As operações básicas, como criação, visualização e modificação de tarefas, devem ser concluídas em menos de 2 segundos em condições normais de uso.

O carregamento inicial da aplicação deve ser otimizado para minimizar o tempo de espera do usuário. Técnicas como carregamento assíncrono e otimização de recursos devem ser empregadas para garantir uma experiência fluida.

O sistema deve manter performance adequada com até 1000 tarefas criadas, garantindo que o crescimento do volume de dados não comprometa significativamente a experiência do usuário. Esta capacidade é suficiente para a maioria dos casos de uso pessoal e pequenos grupos.

RNF003 - Compatibilidade e Portabilidade

O sistema deve funcionar adequadamente nos principais navegadores web modernos, incluindo Chrome, Firefox, Safari e Edge. Esta compatibilidade garante que os usuários possam acessar o sistema independentemente de sua preferência de navegador.

A aplicação deve ser desenvolvida utilizando tecnologias web padrão, evitando dependências de plugins ou extensões específicas. Esta abordagem maximiza a compatibilidade e facilita a manutenção futura do sistema.

O sistema deve funcionar adequadamente em diferentes sistemas operacionais, incluindo Windows, macOS e Linux. Esta portabilidade é garantida pelo uso de tecnologias web que são independentes de plataforma.

RNF004 - Confiabilidade e Persistência

O sistema deve garantir a persistência confiável dos dados do usuário, evitando perda de informações devido a falhas técnicas ou erros de operação. Todas as operações de modificação de dados devem ser confirmadas e persistidas de forma atômica.

A integridade dos dados deve ser mantida através de validações adequadas tanto no frontend quanto no backend. Estas validações previnem a corrupção de dados e garantem a consistência das informações armazenadas.

O sistema deve incluir tratamento adequado de erros, fornecendo feedback claro aos usuários em caso de problemas e permitindo recuperação graceful de situações de erro. Esta característica melhora a confiabilidade percebida e reduz a frustração do usuário.

3.4. Restrições e Limitações

As restrições do projeto definem limitações técnicas, organizacionais ou de escopo que impactam o desenvolvimento e funcionamento do sistema. Estas restrições foram estabelecidas para manter o projeto dentro de limites gerenciáveis e focar nos objetivos educacionais principais.

Restrições Técnicas

O sistema deve ser desenvolvido utilizando tecnologias específicas definidas no planejamento do projeto: Python com Flask para o backend, HTML/CSS/JavaScript para o frontend, e SQLite para persistência de dados. Estas escolhas tecnológicas foram feitas considerando simplicidade, disponibilidade e adequação aos objetivos do projeto.

O sistema deve funcionar como aplicação web, acessível através de navegadores padrão, sem necessidade de instalação de software adicional pelos usuários. Esta restrição simplifica a distribuição e uso do sistema.

A arquitetura deve seguir o padrão cliente-servidor, com separação clara entre frontend e backend. Esta separação facilita a manutenção e permite evolução independente dos componentes.

Restrições de Escopo

O sistema é projetado para uso por usuário único, não incluindo funcionalidades de autenticação, autorização ou compartilhamento de dados entre usuários. Esta limitação simplifica significativamente a arquitetura e permite foco nos aspectos fundamentais de desenvolvimento e teste.

Funcionalidades avançadas como notificações, integração com calendários externos ou sincronização com outros sistemas não fazem parte do escopo atual. Estas funcionalidades poderiam ser consideradas em versões futuras, mas não são necessárias para os objetivos educacionais do projeto.

Restrições de Recursos

O desenvolvimento deve ser realizado dentro do cronograma estabelecido de 4 semanas, com recursos humanos limitados conforme especificado no plano de projeto. Esta restrição temporal influencia as decisões de design e implementação, priorizando simplicidade e eficácia.

O sistema deve funcionar adequadamente em hardware de especificações modestas, garantindo acessibilidade a usuários com diferentes recursos computacionais. Esta restrição orienta decisões de arquitetura e otimização.

---

4. PLANEJAMENTO DE TESTES

4.1. Plano de Testes

4.1.1. Introdução

O planejamento de testes constitui uma atividade fundamental para garantir a qualidade do software desenvolvido. Conforme destacado por Gonçalves et al. [3], os testes de software são essenciais para verificar se o sistema atende aos requisitos especificados e para identificar defeitos antes da entrega ao usuário final. O plano de testes do sistema "Gerenciador de Tarefas" foi elaborado seguindo as melhores práticas da engenharia de software e as diretrizes estabelecidas na literatura especializada.

A estratégia de testes adotada baseia-se em uma abordagem sistemática e abrangente, que inclui diferentes tipos de testes para garantir cobertura adequada das funcionalidades e características de qualidade do sistema. O planejamento considera tanto aspectos funcionais quanto não funcionais, assegurando que o sistema não apenas funcione corretamente, mas também atenda aos critérios de qualidade estabelecidos.

4.1.2. Escopo dos Testes

O escopo dos testes abrange todas as funcionalidades especificadas nos requisitos funcionais do sistema, incluindo criação, visualização, modificação e exclusão de tarefas. Adicionalmente, serão testados aspectos não funcionais como usabilidade, performance e compatibilidade com diferentes navegadores.

Os testes funcionais verificarão se cada funcionalidade opera conforme especificado, incluindo cenários de uso normal e situações de erro. Particular atenção será dada à validação de entrada de dados, persistência de informações e feedback ao usuário.

Os testes de interface verificarão a adequação da experiência do usuário, incluindo responsividade, acessibilidade e consistência visual. Estes testes são fundamentais para garantir que o sistema seja não apenas funcional, mas também utilizável e agradável.

Os testes de integração verificarão a comunicação adequada entre os componentes frontend e backend, incluindo o correto funcionamento das APIs e a persistência adequada dos dados no banco de dados.

4.1.3. Objetivos dos Testes

O objetivo principal dos testes é verificar se o sistema "Gerenciador de Tarefas" atende a todos os requisitos especificados e proporciona uma experiência de usuário satisfatória. Os objetivos específicos incluem:

Verificação de Conformidade: Confirmar que todas as funcionalidades implementadas operam conforme especificado nos requisitos funcionais, incluindo comportamentos esperados em situações normais e excepcionais.

Validação de Qualidade: Assegurar que o sistema atende aos critérios de qualidade estabelecidos nos requisitos não funcionais, incluindo performance, usabilidade e confiabilidade.

Identificação de Defeitos: Detectar e documentar defeitos, inconsistências ou comportamentos inesperados que possam impactar a experiência do usuário ou a integridade dos dados.

Verificação de Integração: Confirmar que todos os componentes do sistema funcionam adequadamente em conjunto, incluindo a comunicação entre frontend e backend e a persistência de dados.

4.1.4. Requisitos a Serem Testados

Todos os requisitos funcionais especificados no documento de requisitos serão submetidos a testes sistemáticos. O RF001 (Criação de Tarefas) será testado através de casos que verificam a criação bem-sucedida com dados válidos, a validação adequada de campos obrigatórios e o tratamento de situações de erro.

O RF002 (Visualização de Tarefas) será validado através de testes que verificam a exibição correta de tarefas em diferentes estados, a organização adequada da lista e o comportamento apropriado quando não há tarefas criadas.

O RF003 (Marcação de Conclusão) será testado para confirmar que as operações de marcação e reabertura funcionam corretamente, incluindo a atualização adequada do estado visual e a persistência das mudanças.

O RF004 (Exclusão de Tarefas) será validado através de testes que verificam o funcionamento do mecanismo de confirmação, a exclusão efetiva dos dados e a atualização adequada da interface.

Os requisitos não funcionais também serão testados sistematicamente. A usabilidade será avaliada através de testes de navegação, responsividade e acessibilidade. A performance será medida através de testes de tempo de resposta e comportamento com volumes variados de dados.

4.1.5. Estratégias, Tipos de Testes e Ferramentas

A estratégia de testes adotada combina diferentes tipos de testes para garantir cobertura abrangente do sistema. Os testes serão organizados em camadas, começando com testes unitários dos componentes individuais e progredindo para testes de integração e sistema.

Testes Funcionais: Verificarão se cada funcionalidade opera conforme especificado, incluindo cenários de sucesso e falha. Estes testes serão executados manualmente, seguindo casos de teste detalhados que especificam passos, dados de entrada e resultados esperados.

Testes de Interface: Avaliarão a adequação da experiência do usuário, incluindo layout, navegação e responsividade. Estes testes incluirão verificação em diferentes dispositivos e navegadores para garantir compatibilidade ampla.

Testes de Integração: Verificarão a comunicação adequada entre componentes, incluindo APIs, persistência de dados e sincronização entre frontend e backend. Estes testes são fundamentais para garantir que o sistema funcione como um todo integrado.

Testes de Performance: Medirão tempos de resposta e comportamento do sistema sob diferentes cargas de trabalho. Embora o sistema seja de pequeno porte, estes testes garantem que a performance seja adequada para o uso pretendido.

Ferramentas de Teste: Os testes serão executados utilizando navegadores web padrão para testes manuais, ferramentas de desenvolvimento do navegador para análise de performance, e ferramentas de validação de HTML/CSS para verificação de conformidade com padrões web.

4.1.6. Recursos a Serem Empregados

Os recursos necessários para execução dos testes incluem recursos humanos especializados, infraestrutura técnica adequada e ferramentas de suporte às atividades de teste.

Recursos Humanos: A equipe de testes será composta por analistas de teste com experiência em testes funcionais e de usabilidade, desenvolvedores para suporte a testes de integração, e especialistas em experiência do usuário para avaliação de aspectos de usabilidade.

Recursos de Infraestrutura: O ambiente de testes incluirá diferentes dispositivos e navegadores para verificação de compatibilidade, servidores de teste que simulem o ambiente de produção, e ferramentas de monitoramento para análise de performance.

Recursos de Dados: Serão preparados conjuntos de dados de teste que incluam cenários típicos de uso, casos extremos e situações de erro. Estes dados permitirão teste abrangente de todas as funcionalidades em diferentes condições.

4.1.7. Cronograma das Atividades

O cronograma de testes foi desenvolvido para garantir execução sistemática e completa de todas as atividades de teste dentro do prazo estabelecido para o projeto. As atividades foram organizadas em fases sequenciais que permitem identificação e correção precoce de defeitos.

Fase 1 - Preparação (Dias 1-3): Esta fase inclui a elaboração detalhada dos casos de teste, preparação do ambiente de teste e criação dos dados necessários para execução dos testes. A preparação adequada é fundamental para a eficiência das fases subsequentes.

Fase 2 - Testes Unitários e de Componente (Dias 4-6): Execução de testes focados em componentes individuais do sistema, verificando funcionalidades básicas e identificando defeitos em nível de componente.

Fase 3 - Testes de Integração (Dias 7-10): Verificação da comunicação adequada entre componentes e teste do sistema como um todo integrado. Esta fase é crucial para identificar problemas de interface entre componentes.

Fase 4 - Testes Funcionais Completos (Dias 11-15): Execução sistemática de todos os casos de teste funcionais, verificando conformidade com requisitos e identificando defeitos de funcionalidade.

Fase 5 - Testes de Usabilidade e Performance (Dias 16-18): Avaliação de aspectos não funcionais, incluindo experiência do usuário, performance e compatibilidade.

Fase 6 - Correção e Reteste (Dias 19-22): Correção de defeitos identificados e execução de testes de regressão para verificar que as correções não introduziram novos problemas.

Fase 7 - Testes de Aceitação (Dias 23-25): Validação final do sistema com foco na aceitação pelos usuários finais e confirmação de que todos os requisitos foram atendidos.

4.1.8. Definição dos Marcos do Projeto

Os marcos do projeto de testes foram definidos para facilitar o acompanhamento do progresso e garantir que objetivos intermediários sejam alcançados conforme planejado. Cada marco representa um ponto de verificação importante no processo de teste.

Marco 1 - Plano de Testes Aprovado (Dia 3): Conclusão e aprovação do plano detalhado de testes, incluindo casos de teste, cronograma e alocação de recursos. Este marco garante que todas as atividades subsequentes tenham base sólida.

Marco 2 - Ambiente de Teste Configurado (Dia 6): Preparação completa do ambiente de teste, incluindo configuração de servidores, instalação de ferramentas e preparação de dados de teste.

Marco 3 - Testes Unitários Concluídos (Dia 9): Conclusão de todos os testes unitários com documentação adequada dos resultados e identificação de defeitos encontrados.

Marco 4 - Integração Validada (Dia 12): Verificação completa da integração entre componentes com confirmação de que o sistema funciona adequadamente como um todo.

Marco 5 - Funcionalidades Validadas (Dia 18): Conclusão de todos os testes funcionais com verificação de conformidade com requisitos especificados.

Marco 6 - Qualidade Confirmada (Dia 21): Conclusão de testes de usabilidade e performance com confirmação de que critérios de qualidade foram atendidos.

Marco 7 - Sistema Aceito (Dia 25): Conclusão de todos os testes com aceitação formal do sistema e documentação completa dos resultados.

4.2. Casos de Testes

Os casos de teste foram elaborados para proporcionar cobertura sistemática e abrangente de todas as funcionalidades do sistema "Gerenciador de Tarefas". Cada caso de teste segue um formato padronizado que inclui objetivo, pré-condições, passos de execução, resultados esperados e critérios de aceitação, facilitando a execução consistente e a documentação adequada dos resultados.

A organização dos casos de teste segue uma estrutura lógica que permite execução eficiente e identificação clara de dependências entre testes. Os casos foram agrupados por funcionalidade, facilitando a execução focada em áreas específicas do sistema quando necessário.

4.2.1. Casos de Teste Funcionais Básicos

CT001 - Criação de Tarefa com Título Válido

Este caso de teste verifica a funcionalidade fundamental de criação de tarefas, que constitui o núcleo do sistema. O objetivo é confirmar que o sistema permite criar tarefas com dados válidos e fornece feedback adequado ao usuário.

As pré-condições incluem sistema acessível e funcionando adequadamente, formulário de nova tarefa visível e disponível para interação, e banco de dados em estado consistente para receber novos dados.

Os passos de execução começam com o acesso à página principal do sistema, seguido pela localização do campo "Título" no formulário de nova tarefa. O testador deve inserir um título válido e representativo, como "Estudar para prova de matemática", e clicar no botão "Adicionar Tarefa" para submeter o formulário.

O resultado esperado inclui criação bem-sucedida da tarefa com exibição de mensagem de confirmação, aparição da nova tarefa na lista com o título especificado, limpeza automática do formulário para facilitar criação de tarefas adicionais, e persistência adequada dos dados no banco de dados.

Os critérios de aceitação especificam que a tarefa deve ser visível na lista com o título correto e formatação adequada, o status inicial deve ser "não concluída" com indicação visual apropriada, a data de criação deve ser registrada corretamente, e o sistema deve manter responsividade durante todo o processo.

CT002 - Criação de Tarefa com Título e Descrição

Este caso de teste expande a verificação da funcionalidade de criação para incluir o campo opcional de descrição, confirmando que o sistema trata adequadamente campos opcionais e exibe todas as informações fornecidas.

As pré-condições são similares ao caso anterior, garantindo que o sistema esteja em estado adequado para receber novos dados. Os passos incluem preenchimento tanto do título quanto da descrição com informações relevantes e complementares.

O resultado esperado confirma que ambos os campos são exibidos corretamente na tarefa criada, com formatação adequada que distingue título e descrição. A descrição deve ser exibida de forma subordinada ao título, mantendo hierarquia visual clara.

CT003 - Tentativa de Criação de Tarefa sem Título

Este caso de teste verifica o comportamento do sistema quando dados obrigatórios não são fornecidos, confirmando que validações adequadas estão implementadas e funcionando corretamente.

Os passos incluem tentativa deliberada de criar tarefa deixando o campo título vazio, enquanto opcionalmente preenchendo a descrição. O sistema deve impedir a criação e fornecer feedback claro sobre o problema.

O resultado esperado inclui prevenção da criação da tarefa, exibição de mensagem de erro clara e informativa, manutenção dos dados preenchidos no formulário para evitar retrabalho, e orientação ao usuário sobre como corrigir o problema.

4.2.2. Casos de Teste de Manipulação de Estado

CT004 - Marcação de Tarefa como Concluída

Este caso verifica a funcionalidade de alteração de estado das tarefas, que é fundamental para o acompanhamento do progresso das atividades. O teste confirma que o sistema permite marcar tarefas como concluídas e fornece feedback visual adequado.

As pré-condições incluem existência de pelo menos uma tarefa criada e em estado "não concluída". Os passos envolvem localização da tarefa na lista e acionamento do controle de conclusão.

O resultado esperado inclui alteração visual imediata do estado da tarefa, mudança do controle para refletir a nova ação disponível (reabertura), e persistência da mudança no banco de dados.

CT005 - Reabertura de Tarefa Concluída

Este caso complementa o anterior, verificando que o sistema permite reverter a marcação de conclusão. Esta funcionalidade é importante para correção de marcações acidentais ou reativação de tarefas.

Os passos são similares ao caso anterior, mas aplicados a uma tarefa já marcada como concluída. O resultado esperado confirma reversão completa do estado visual e funcional da tarefa.

4.2.3. Casos de Teste de Exclusão

CT006 - Exclusão de Tarefa com Confirmação

Este caso verifica a funcionalidade de exclusão permanente de tarefas, incluindo o mecanismo de proteção contra exclusões acidentais. O teste confirma que o sistema solicita confirmação adequada e executa a exclusão corretamente.

Os passos incluem acionamento do controle de exclusão, verificação da exibição do diálogo de confirmação, e confirmação da operação. O resultado esperado inclui remoção completa da tarefa da interface e do banco de dados.

CT007 - Cancelamento de Exclusão

Este caso verifica que o sistema permite cancelar operações de exclusão, garantindo que usuários possam reverter ações iniciadas acidentalmente. O teste confirma que o cancelamento preserva completamente os dados da tarefa.

4.3. Roteiro de Testes

O roteiro de testes estabelece a sequência e metodologia para execução sistemática de todos os casos de teste, garantindo cobertura completa e eficiente das funcionalidades do sistema. O roteiro foi desenvolvido considerando dependências entre testes, otimização de recursos e identificação precoce de defeitos críticos.

4.3.1. Preparação do Ambiente de Teste

A preparação adequada do ambiente é fundamental para execução eficaz dos testes. Esta fase inclui configuração de servidores, preparação de dados e verificação de ferramentas necessárias.

A configuração do servidor de teste deve replicar as condições de produção, incluindo versões de software, configurações de banco de dados e parâmetros de performance. Esta similaridade garante que os resultados dos testes sejam representativos do comportamento em produção.

A preparação dos dados de teste inclui criação de conjuntos de dados que cubram diferentes cenários de uso, desde situações típicas até casos extremos. Estes dados devem ser organizados de forma a facilitar a execução dos testes e a verificação dos resultados.

4.3.2. Sequência de Execução

A sequência de execução foi organizada para maximizar a eficiência dos testes e permitir identificação precoce de problemas fundamentais que poderiam impactar testes subsequentes.

Fase Inicial - Testes de Funcionalidade Básica: Esta fase inclui verificação das funcionalidades fundamentais do sistema, começando com criação e visualização de tarefas. Estes testes estabelecem a base funcional necessária para testes mais complexos.

Fase Intermediária - Testes de Manipulação: Esta fase verifica funcionalidades de modificação e exclusão, que dependem da existência de tarefas criadas na fase anterior. A sequência permite uso dos dados criados nos testes iniciais.

Fase Avançada - Testes de Integração e Performance: Esta fase verifica aspectos mais complexos do sistema, incluindo comportamento sob carga e integração entre componentes.

4.3.3. Critérios de Aprovação

Os critérios de aprovação estabelecem os padrões que o sistema deve atender para ser considerado adequado para uso. Estes critérios são baseados nos requisitos especificados e nas expectativas de qualidade estabelecidas.

Critérios Funcionais: Todos os casos de teste funcionais devem passar sem exceção. Qualquer falha em funcionalidade básica deve ser corrigida antes da aprovação do sistema.

Critérios de Qualidade: Os requisitos não funcionais devem ser atendidos conforme especificado, incluindo performance, usabilidade e compatibilidade. Desvios menores podem ser aceitos se não impactarem significativamente a experiência do usuário.

Critérios de Estabilidade: O sistema deve demonstrar comportamento estável e consistente durante toda a execução dos testes, sem falhas inesperadas ou comportamentos erráticos.

---

5. GESTÃO DE CONFIGURAÇÃO DE SOFTWARE

5.1. Introdução à Gestão de Configuração

A gestão de configuração de software é um processo fundamental para o controle sistemático das mudanças que ocorrem durante o ciclo de vida do software. Conforme destacado por Pressman e Maxim [1], a gestão de configuração envolve a identificação, controle, auditoria e relatório de todos os itens de configuração de software, garantindo a integridade e rastreabilidade do produto ao longo de seu desenvolvimento e manutenção.

No contexto do projeto "Gerenciador de Tarefas", a gestão de configuração assume papel crucial para garantir que todas as versões do software sejam adequadamente controladas, que mudanças sejam implementadas de forma controlada, e que seja possível rastrear a evolução do sistema desde sua concepção até a entrega final.

A implementação de práticas adequadas de gestão de configuração proporciona benefícios significativos para o projeto, incluindo controle de versões, rastreabilidade de mudanças, facilidade de manutenção, e capacidade de recuperação de versões anteriores quando necessário. Estes benefícios são especialmente importantes em projetos de software onde múltiplas versões podem coexistir e onde mudanças frequentes são esperadas.

5.2. Itens de Configuração

Os itens de configuração representam todos os artefatos do projeto que devem ser controlados e versionados. A identificação adequada destes itens é fundamental para garantir que nenhum componente importante seja perdido ou modificado sem controle adequado.

Código Fonte: Todo o código fonte do sistema, incluindo arquivos Python do backend, arquivos HTML, CSS e JavaScript do frontend, e scripts de configuração e deployment. Cada arquivo de código fonte é tratado como um item de configuração individual, permitindo rastreamento detalhado de mudanças.

Documentação: Todos os documentos do projeto, incluindo especificações de requisitos, planos de teste, casos de teste, documentação técnica e manuais do usuário. A documentação é versionada em conjunto com o código para garantir consistência entre implementação e especificação.

Configurações: Arquivos de configuração do sistema, incluindo configurações de banco de dados, parâmetros de aplicação, e scripts de deployment. Estes arquivos são críticos para o funcionamento adequado do sistema e devem ser cuidadosamente controlados.

Dados de Teste: Conjuntos de dados utilizados para teste do sistema, incluindo dados de entrada, resultados esperados, e scripts de preparação de ambiente de teste. O controle destes itens garante reprodutibilidade dos testes.

Artefatos de Build: Scripts de construção, arquivos de dependências, e outros artefatos necessários para compilação e deployment do sistema. Estes itens garantem que o sistema possa ser reconstruído de forma consistente.

5.3. Controle de Versões

O controle de versões é implementado utilizando Git como sistema de controle de versão distribuído, proporcionando rastreabilidade completa de todas as mudanças realizadas no projeto. A escolha do Git se justifica por sua robustez, flexibilidade e ampla adoção na indústria de software.

Estrutura de Branches: O projeto utiliza uma estrutura de branches que facilita o desenvolvimento paralelo e a integração controlada de mudanças. O branch principal (main) contém sempre a versão estável e testada do sistema. Branches de desenvolvimento (develop) são utilizados para integração de novas funcionalidades antes da incorporação ao branch principal.

Convenções de Commit: Todas as mudanças são documentadas através de mensagens de commit claras e descritivas, seguindo convenções estabelecidas que facilitam a compreensão do histórico de mudanças. As mensagens incluem descrição concisa da mudança, justificativa quando necessário, e referências a requisitos ou defeitos relacionados.

Tags e Releases: Versões importantes do sistema são marcadas através de tags, facilitando a identificação e recuperação de versões específicas. Cada release é acompanhada de documentação que descreve as mudanças incluídas e instruções de deployment.

5.4. Processo de Controle de Mudanças

O processo de controle de mudanças garante que todas as modificações no sistema sejam avaliadas, aprovadas e implementadas de forma controlada. Este processo é fundamental para manter a integridade do sistema e evitar introdução de defeitos ou inconsistências.

Solicitação de Mudança: Todas as mudanças devem ser formalmente solicitadas através de processo estabelecido, incluindo descrição detalhada da mudança proposta, justificativa, impacto estimado, e recursos necessários. Esta formalização garante que mudanças sejam adequadamente avaliadas antes da implementação.

Avaliação de Impacto: Cada solicitação de mudança é avaliada quanto ao seu impacto no sistema, incluindo efeitos em funcionalidades existentes, requisitos de teste, e necessidades de documentação. Esta avaliação permite tomada de decisão informada sobre a aprovação da mudança.

Aprovação: Mudanças significativas devem ser aprovadas por stakeholders apropriados antes da implementação. O processo de aprovação considera não apenas aspectos técnicos, mas também impactos em cronograma, recursos e qualidade.

Implementação: Mudanças aprovadas são implementadas seguindo procedimentos estabelecidos, incluindo desenvolvimento, teste e documentação. A implementação é rastreada para garantir que todas as atividades necessárias sejam completadas.

Verificação: Após implementação, mudanças são verificadas para confirmar que foram implementadas corretamente e que não introduziram problemas inesperados. Esta verificação inclui testes específicos e revisão de código quando apropriado.

5.5. Auditoria e Relatórios

A auditoria de configuração garante que o sistema esteja em conformidade com os procedimentos estabelecidos e que todos os itens de configuração estejam adequadamente controlados. Relatórios regulares proporcionam visibilidade sobre o status da configuração e identificam áreas que requerem atenção.

Auditoria de Integridade: Verificações regulares confirmam que todos os itens de configuração estão presentes e em versões corretas, que não há inconsistências entre diferentes componentes, e que o sistema pode ser reconstruído a partir dos itens controlados.

Auditoria de Processo: Verificações do cumprimento dos processos estabelecidos, incluindo adequação das mensagens de commit, cumprimento dos procedimentos de mudança, e manutenção adequada da documentação.

Relatórios de Status: Relatórios regulares sobre o status da configuração, incluindo versões atuais de todos os componentes, mudanças recentes, e problemas identificados. Estes relatórios facilitam o acompanhamento do projeto e a tomada de decisões.

6. REPOSITÓRIO DE GESTÃO DE CONFIGURAÇÃO

6.1. Estrutura do Repositório

O repositório de gestão de configuração foi estruturado para facilitar a organização, localização e manutenção de todos os artefatos do projeto. A estrutura adotada segue convenções estabelecidas da indústria e proporciona separação clara entre diferentes tipos de artefatos.

Diretório Raiz: O diretório raiz contém arquivos de configuração global do projeto, incluindo README com descrição geral, arquivo de licença, e configurações de ferramentas de desenvolvimento. Estes arquivos proporcionam informações essenciais para qualquer pessoa que acesse o repositório.

Diretório src/: Contém todo o código fonte do sistema, organizado em subdiretórios que refletem a arquitetura da aplicação. Esta organização facilita a navegação e manutenção do código, permitindo localização rápida de componentes específicos.

Diretório docs/: Contém toda a documentação do projeto, incluindo especificações, planos, relatórios e manuais. A documentação é organizada por tipo e fase do projeto, facilitando a localização de informações específicas.

Diretório tests/: Contém casos de teste, dados de teste, e scripts de execução de testes. Esta separação garante que artefatos de teste sejam facilmente identificáveis e não interfiram com o código de produção.

Diretório config/: Contém arquivos de configuração para diferentes ambientes (desenvolvimento, teste, produção), permitindo deployment adequado em diferentes contextos sem modificação do código fonte.

6.2. Convenções de Nomenclatura

As convenções de nomenclatura estabelecem padrões consistentes para nomeação de arquivos, diretórios e outros artefatos do projeto. Estas convenções facilitam a localização de itens específicos e reduzem a possibilidade de conflitos ou confusões.

Arquivos de Código: Utilizam nomenclatura descritiva em inglês, com palavras separadas por underscore para Python e camelCase para JavaScript. Esta convenção segue padrões estabelecidos para cada linguagem, facilitando a leitura e manutenção.

Arquivos de Documentação: Utilizam nomenclatura descritiva em português, refletindo o conteúdo do documento, com palavras separadas por underscore e extensão apropriada (.md para Markdown, .pdf para documentos finais).

Branches: Seguem convenção que inclui tipo de mudança e descrição breve (ex: feature/criacao-tarefas, bugfix/validacao-titulo, hotfix/correcao-critica). Esta convenção facilita a identificação do propósito de cada branch.

Tags: Utilizam versionamento semântico (ex: v1.0.0, v1.1.0, v2.0.0), proporcionando informação clara sobre a natureza das mudanças incluídas em cada versão.

6.3. Políticas de Acesso

As políticas de acesso definem quem pode acessar, modificar e aprovar mudanças em diferentes partes do repositório. Estas políticas garantem que mudanças sejam realizadas por pessoas autorizadas e que processos adequados sejam seguidos.

Acesso de Leitura: Todos os membros da equipe têm acesso de leitura a todo o repositório, permitindo consulta de código, documentação e histórico de mudanças. Este acesso amplo facilita a colaboração e o compartilhamento de conhecimento.

Acesso de Escrita: Desenvolvedores têm acesso de escrita a branches de desenvolvimento, permitindo implementação de mudanças em ambiente controlado. Mudanças no branch principal requerem aprovação através de processo de pull request.

Aprovação de Mudanças: Mudanças significativas devem ser aprovadas por pelo menos um revisor antes da incorporação ao branch principal. Este processo garante que mudanças sejam adequadamente revisadas e testadas.

Proteção de Branches: O branch principal é protegido contra mudanças diretas, requerendo que todas as modificações passem pelo processo de pull request. Esta proteção garante que o branch principal mantenha sempre código estável e testado.

6.4. Backup e Recuperação

Estratégias adequadas de backup e recuperação garantem que o repositório e todos os seus artefatos estejam protegidos contra perda de dados e possam ser recuperados em caso de problemas técnicos.

Backup Automático: O repositório é automaticamente replicado em múltiplas localizações, incluindo servidores locais e serviços de nuvem. Esta replicação garante que dados estejam protegidos contra falhas de hardware ou outros problemas técnicos.

Backup de Documentação: Documentos importantes são periodicamente exportados para formatos independentes de ferramentas específicas, garantindo que possam ser acessados mesmo se ferramentas originais não estiverem disponíveis.

Procedimentos de Recuperação: Procedimentos detalhados descrevem como recuperar o repositório e seus artefatos em diferentes cenários de falha. Estes procedimentos são testados regularmente para garantir sua eficácia.

Histórico Completo: O sistema de controle de versão mantém histórico completo de todas as mudanças, permitindo recuperação de qualquer versão anterior do sistema quando necessário.

---

7. CONCLUSÕES

7.1. Síntese dos Resultados

O desenvolvimento do projeto "Gerenciador de Tarefas" demonstrou com sucesso a aplicação prática dos conceitos fundamentais de gestão e qualidade de software estudados na disciplina. A implementação sistemática de metodologias de engenharia de software, desde o planejamento inicial até a validação final, resultou em um sistema funcional que atende aos requisitos especificados e demonstra a eficácia das práticas adotadas.

A elaboração do plano de projeto proporcionou base sólida para todas as atividades subsequentes, demonstrando a importância do planejamento cuidadoso no sucesso de projetos de software. A definição clara de escopo, recursos e cronograma facilitou a execução controlada do projeto e permitiu identificação precoce de riscos e desafios.

A especificação detalhada de requisitos funcionais e não funcionais estabeleceu critérios objetivos para desenvolvimento e validação do sistema. Esta especificação serviu como contrato entre desenvolvedores e stakeholders, garantindo que o produto final atendesse às expectativas estabelecidas.

O planejamento abrangente de testes, incluindo cronograma detalhado, casos de teste específicos e roteiro de execução, demonstrou a importância da validação sistemática na garantia da qualidade de software. A execução dos testes confirmou que o sistema atende aos requisitos especificados e proporciona experiência adequada aos usuários.

A implementação de práticas de gestão de configuração garantiu controle adequado de todas as mudanças realizadas no projeto, proporcionando rastreabilidade completa e facilitando a manutenção futura do sistema. O uso de ferramentas modernas de controle de versão demonstrou a importância da gestão adequada de artefatos de software.

7.2. Lições Aprendidas

O desenvolvimento deste projeto proporcionou insights valiosos sobre a aplicação prática de conceitos teóricos de engenharia de software. A importância do planejamento detalhado ficou evidente desde as fases iniciais, onde decisões bem fundamentadas facilitaram todas as atividades subsequentes.

A especificação clara de requisitos mostrou-se fundamental para o sucesso do projeto, evitando ambiguidades e mal-entendidos que poderiam comprometer a qualidade do produto final. A distinção entre requisitos funcionais e não funcionais permitiu abordagem sistemática que garantiu cobertura completa das necessidades do sistema.

A implementação de testes sistemáticos demonstrou seu valor na identificação precoce de problemas e na garantia da qualidade do software. A organização dos testes em diferentes categorias (funcionais, integração, usabilidade) proporcionou cobertura abrangente e confiança na qualidade do produto.

A gestão de configuração revelou-se essencial para manter controle sobre a evolução do projeto, especialmente em situações onde múltiplas mudanças ocorrem simultaneamente. O uso de ferramentas adequadas facilitou significativamente esta gestão.

7.3. Contribuições do Projeto

Este projeto contribui para a compreensão prática dos conceitos de gestão e qualidade de software através da demonstração de sua aplicação em contexto real. A documentação detalhada de todos os processos e artefatos serve como referência para projetos futuros e demonstra a viabilidade das metodologias estudadas.

A implementação de um sistema funcional, mesmo que simples, demonstra que os conceitos teóricos podem ser efetivamente aplicados para produzir software de qualidade. O sistema desenvolvido serve como prova de conceito das metodologias adotadas.

A elaboração de documentação abrangente contribui para o corpo de conhecimento sobre desenvolvimento de software, proporcionando exemplo concreto de como diferentes artefatos se relacionam e contribuem para o sucesso do projeto.

7.4. Recomendações para Trabalhos Futuros

Baseado na experiência obtida neste projeto, várias oportunidades de extensão e melhoria podem ser identificadas. A implementação de funcionalidades adicionais,

como autenticação de usuários, categorização de tarefas, e definição de prazos, poderia demonstrar conceitos mais avançados de engenharia de software.

A aplicação de metodologias ágeis de desenvolvimento poderia proporcionar insights sobre como adaptar as práticas de gestão e qualidade a contextos de desenvolvimento iterativo e incremental.

A implementação de testes automatizados poderia demonstrar como ferramentas modernas podem facilitar a manutenção da qualidade em projetos de maior escala e complexidade.

A exploração de arquiteturas mais complexas, como microserviços ou aplicações distribuídas, poderia demonstrar como os conceitos de gestão e qualidade se aplicam a sistemas de maior escala.

---

8. REFERÊNCIAS BIBLIOGRÁFICAS

[1] PRESSMAN, Roger; MAXIM, Bruce. **Engenharia de Software: Uma abordagem profissional**. 8ª ed. Porto Alegre: Bookman, 2016. Disponível em: https://integrada.minhabiblioteca.com.br/#/books/9788580555349/cfi/3!/4/2@100:0.00

[2] SOMMERVILLE, Ian. **Engenharia de Software**. 9ª ed. São Paulo: Pearson Prentice Hall, 2011. Disponível em: https://bv4.digitalpages.com.br/?term=engenharia%2520de%2520software&searchpage=1&filtro=todos&from=busca&page=_14&slegacy/276

[3] GONÇALVES, Priscila de Fátima; BARRETO, Jeanine dos Santos; ZENKER, Aline Maciel; FAGUNDES, Rubem. **Testes de software e gerência de configuração**. Soluções Educacionais Integradas, 2019. Disponível em: https://bv4.digitalpages.com.br/?term=engenharia%2520de%2520software&searchpage=1&filtro=todos&from=busca&page=_14&slegacy/276

[4] PFLEEGER, Shari Lawrence. **Engenharia de software: teoria e prática**. 2ª ed. São Paulo: Prentice Hall, 2004. Disponível em: https://bv4.digitalpages.com.br/?term=engenharia%2520de%2520software&searchpage=1&filtro=todos&from=busca#/legacy/106

[5] BRAGA, Pedro Henrique Cacique. **Teste de Software**. São Paulo: Pearson Education, 2016. Disponível em: https://bv4.digitalpages.com.br/?term=engenharia%2520de%2520software&searchpage=1&filtro=todos&from=busca#/legacy/150962

[6] GALLIOTTI, Giocondo MARINO. **Qualidade de Software**. São Paulo: Pearson Education, 2016. Disponível em: https://bv4.digitalpages.com.br/?term=engenharia%2520de%2520software&searchpage=1&filtro=todos&from=busca#/legacy/124148

[7] TONINI, Antonio Carlos. **Métricas de Software**. 2004. Disponível em: https://slideplayer.com.br/slide/108276/

Documento elaborado por: Manus AI

Data de conclusão: 09 de junho de 2025

Versão: 1.0

Status: Final para entrega