Documentação de Projeto

para o sistema

SET (Software Estimation Tool)

Versão 0.1

Projeto de sistema elaborado pelo aluno **Inácio Moraes da Silva** e apresentado ao curso de **Engenharia de Software** da **PUC Minas** como parte do Trabalho de Conclusão de Curso (TCC) sob orientação de conteúdo e orientação acadêmica do professor **Cleiton Silva Tavares**.

1 de agosto 2025

Tabela de Conteúdo

Histórico de Revisões

Nome	Data	Razões para Mudança	Versão
Inacio Moraes	10/09/2025	Documentação inicial	0.1

1. Introdução

Este documento agrega: 1) a elaboração e revisão de modelos de domínio e 2) modelos de projeto para o sistema **SET** (Software Estimation Tool). A referência principal para a descrição geral do problema, domínio e requisitos do sistema é o Documento de Visão que descreve a visão de domínio do sistema.

O SET é uma ferramenta **CLI** (Command Line Interface) inteligente desenvolvida em **Go** que utiliza Inteligência Artificial para revolucionar o processo de estimativa de esforço em projetos de software. A escolha da linguagem Go proporciona alta performance, facilidade de distribuição multiplataforma através de binários únicos, e excelente suporte para integração com APIs REST.

1.1 Público-Alvo e Natureza CLI

Importante: O SET é uma ferramenta de linha de comando (CLI) projetada especificamente para profissionais com background técnico em desenvolvimento de software. O público-alvo inclui:

- Desenvolvedores que trabalham diariamente com terminal e ferramentas CLI
- Scrum Masters com formação técnica e experiência em desenvolvimento
- Product Owners com background como desenvolvedores

Todos os usuários do sistema possuem familiaridade com:

- Terminal/console (Linux, macOS, Windows PowerShell)
- Git e versionamento de código
- Ferramentas CLI comuns (npm, docker, kubectl, etc.)
- Conceitos de desenvolvimento de software

Esta escolha deliberada por CLI permite integração natural no workflow de desenvolvedores, automação via scripts, e uso em pipelines CI/CD, mantendo a ferramenta enxuta e eficiente.

2. Modelos de Usuário e Requisitos

2.1 Descrição de Atores

Scrum Master: Facilitador de metodologias ágeis com formação técnica em desenvolvimento de software, responsável por utilizar a ferramenta para obter estimativas mais precisas durante o planejamento de sprints e auxiliar na tomada de decisões sobre a capacidade da equipe. Confortável com terminal e ferramentas CLI.

Desenvolvedor: Membro da equipe de desenvolvimento que utilizará a ferramenta diariamente para validar estimativas próprias e do time, comparando com dados históricos e obtendo sugestões baseadas em IA. Usuário avançado de terminal e ferramentas de linha de comando.

Product Owner: Responsável pelo backlog do produto com experiência prévia como desenvolvedor, que usará as estimativas geradas para priorização de features e planejamento de releases. Mantém familiaridade com ferramentas técnicas e CLI para entender melhor a complexidade das tarefas.

2.2 Modelos de Usuários (Personas)

Esta seção apresenta os modelos de usuários desenvolvidos por meio da implementação de personas que representam os principais usuários da ferramenta CLI SET. As personas foram criadas baseando-se nas necessidades identificadas no Documento de Visão e nos padrões observados em equipes ágeis de desenvolvimento.

Perfil Técnico das Personas: Todas as personas possuem formação técnica e experiência com desenvolvimento de software, refletindo a natureza CLI da ferramenta. O SET é projetado para profissionais que trabalham diariamente com terminal, versionamento de código (Git), e outras ferramentas de linha de comando. A diferença entre as personas está nos seus papéis atuais dentro da equipe ágil, não no background técnico.

2.2.1 Persona 1: Scrum Master

Aspecto	Detalhes
Nome	Carlos Mendes
Idade e Perfil	35 anos, Scrum Master com 8 anos de experiência em metodologias ágeis
Descrição	Carlos é Scrum Master em uma startup de tecnologia com 15 desenvolvedores. Formado em Ciência da Computação, possui certificação PSM I e PSM II. Trabalha com 3 times de Scrum simultaneamente e é apaixonado por métricas e melhoria contínua. Valoriza transparência e busca constantemente formas de otimizar o processo de desenvolvimento.
Contexto Técnico	• Utiliza Jira, Azure DevOps e GitHub diariamente • Familiarizado com CLI tools (git, docker, kubectl)

	• Experiência com ferramentas de métricas (SonarQube, New Relic) • Confortável com terminal Linux/Mac
Dores Principais	• **Planning Poker Subjetivo**: "As estimativas do planning poker variam muito entre os desenvolvedores e nem sempre refletem a realidade" • **Falta de Dados Históricos**: "Não temos uma base sólida de dados para justificar nossas estimativas para o PO" • **Tempo Gasto em Reuniões**: "Gastamos 2-3 horas por sprint só em estimativas e nem sempre acertamos" • **Inconsistência Entre Times**: "Cada time estima de forma diferente, dificultando comparações"
Objetivos	• Reduzir tempo de reuniões de Planning em 50% • Aumentar precisão das estimativas para >80% • Ter dados concretos para apresentar ao management • Melhorar previsibilidade de entrega dos times • Identificar padrões de super/subestimação
Cenários de Uso	**Preparação de Sprint Planning:** • *set sync --custom-fields* • *set batch --file backlog.json --format csv --output sprint-15.csv* • *set batch --file backlog.json --format markdown --output planning-report.md* • *set batch --file backlog.json --workers 10 --progress* **Revisão de Sprint:** • *set inspect --list --custom --limit 50* • *set export --format csv --output team-metrics.csv* • *set export --format markdown --output sprint-summary.md*
Métricas de Sucesso	• Redução de 50% no tempo de planning • Acurácia > 80% nas estimativas

	• Velocity mais previsível • Menos re-estimativas durante o sprint

2.2.2 Persona 2: Desenvolvedor Sênior

Aspecto	Detalhes
Nome	Ana Rodrigues
Idade e Perfil	29 anos, Desenvolvedor Full Stack Sênior com 6 anos de experiência
Descrição	Ana é desenvolvedora sênior em uma consultoria de software que atende múltiplos clientes. Especialista em React, Node.js e Go. Lidera tecnicamente 2 desenvolvedores juniors e frequentemente é consultada para estimativas de tarefas complexas. Preocupa-se com a qualidade técnica e a precisão das entregas.
Contexto Técnico	• Expert em Git, GitHub Actions, Docker • Usuária avançada de terminal (Zsh, Vim) • Experiência com ferramentas CLI (make, npm, go build) • Trabalha com múltiplos repositórios simultaneamente
Dores Principais	• **Pressão por Estimativas**: "Sempre me pedem para estimar tarefas complexas e a responsabilidade é grande" • **Esquecimento de Detalhes**: "Às vezes esqueço de considerar testes, refactoring ou edge cases" • **Comparação Difícil**: "É difícil lembrar quanto tempo levou uma tarefa similar há 3 meses" • **Estimativas de Juniores**: "Os desenvolvedores juniors sempre subestimam por falta de experiência"

Objetivos	• Ter base de dados para fundamentar estimativas • Ajudar desenvolvedores juniors com referências • Melhorar a própria auto calibragem • Reduzir ansiedade nas estimativas • Documentar padrões de desenvolvimento
Cenários de Uso	**Estimativa Individual:** • *set estimate "Implementar auth JWT" --labels backend,security --show-similar* • *set estimate "Criar API REST" --labels backend --similar --output json* • *set estimate "Refatorar módulo" --description "Melhorar arquitetura" --labels refactor* **Análise de Dados:** • *set inspect --list --custom* • *set sync --custom-fields* • *set export --format json --output my-estimates.json*
Métricas de Sucesso	• Menor variação entre estimado vs real • Melhoria na mentoria de juniores • Redução de stress em estimativas

2.2.3 Persona 3: Product Owner

Aspecto	Detalhes
Nome	Roberto Santos
Idade e Perfil	42 anos, Product Owner com background como desenvolvedor (10 anos) e 4 anos em produtos digitais

Descrição	Roberto é PO de um produto SaaS B2B com 50k+ usuários. Começou a carreira como desenvolvedor full-stack, trabalhou 10 anos com código antes de migrar para o time de produtos. Usa sua experiência técnica para fazer estimativas mais realistas e se comunicar melhor com a equipe de engenharia. Mantém contato com código ocasionalmente para entender a complexidade técnica.
Contexto Técnico	• **Ex-desenvolvedor**: 10 anos de experiência com Python, JavaScript e Go • **Confortável com CLI**: Usa terminal diariamente (git, docker, npm) • **GitHub power user**: Revisa PRs ocasionalmente, gerencia issues • **Ferramentas de desenvolvimento**: Familiarizado com IDEs, CI/CD, ferramentas DevOps
Dores Principais	• **Pressão de Prazos**: "O CEO sempre quer saber quando a feature vai estar pronta" • **Falta de Visibilidade**: "Não sei se as estimativas são realistas até ser tarde demais" • **Comunicação com Stakeholders**: "É difícil explicar por que uma 'simples' feature leva 2 sprints" • **Priorização Sem Dados**: "Preciso escolher entre features sem saber o real esforço de cada uma"
Objetivos	• Ter previsões confiáveis para planejamento de releases • Dados para justificar priorizações • Comunicação clara com stakeholders • Melhor ROI nas decisões de produto • Reduzir riscos de atraso em releases

Cenários de Uso	**Planejamento de Release:** • *set batch --file epic-user-management.json --format markdown --output epic-estimate.md* • *set batch --file q1-features.csv --format csv --output roadmap.csv* • *set export --format jira --output jira-import.csv* **Acompanhamento:** • *set inspect --list* • *set export --format github --output github-import.csv* • *set export --format excel --output stakeholder-report.csv* • *set export --filter feature --format markdown --output features-report.md*
Métricas de Sucesso	• Releases entregues no prazo (+90%) • Melhor comunicação com stakeholders • Decisões de priorização data-driven • Redução de scope creep

2.2.4 Padrões de Uso Identificados

Padrão 1: Desenvolvedor Solo

- Usa *set estimate "<descrição>"* para tarefas individuais
- Foca em *--show-similar* para ver tarefas similares históricas
- Adiciona *--labels* para melhorar precisão da busca
- Consulta dados com set inspect *--list*
- Formato JSON disponível: *--output json*
- Pode adicionar contexto extra: *--description* e *--context*

Padrão 2: Scrum Master Facilitador

- Usa *set batch --file backlog.json* para preparar Planning
- Sincroniza dados com *set sync --custom-fields*
- Exporta dados com *set export --format markdown/csv*
- Processa em paralelo com *--workers N*
- Visualiza progresso com *--progress*
- Limita resultados de inspeção: *--limit N*

Padrão 3: Product Owner Estratégico

- Usa *set batch* para estimativas de épicos completos
- Exporta para Jira: *set export --format jira*
- Exporta para GitHub Projects: *set export --format github*
- Relatórios executivos: *set export --format markdown* ou *--format excel*
- Estatísticas agregadas via batch com múltiplos formatos
- Filtra por label: *set export --filter <label>*

2.3 Modelo de Casos de Uso e Histórias de Usuários

Esta seção apresenta o diagrama de casos de uso e as histórias de usuários que descrevem as principais interações dos atores com o sistema SET CLI. O diagrama ilustra as funcionalidades essenciais e suas relações, enquanto as histórias de usuários detalham os requisitos funcionais na perspectiva de cada tipo de usuário.

2.3.1 Diagrama de Casos de Uso

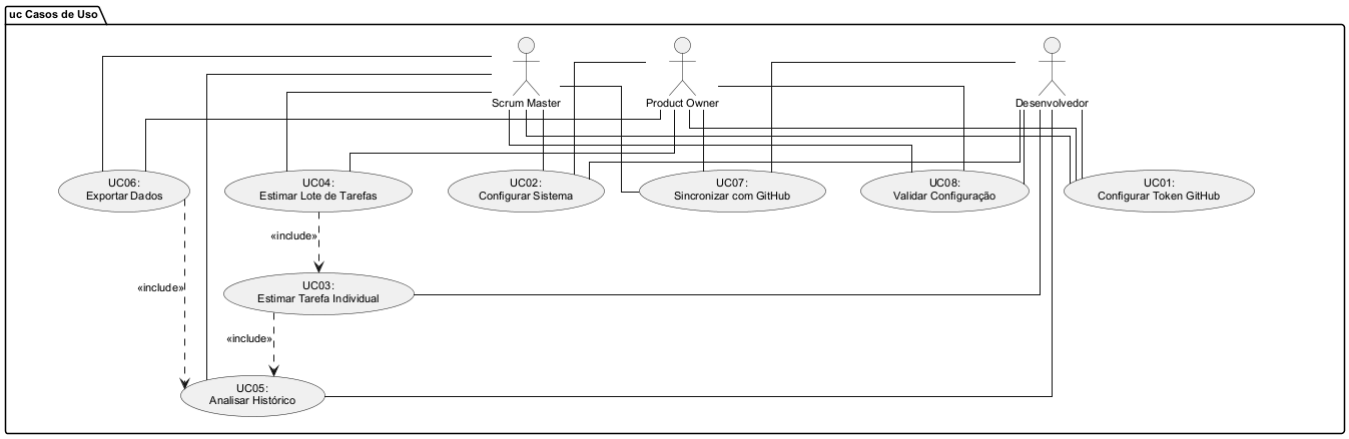


Figura 2.1 - Casos de Uso

O diagrama(Figura 2.1) acima apresenta os principais casos de uso do sistema SET CLI implementados. Para fins de organização, utiliza-se identificadores no formato UC#ID, onde UC refere-se a Use Case.

Casos de Uso:

Código	Caso de Uso	CLI
UC01	Configurar Token GitHub	*set configure --github-token*

UC02	Configurar Sistema	*set configure --initial*
UC03	Estimar Tarefa Individual	*set estimate "<descrição>"*
UC04	Estimar Lote de Tarefas	*set batch --file <arquivo>*
UC05	Analisar Histórico	*set inspect [--list]*
UC06	Exportar Dados	*set export --format <csv|json|markdown|jira|github|excel>*
UC07	Sincronizar com GitHub	*set sync [--custom-fields]*
UC08	Validar Configuração	*set configure --validate*

2.3.2 Histórias de Usuários

As histórias de usuário apresentadas a seguir estão organizadas por ator e relacionadas aos casos de uso identificados. Para organização, utiliza-se o formato **US#ID**, onde US refere-se a User Story.

Histórias do Desenvolvedor:

ID	Caso de Uso	História
US01	UC01, UC02	Como **desenvolvedor**, eu quero **configurar a CLI SET com meu token GitHub** para que o sistema possa acessar dados dos meus repositórios e gerar estimativas personalizadas.

US02	UC03	Como **desenvolvedor**, eu quero **estimar uma tarefa específica via CLI** para obter uma estimativa baseada em dados históricos e IA, economizando tempo em planning.
US03	UC03	Como **desenvolvedor**, eu quero **ver tarefas similares encontradas durante a estimativa** para validar se a sugestão da IA faz sentido com minha experiência.
US04	UC05	Como **desenvolvedor**, eu quero **analisar meu histórico pessoal de estimativas** para identificar padrões e melhorar minha precisão ao estimar.
US05	UC03	Como **desenvolvedor**, eu quero **receber um score de confiança com a estimativa** para entender a confiabilidade da sugestão da IA.
US06	UC03, UC05	Como **desenvolvedor**, eu quero **usar labels para categorizar tarefas** para melhorar a precisão das estimativas baseadas em similaridade.
US07	UC08	Como **desenvolvedor**, eu quero **validar minha configuração do sistema** para garantir que o token GitHub e repositório estão corretos antes de começar a usar a ferramenta.

Histórias do Scrum Master:

ID	Caso de Uso	História
US08	UC01, UC02	Como **Scrum Master**, eu quero **configurar a CLI SET com meu token GitHub e repositório padrão** para acessar dados históricos do time e automatizar o processo de estimativas.

US09	UC04	Como **Scrum Master**, eu quero **estimar um lote de tarefas do backlog** para preparar a reunião de Planning com dados concretos e acelerar as discussões.
US10	UC05, UC06	Como **Scrum Master**, eu quero **exportar dados históricos de estimativas** para análise externa e identificação de padrões do time.
US11	UC05	Como **Scrum Master**, eu quero **inspecionar estatísticas do banco de dados local** para entender a quantidade e distribuição de tarefas históricas.
US12	UC04, UC06	Como **Scrum Master**, eu quero **processar múltiplas tarefas em paralelo** para obter estimativas rapidamente mesmo com grandes backlogs.
US13	UC07	Como **Scrum Master**, eu quero **sincronizar dados do GitHub incluindo custom fields** para ter acesso a Worker Hours e Story Points históricos.

Histórias do Product Owner:

ID	Caso de Uso	História
US14	UC01, UC02	Como **Product Owner**, eu quero **configurar a CLI SET com acesso ao repositório do produto** para poder consultar estimativas e gerar relatórios estratégicos.
US15	UC06	Como **Product Owner**, eu quero **gerar relatórios executivos de estimativas em múltiplos formatos** (CSV, JSON, Markdown) para comunicar timelines realistas aos stakeholders e management.
US16	UC04	Como **Product Owner**, eu quero **estimar épicos completos via batch processing** para comparar esforço entre diferentes features e priorizar com base em ROI.
US17	UC06	Como **Product Owner**, eu quero **exportar dados em formatos compatíveis com Jira e GitHub Projects** para integrar as estimativas ao planejamento estratégico do produto.

US18	UC04, UC06	Como **Product Owner**, eu quero **ver estatísticas agregadas de estimativas em lote** (total de horas, distribuição por tamanho, nível de confiança) para tomar decisões informadas sobre releases.

2.4 Diagrama de Sequência do Sistema e Contrato de Operações

Esta seção apresenta os Diagramas de Sequência do Sistema (DSS) que descrevem as principais interações entre os atores e o sistema SET CLI em uma visão de caixa-preta. Os DSS mostram apenas as mensagens trocadas entre o ator externo e o sistema, sem revelar a estrutura interna dos componentes.

DSS 01: Configurar Sistema

Elemento	**Descrição**
Operação	configurar(token: string, repositorio: string)
Referências	UC01, UC02, US01, US08, US14
Pré-condições	• Usuário possui token GitHub válido • Usuário tem permissões no repositório especificado
Pós-condições	• Token armazenado de forma segura no sistema • Repositório padrão configurado • Sistema pronto para operação

A Figura 2.2 representa o primeiro contato do usuário com o sistema SET CLI, onde é necessário realizar a configuração inicial para que a ferramenta possa funcionar adequadamente. Este fluxo é fundamental pois estabelece a conexão com o GitHub API e define os parâmetros básicos de funcionamento.

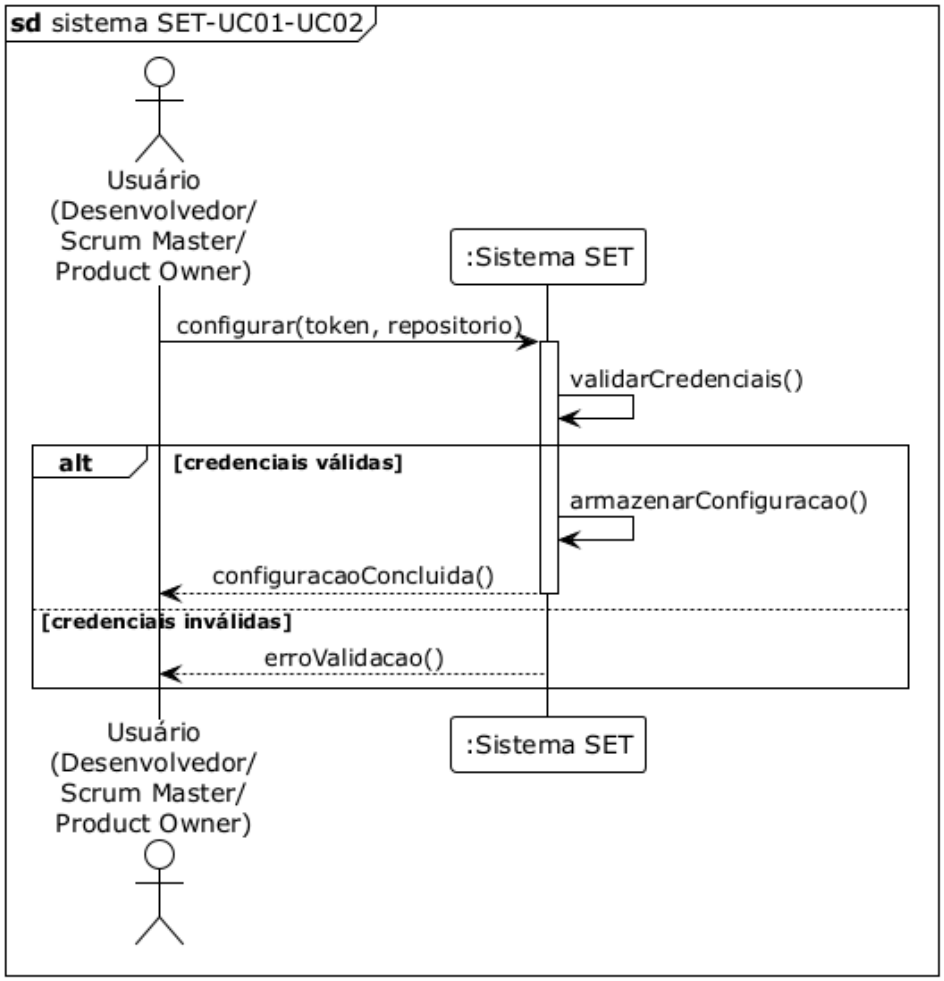


Figura 2.2 - Diagrama de Sequência do Sistema: Configurar Sistema

Descrição do Fluxo:

1. O ator (Desenvolvedor/Scrum Master/Product Owner) inicia o comando de configuração através de um wizard interativo (*--initial*) ou via flags individuais *(--github-token, --default-repo, --ai-provider, --ai-key*)
2. O sistema valida o formato do token GitHub fornecido (formato ghp_...) e opcionalmente o repositório (padrão owner/repo)
3. O sistema armazena a configuração de forma segura no arquivo ~/.set/config.yaml, incluindo token GitHub, repositório padrão, provedor de IA (OpenAI ou Claude) e suas respectivas chaves de API
4. O sistema confirma que a configuração foi bem-sucedida exibindo um resumo dos valores configurados (com o token mascarado por segurança)

DSS 02: Validar Configuração

Elemento	Descrição
Operação	validarConfiguracao()
Referências	UC08, US07
Pré-condições	• Configuração previamente realizada • Arquivo de configuração existe
Pós-condições	• Status de validação retornado ao usuário • Erros identificados reportados (se existirem) • Conectividade com GitHub verificada

A Figura 2.3 apresenta a interação para validação da configuração existente do SET CLI, permitindo ao usuário verificar se as credenciais e permissões estão corretas antes de iniciar operações com o GitHub.

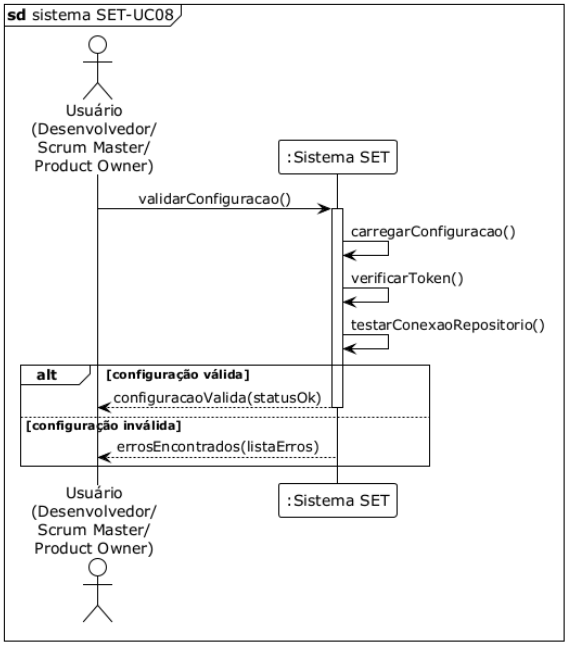


Figura 2.3 - Diagrama de Sequência do Sistema: Validar Configuração

Descrição do Fluxo:

1. O ator executa o comando *set configure --validate* para verificar a integridade da configuração atual
2. O sistema carrega a configuração armazenada em *~/.set/config.yaml* e verifica se o token GitHub está presente
3. O sistema realiza uma chamada à API do GitHub para validar o token, verificando sua validade, scopes (permissões) disponíveis e nome de usuário autenticado
4. O sistema testa a conexão com o repositório padrão configurado (se houver), verificando se o token tem permissões de leitura sobre o repositório
5. O sistema retorna um relatório detalhado incluindo: status de validação (válido ou inválido), nome de usuário autenticado, scopes disponíveis (repo, read:org, etc.), status de acesso ao repositório, e lista de erros específicos caso a validação falhe

DSS 03: Estimar Tarefa Individual

Elemento	Descrição
Operação	estimar(descricao: string, opcoes: OpcoesEstimativa)
Referências	UC03, US02, US03, US05, US06
Pré-condições	• Sistema configurado • Descrição da tarefa fornecida
Pós-condições	• Estimativa calculada e exibida • Nível de confiança determinado • Tarefas similares identificadas (se disponíveis) • Estimativa registrada no histórico

A Figura 2.4 ilustra a interação para estimativa de uma tarefa individual utilizando IA e dados históricos, representando o caso de uso mais frequente do sistema.

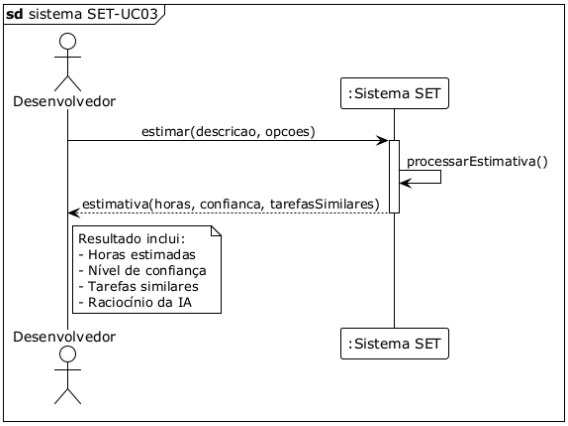


Figura 2.4 - Diagrama de Sequência do Sistema: Estimar Tarefa Individual

Descrição do Fluxo:

1. O ator (principalmente Desenvolvedor) fornece o título da tarefa e opcionalmente: descrição detalhada (*--description*), labels para categorização (*--labels*), contexto adicional (*--context*), e flags de controle como *--similar* para exibir tarefas similares ou *--no-ai* para usar apenas dados históricos
2. O sistema processa a solicitação através de múltiplas estratégias: busca por tarefas similares no histórico local (BoltDB) usando algoritmos de similaridade textual, consulta ao provedor de IA configurado (OpenAI GPT ou Claude) para análise semântica da tarefa, análise de custom fields do GitHub Projects (size, story points, worker hours), e cálculo estatístico baseado em performance histórica
3. O sistema retorna uma estimativa consolidada contendo: número de horas estimadas (média ponderada entre IA e dados históricos), nível de confiança da estimativa (0-100%), lista de tarefas similares encontradas (título, labels, tempo real de conclusão), raciocínio da IA explicando a estimativa, além da opção de exportar o resultado em múltiplos formatos (text, JSON, CSV)

DSS 04: Processar Lote de Tarefas

Elemento	**Descrição**
Operação	processarLote(arquivoEntrada: string, formatoSaida: string)
Referências	UC04, US09, US12, US16, US18
Pré-condições	• Sistema configurado

	• Arquivo de entrada válido (JSON/CSV) • Formato de saída suportado
Pós-condições	• Todas as tarefas estimadas • Estatísticas agregadas calculadas (total horas, distribuição, confiança média) • Arquivo de saída gerado no formato especificado • Estimativas registradas no histórico

A Figura 2.5 mostra a interação para processamento em lote de estimativas, funcionalidade estratégica utilizada por Scrum Masters durante Sprint Planning e Product Owners no planejamento de releases.

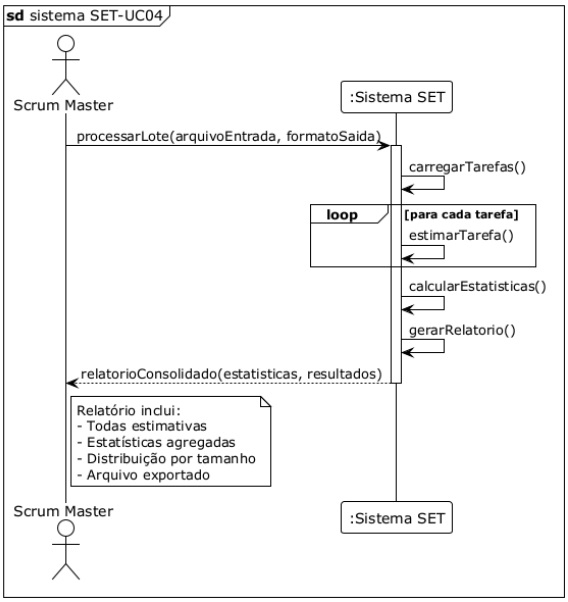


Figura 2.5 - Diagrama de Sequência do Sistema: Processar Lote de Tarefas

Descrição do Fluxo:

1. O ator (principalmente Scrum Master ou Product Owner) fornece arquivo de entrada em formato JSON ou CSV (*--file tasks.json*) contendo múltiplas tarefas com seus respectivos metadados (id, título, descrição, labels, prioridade, assignee), especifica o formato de saída desejado (*--format text|json|csv|markdown*), define o número de workers para processamento paralelo (*--workers 10*), e opcionalmente ativa indicador de progresso (*--progress*)

2. O sistema carrega e valida o arquivo de entrada, processa todas as tarefas em paralelo utilizando pool de workers configurável (padrão 5 workers), aplica o mesmo algoritmo de estimativa usado no comando individual (IA + similaridade + dados históricos) para cada tarefa, coleta métricas de performance durante o processamento, e salva cada estimativa no histórico local (BoltDB)
3. O sistema gera um relatório consolidado contendo: estatísticas agregadas (total de horas, média, mediana, desvio padrão), distribuição de tarefas por tamanho (pequenas/médias/grandes), nível de confiança médio do lote, breakdown por label/categoria, tempo total de processamento, e lista completa de estimativas individuais
4. O sistema exporta o relatório no formato especificado: texto formatado para terminal, JSON estruturado para integração com APIs, CSV para análise em planilhas, ou Markdown para documentação, salvando em arquivo (*--output report.md*) ou exibindo no stdout

DSS 05: Sincronizar com GitHub

Elemento	Descrição
Operação	sincronizar(repositorio: string, incluirCustomFields: boolean)
Referências	UC07, US13
Pré-condições	• Sistema configurado com token válido • Repositório acessível • Conectividade com GitHub API
Pós-condições	• Issues fechadas importadas para histórico • Pull Requests mesclados importados • Custom fields (Worker Hours, Story Points) sincronizados se solicitado • Timestamp de última sincronização atualizado • Base de dados local atualizada

A Figura 2.6 apresenta a interação para sincronização de dados históricos do GitHub com o banco de dados local, permitindo análise offline e consultas rápidas sem consumir rate limit da API.

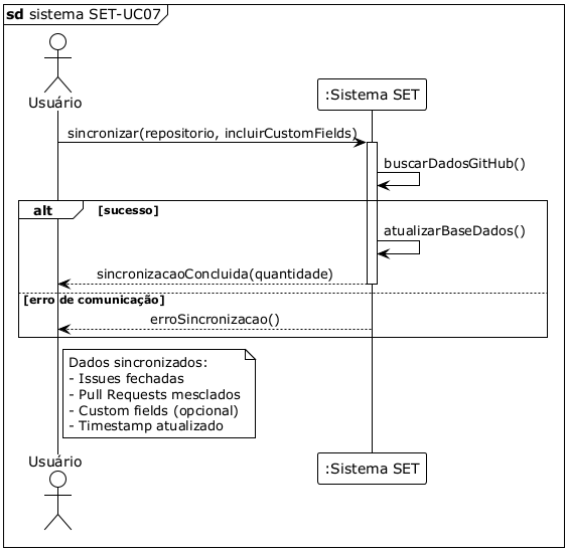


Figura 2.6 - Diagrama de Sequência do Sistema: Sincronizar com GitHub

Descrição do Fluxo:

1. O ator executa o comando set sync com opções de controle: sincronização completa (*--full*) ignorando timestamp da última sincronização, sincronização forçada (*--force*) mesmo se executada recentemente, filtros específicos (*--issues-only* ou *--prs-only*), e inclusão de custom fields do GitHub Projects V2 (*--custom-fields*) para obter size, story points, e estimativas
2. O sistema valida a configuração (token e repositório), determina o modo de sincronização (incremental usando timestamp da última sync ou completo desde o início), e estabelece conexão autenticada com a API REST do GitHub v3 e GraphQL API (para custom fields)
3. O sistema busca dados do repositório configurado através de chamadas paginadas à API: issues fechadas com seus metadados (título, descrição, labels, assignee, milestone, created_at, closed_at), pull requests mesclados (merged PRs), custom fields do GitHub Projects V2 se solicitado (requer permissões adicionais), e calcula duração real de cada tarefa (closed_at - created_at)
4. O sistema persiste os dados no BoltDB local (*~/.set/data.db*) organizados em buckets (issues, prs, custom_fields), atualiza o timestamp da última sincronização, exibe barra de progresso durante o download, e retorna um relatório de sincronização contendo: quantidade de issues importadas, quantidade de PRs importados, quantidade de custom fields sincronizados, duração total da operação, e próximo horário recomendado para sync (respeitando rate limit)

DSS 06: Exportar Dados

Elemento	**Descrição**

Operação	exportar(formato: FormatoExportacao, filtros: FiltrosExportacao)
Referências	UC06, US10, US12, US15, US17, US18
Pré-condições	• Dados disponíveis no sistema • Formato de exportação suportado (CSV, JSON, Markdown, Jira, GitHub, Excel)
Pós-condições	• Arquivo gerado no formato especificado • Dados filtrados conforme critérios • Estrutura compatível com ferramenta de destino (Jira/GitHub)

A Figura 2.7 ilustra o processo de exportação de dados históricos e estimativas em diferentes formatos para integração com ferramentas de gestão de projetos e análise de dados.

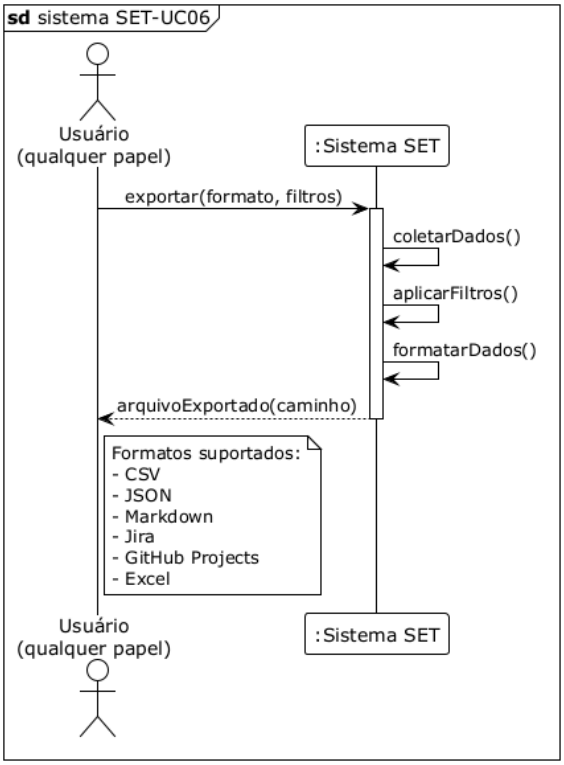


Figura 2.7 - Diagrama de Sequência do Sistema: Exportar Dados

Descrição do Fluxo:

1. O ator (qualquer papel) executa set export especificando formato de saída (*--format csv|json|jira|github|excel|markdown*), arquivo de destino (*--output issues.csv*), e filtros opcionais: filtro por label (*--filter bug*), intervalo de datas (*--date-from 2025-09-19 --date-to 2025-11-16*), e inclusão/exclusão de issues fechadas (*--include-closed*)
2. O sistema consulta o BoltDB local aplicando os filtros especificados: busca issues/PRs no intervalo de datas, filtra por labels quando especificado, inclui ou exclui items fechados conforme configurado, e carrega custom fields associados quando disponíveis
3. O sistema transforma os dados para o formato solicitado aplicando mapeamentos específicos: CSV genérico com todas as colunas disponíveis, JSON estruturado com objetos aninhados completos, formato Jira-compatível seguindo schema de importação do Jira Cloud (Summary, Description, Issue Type, Priority, Labels), formato GitHub Projects seguindo CSV import template do GitHub, formato Excel com colunas enriquecidas incluindo fórmulas e custom fields, ou Markdown table para documentação legível
4. O sistema persiste o arquivo exportado no caminho especificado ou exibe no stdout se *--output* não for fornecido, valida a integridade dos dados exportados, e retorna confirmação contendo: quantidade de registros exportados, tamanho do arquivo gerado, path absoluto do arquivo, e preview das primeiras linhas quando formato é texto

DSS 07: Analisar Histórico

Elemento	**Descrição**
Operação	`analisarHistorico(filtros: FiltrosAnalise, limite: number)`
Referências	UC05, US04, US06, US10, US11
Pré-condições	• Dados históricos disponíveis • Banco de dados local acessível
Pós-condições	• Estatísticas calculadas (total de tarefas, distribuição por tipo, precisão média) • Padrões identificados (tendências de super/subestimação) • Insights gerados sobre performance de estimativas

• Relatório exibido ao usuário

A Figura 2.8 mostra a interação para análise e inspeção de dados históricos armazenados localmente, permitindo visualização detalhada de issues, PRs e custom fields sincronizados.

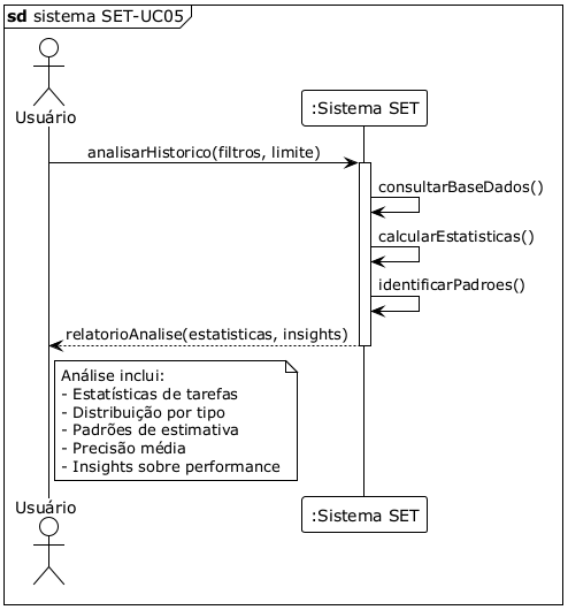


Figura 2.8 - Diagrama de Sequência do Sistema: Analisar Histórico

Descrição do Fluxo:

1. O ator executa *set inspect* com diferentes modos de operação: visualizar issue específica (*--issue 123*), visualizar pull request específico (*--pr 45*), listar todos os itens armazenados (*--list*), filtrar apenas itens com custom fields (*--custom*), limitar quantidade de resultados (*--limit 50*), ou exportar em JSON (*--json*)
2. O sistema verifica a existência do banco de dados local (*~/.set/data.db*) e retorna erro caso não exista (instruindo executar *set sync* primeiro), abre conexão read-only com o BoltDB, e executa queries nos buckets apropriados (issues, prs, custom_fields) conforme o modo solicitado
3. O sistema processa e enriquece os dados recuperados: calcula estatísticas agregadas quando em modo list (total de itens, distribuição por status, breakdown por label), formatar datas em formato human-readable, máscara informações sensíveis se houver, organiza custom fields em estrutura hierárquica, e calcula métricas derivadas (duração média, velocidade do time, precisão de estimativas)
4. O sistema exibe os resultados formatados: tabela colorida no terminal para modo list, detalhes completos para inspeção individual (todos os campos, timeline de eventos, custom fields, links relacionados), JSON estruturado para modo *--json*, estatísticas resumidas (total de tarefas por tipo, distribuição temporal, padrões identificados), e insights sobre performance de estimativas comparando tempo estimado vs. tempo real

3. Modelos de Projeto

3.1 Diagrama de Classes

Esta seção apresenta os diagramas de classes do sistema SET CLI organizados por pacotes Go (packages), seguindo a estrutura modular da implementação. Cada diagrama mostra as classes, interfaces e relacionamentos específicos de um pacote, refletindo o código efetivamente implementado.

A arquitetura segue os princípios de Clean Architecture e Domain-Driven Design (DDD), com clara separação de responsabilidades entre as camadas de domínio, aplicação, infraestrutura e interface.

3.1.1 Pacote AI (internal/ai)

O pacote internal/ai define a abstração para provedores de inteligência artificial e os tipos relacionados ao processo de estimativa assistida por IA. Este pacote encapsula toda a lógica de integração com modelos de linguagem de grande escala (LLMs), permitindo que diferentes provedores (OpenAI, Claude, modelos locais) sejam utilizados de forma intercambiável através da interface AIProvider. O design orientado a interfaces facilita a extensibilidade e testabilidade do sistema, permitindo mock objects durante testes unitários e a adição de novos provedores de IA sem modificar o código cliente. As estruturas de dados foram projetadas como Value Objects imutáveis para garantir thread-safety durante processamento concorrente de múltiplas estimativas.

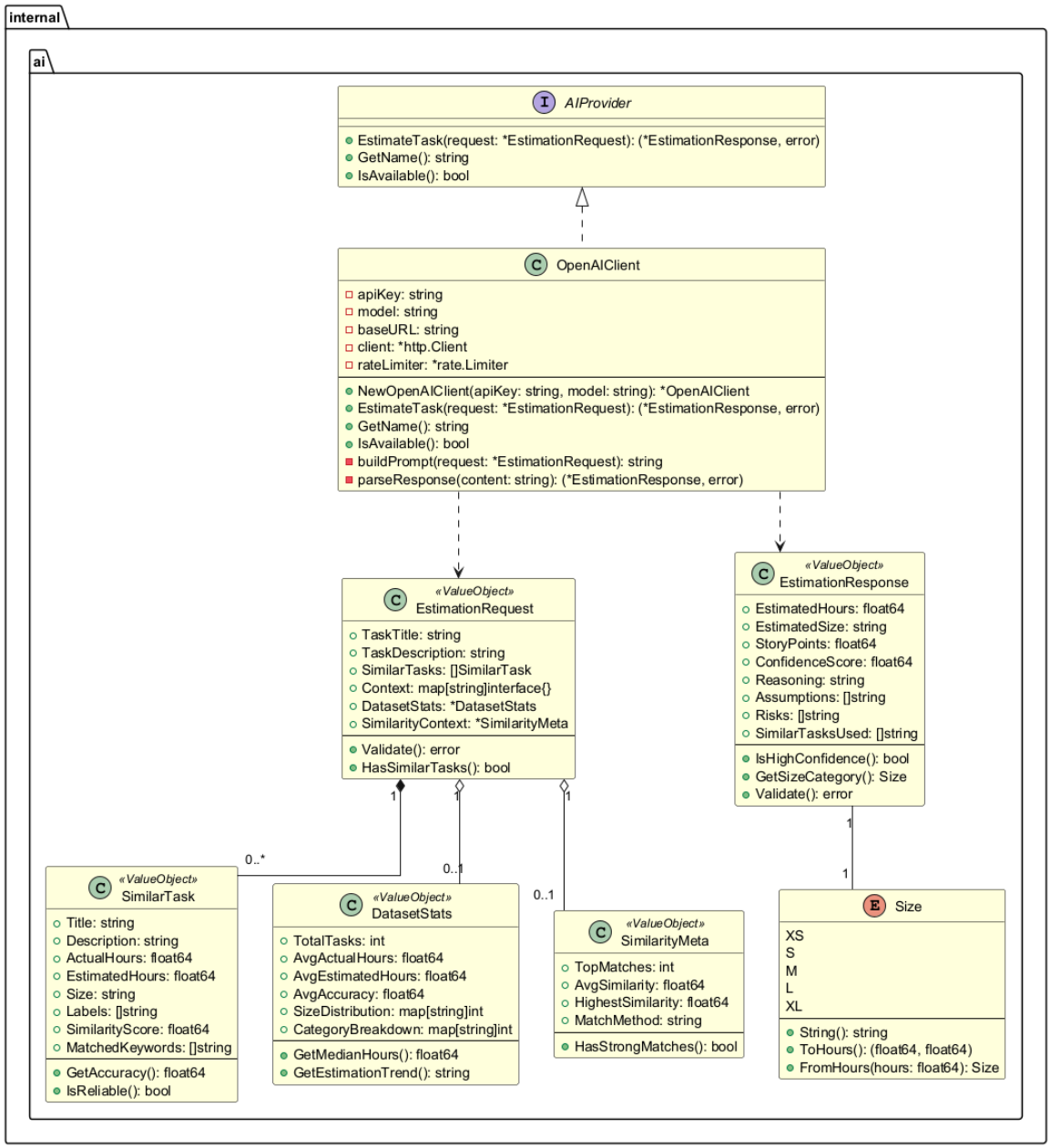


Figura 3.1.1 - Diagrama de Classes: Pacote AI

Componentes Principais:

- **AIProvider (Interface):** Abstração que permite diferentes implementações de provedores de IA (OpenAI, Claude, modelos locais). Define o contrato para estimação de tarefas.
- **OpenAIClient:** Implementação concreta do AIProvider que integra com a API da OpenAI. Gerencia rate limiting, constrói prompts contextualizados e processa respostas do modelo GPT.
- **EstimationRequest**: Value Object que encapsula todo o contexto necessário para gerar uma estimativa: descrição da tarefa, tarefas similares do histórico, estatísticas do dataset e metadados de similaridade.

- **EstimationResponse:** Value Object que representa o resultado da estimativa gerada pela IA, incluindo horas estimadas, story points, tamanho (XS-XL), nível de confiança, raciocínio detalhado e riscos identificados.
- **Size (Enum):** Enumerações de tamanhos no formato t-shirt sizing (XS, S, M, L, XL), com conversão para faixas de horas.

3.1.2 Pacote Estimator (internal/estimator)

O pacote internal/estimator contém a lógica central de estimativa, incluindo o motor de similaridade baseado em TF-IDF e o orquestrador que combina dados históricos com IA. Este é o coração do domínio da aplicação, onde reside a regra de negócio principal: combinar análise histórica com inteligência artificial para gerar estimativas precisas. O SimilarityEngine implementa algoritmos de processamento de linguagem natural (NLP) usando vetorização TF-IDF e similaridade de cosseno para encontrar padrões em tarefas passadas. A classe Estimator atua como um orquestrador (facade), coordenando múltiplos componentes (storage, similarity engine, AI provider) para produzir estimativas calibradas. O design suporta processamento em lote com execução paralela via goroutines, aproveitando as capacidades de concorrência nativa do Go para alta performance.

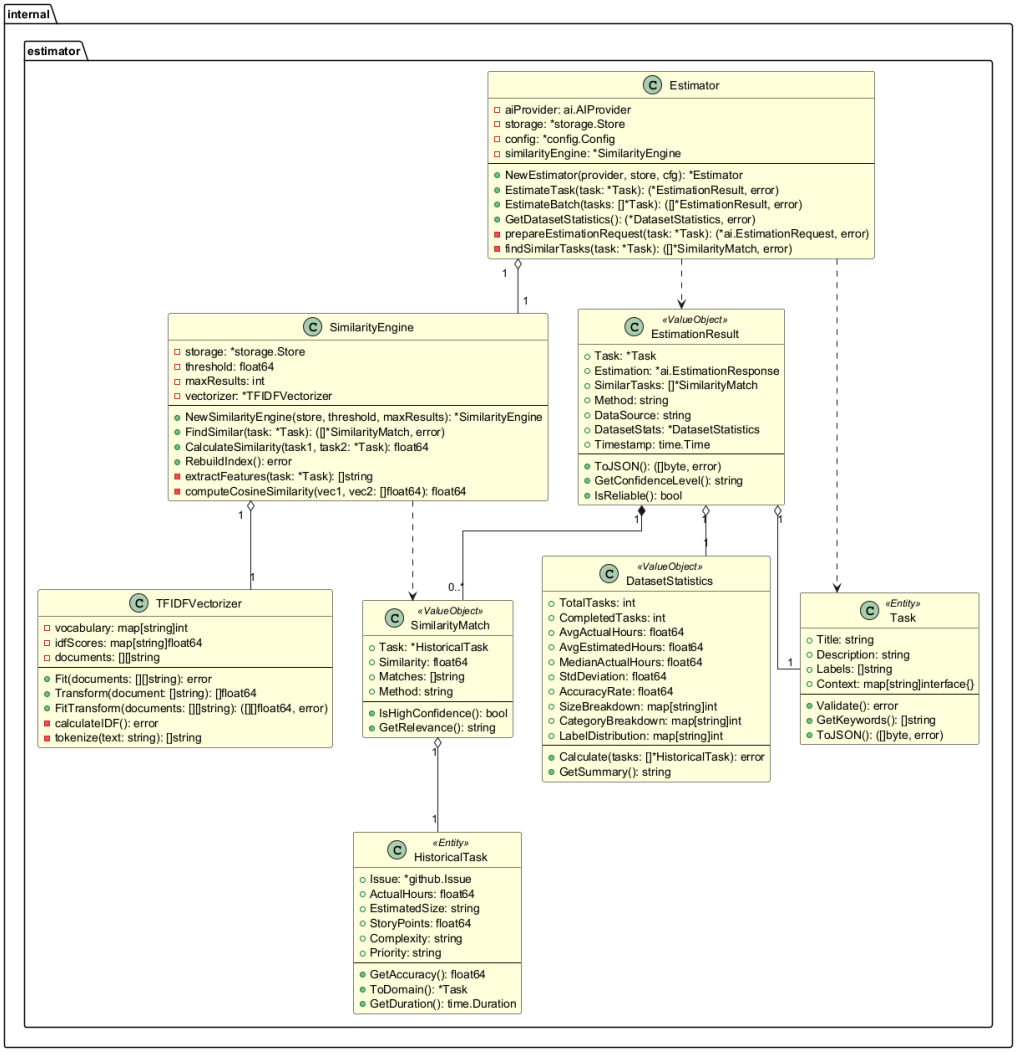


Figura 3.1.2 - Diagrama de Classes: Pacote Estimator

Componentes Principais:

- **Estimator:** Serviço principal que orquestra o processo de estimativa. Busca tarefas similares no histórico, calcula estatísticas do dataset, solicita estimativa à IA e consolida os resultados.
- **SimilarityEngine:** Motor de busca por similaridade que utiliza vetorização TF-IDF e similaridade de cosseno para encontrar tarefas históricas semanticamente similares à tarefa sendo estimada.
- **TFIDFVectorizer:** Componente de NLP (Natural Language Processing) que implementa o algoritmo Term Frequency-Inverse Document Frequency para transformar texto em vetores numéricos comparáveis.
- **Task:** Entidade que representa uma tarefa a ser estimada, com título, descrição, labels e contexto adicional.
- **HistoricalTask:** Entidade que representa uma tarefa concluída do histórico, com dados reais de esforço (horas, story points, tamanho) para calibração das estimativas.
- **EstimationResult:** Value Object que encapsula o resultado completo do processo de estimativa, incluindo a estimativa da IA, tarefas similares encontradas e estatísticas do dataset.
- **DatasetStatistics:** Value Object com métricas estatísticas agregadas do histórico (média, mediana, desvio padrão, distribuição por tamanho/categoria).

3.1.3 Pacote GitHub (internal/github)

O pacote internal/github implementa a integração com a API do GitHub para sincronização de dados históricos de repositórios. Este pacote atua como uma camada de adaptação (Adapter Pattern) entre o domínio interno da aplicação e a API externa do GitHub, isolando detalhes de implementação da API REST/GraphQL e transformando entidades do GitHub (Issues, Pull Requests) em objetos de domínio (HistoricalTask). O cliente HTTP implementa estratégias de resiliência incluindo rate limiting para respeitar os limites da API do GitHub (5000 requisições/hora), retry logic com backoff exponencial, e tratamento robusto de erros de rede. O suporte a GitHub Projects V2 permite sincronização de custom fields (story points, horas estimadas) que enriquecem o dataset histórico e melhoram a precisão das estimativas.

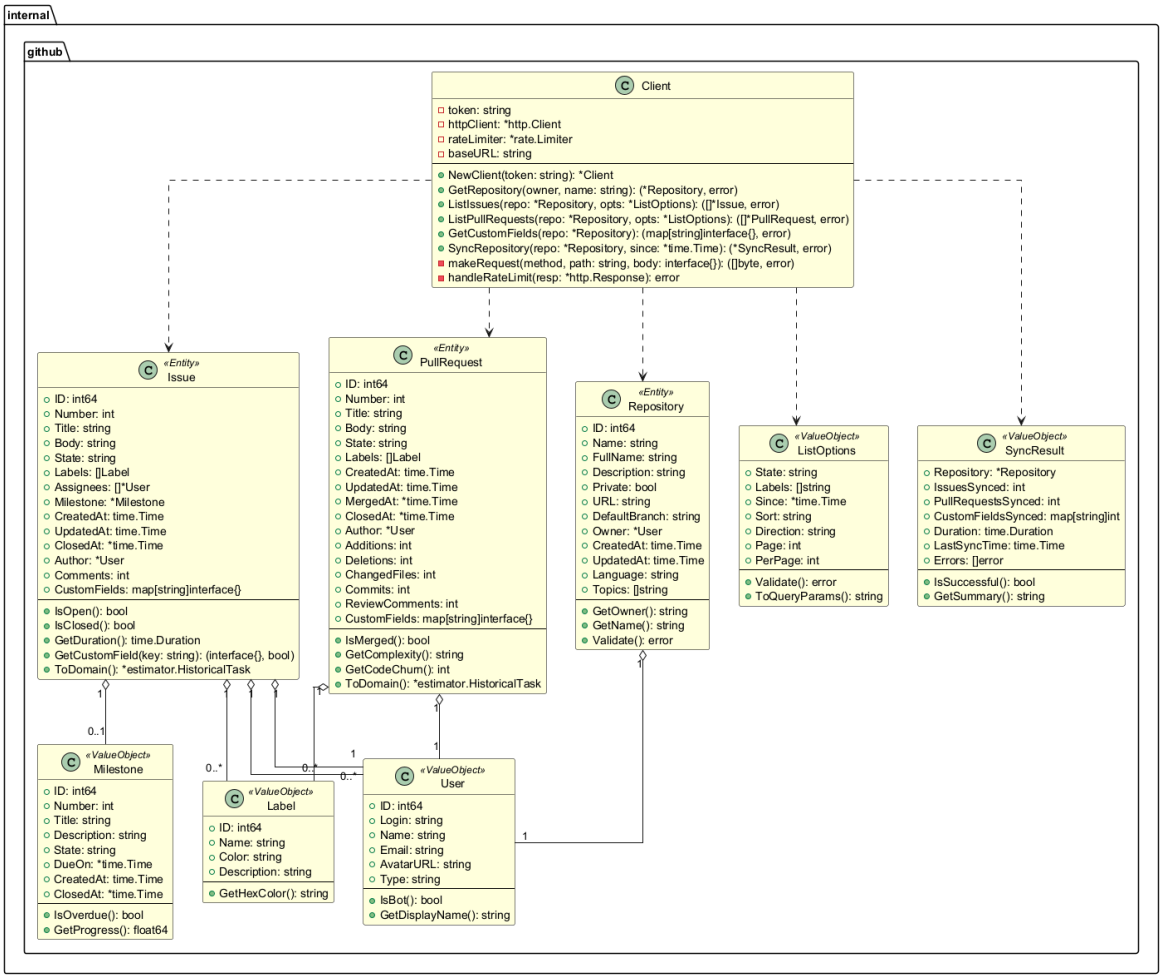


Figura 3.1.3 - Diagrama de Classes: Pacote GitHub

Componentes Principais:

- **Client:** Cliente HTTP que gerencia comunicação com as APIs REST e GraphQL do GitHub. Implementa rate limiting, retry logic e tratamento de erros para operações de sincronização.
- **Repository:** Entidade que representa um repositório GitHub com metadados como nome, descrição, linguagem, tópicos e owner.
- **Issue:** Entidade que representa uma issue do GitHub. Contém informações completas incluindo labels, assignees, milestone, datas de ciclo de vida e custom fields do GitHub Projects V2. Pode ser convertida para HistoricalTask.
- **PullRequest:** Entidade que representa um pull request do GitHub. Inclui métricas de complexidade (linhas adicionadas/deletadas, arquivos alterados) que auxiliam na estimativa. Pode ser convertida para HistoricalTask.
- **User:** Value Object representando um usuário do GitHub (autor, assignee).
- **Label, Milestone:** Value Objects com metadados adicionais de organização.
- **ListOptions:** Value Object para parametrizar consultas à API (filtros, ordenação, paginação).

- **SyncResult:** Value Object com resultado de operação de sincronização (quantidade sincronizada, duração, erros).

3.1.4 Pacote Storage (internal/storage)

O pacote internal/storage implementa o padrão Repository para persistência local usando BoltDB. A escolha do BoltDB como banco de dados embarcado foi estratégica para uma aplicação CLI: não requer instalação ou configuração externa, armazena todos os dados em um único arquivo portável, oferece transações ACID completas, e proporciona excelente performance de leitura através de índices B+tree otimizados. O padrão Repository abstrai completamente os detalhes de persistência, permitindo que a camada de domínio trabalhe com interfaces de alto nível sem conhecimento de SQL ou estruturas de armazenamento. A implementação utiliza buckets separados para issues, pull requests e metadados, com serialização JSON para flexibilidade no schema. As operações em lote (Batch) minimizam I/O de disco durante sincronizações grandes de repositórios.

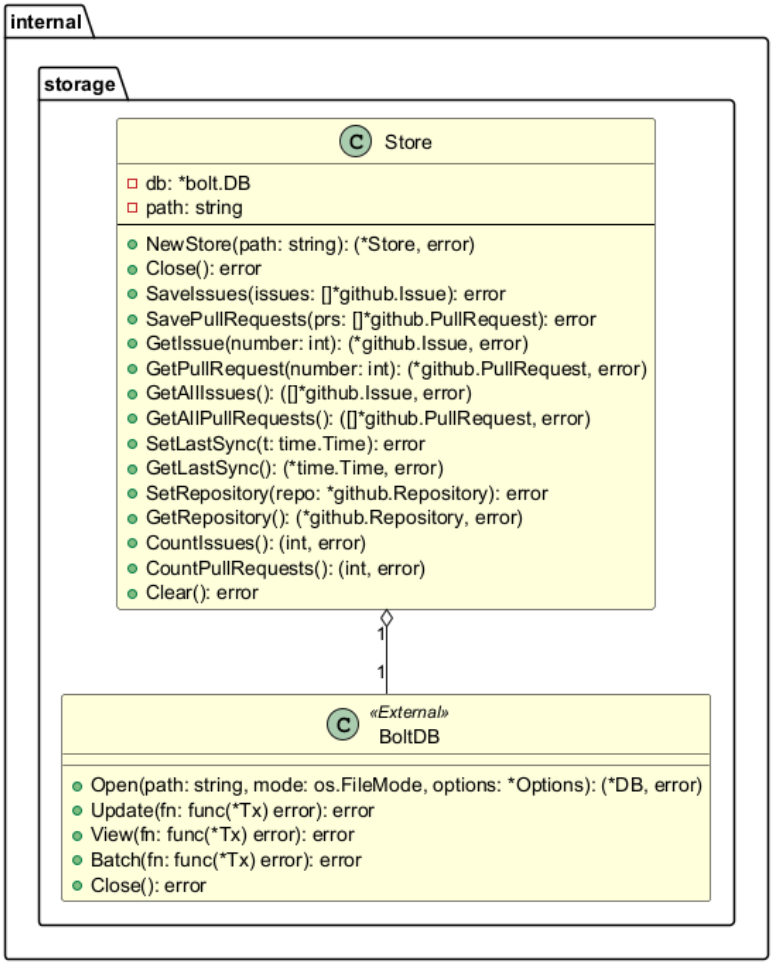


Figura 3.1.4 - Diagrama de Classes: Pacote Storage

Componentes Principais:

1. **Store:** Implementação do padrão Repository que abstrai o acesso a dados. Gerencia três buckets no BoltDB: issues (issues do GitHub), prs (pull requests) e metadata (informações de sincronização).
2. **BoltDB:** Banco de dados embarcado key-value utilizado para persistência local. Características: zero configuração, ACID compliant, performance otimizada para leitura via B+tree, ideal para aplicações CLI.

Operações Principais:

- Persistência de issues e pull requests com serialização JSON
- Consultas por número ou listagem completa
- Controle de timestamp da última sincronização
- Operações em lote para performance
- Contadores e estatísticas rápidas

3.1.5 Pacote Config (internal/config)

O pacote internal/config gerencia todas as configurações do sistema utilizando Viper para suportar múltiplas fontes (arquivo, variáveis de ambiente, flags CLI). A configuração segue uma hierarquia bem definida onde valores mais específicos sobrescrevem valores mais genéricos: defaults do código < arquivo YAML (~/.set.yaml) < variáveis de ambiente < flags de linha de comando. Esta abordagem flexível permite diferentes estratégias de configuração conforme o contexto: desenvolvedores usam arquivos locais, ambientes de CI/CD usam variáveis de ambiente, e usuários finais podem sobrescrever valores via flags sem editar arquivos. A classe Config atua como um Aggregate Root no contexto de DDD, garantindo validação consistente de todas as configurações relacionadas e evitando estados inválidos (por exemplo, API key vazia quando AI está habilitada).

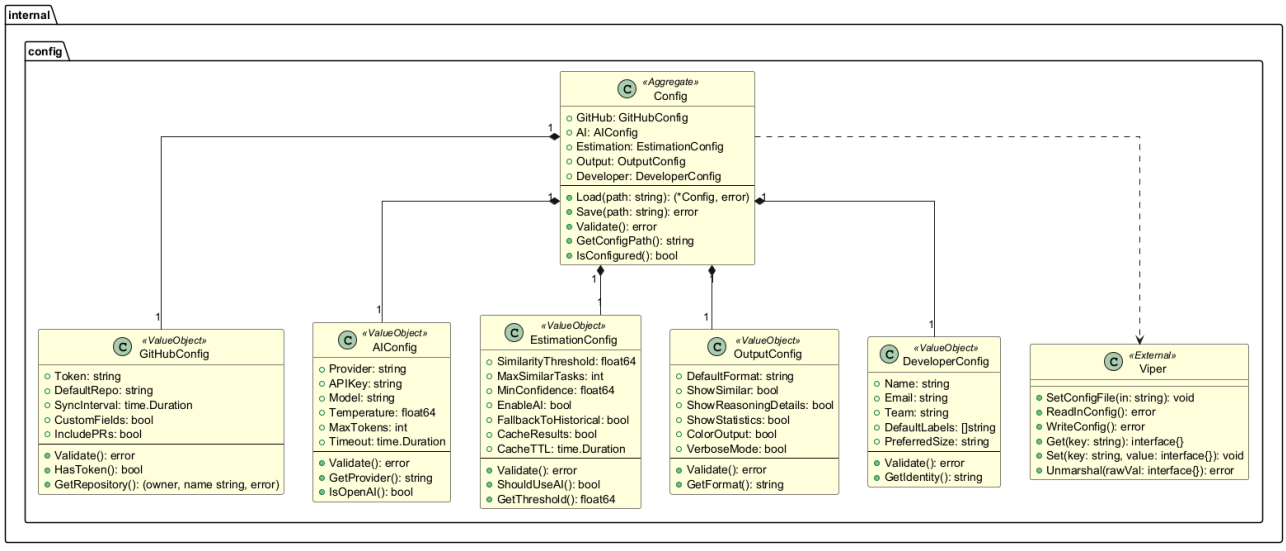


Figura 3.1.5 - Diagrama de Classes: Pacote Config

Componentes Principais:

- **Config:** Aggregate Root que encapsula todas as configurações do sistema. Arquivo de configuração localizado em ~/.set.yaml.
- **GitHubConfig:** Configurações de integração com GitHub (token, repositório padrão, intervalo de sincronização, custom fields).
- **AIConfig:** Configurações do provedor de IA (OpenAI), incluindo API key, modelo (GPT-4/GPT-3.5), temperatura e limites.
- **EstimationConfig:** Parâmetros para controle do processo de estimativa (threshold de similaridade, máximo de tarefas similares, confiança mínima, cache).
- **OutputConfig:** Preferências de formatação de saída (formato padrão, verbosidade, cores, estatísticas).
- **DeveloperConfig:** Informações do desenvolvedor (nome, email, equipe, labels padrão).

Hierarquia de Configuração: defaults <- arquivo <- variáveis de ambiente <- flags CLI

3.1.6 Padrões de Design Aplicados

A modelagem do sistema aplica diversos padrões consolidados:

Domain-Driven Design (DDD):

- **Entities** com identidade: Task, HistoricalTask, Issue, PullRequest, Repository
- **Value Objects** imutáveis: EstimationRequest, EstimationResponse, SimilarTask, Config, etc.
- **Aggregate Roots**: Config (configurações), SeedData (dados de exemplo)
- **Repositories**: Store (padrão Repository para abstração de persistência)

Clean Architecture:

- Separação em camadas: Domain (estimator, ai) → Application → Infrastructure (github, storage) → Interface (cmd)
- Dependências apontando para dentro (regras de negócio independentes de frameworks)
- Uso de interfaces para inversão de dependências (AIProvider, Repository implícito)

Padrões GoF:

Strategy: AIProvider permite trocar algoritmo de estimativa

Factory: Construtores como NewEstimator, NewClient, NewStore

Adapter: Conversões ToDomain() adaptam entidades externas (GitHub) para domínio interno

3.2 Diagramas de Sequência

Esta seção apresenta os Diagramas de Sequência detalhados que descrevem as interações entre os componentes internos do sistema SET CLI. Diferentemente dos Diagramas de Sequência do Sistema (DSS) apresentados na Seção 2.4 do documento de requisitos, estes diagramas mostram a visão interna (caixa-branca) do sistema, revelando como os componentes colaboram para realizar cada operação.

3.2.1 Configurar Sistema

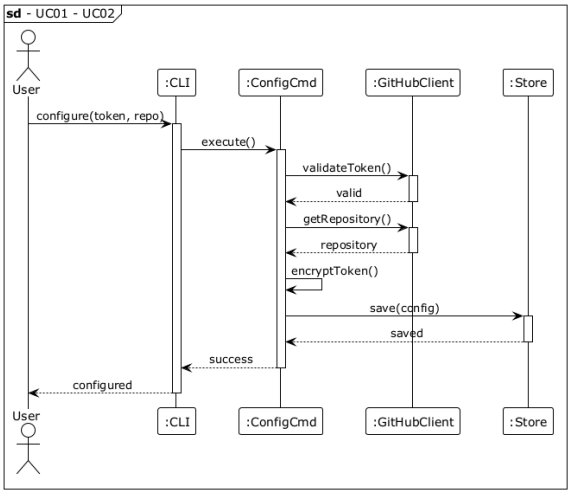


Figura 3.2 - Diagrama de Sequência: Configurar Sistema

3.2.2 Estimar Tarefa Individual

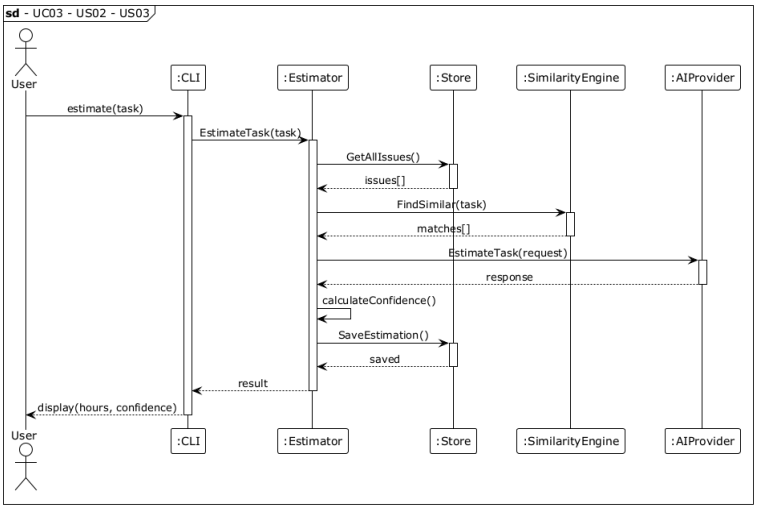


Figura 3.3 - Diagrama de Sequência: Estimar Tarefa Individual

3.2.3 Estimar em Lote

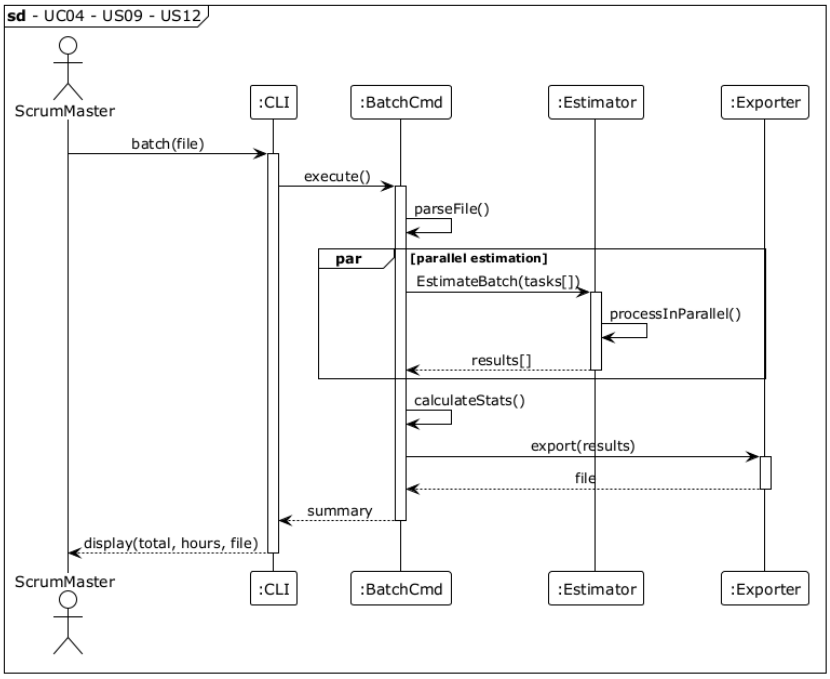


Figura 3.4 - Diagrama de Sequência: Estimar em Lote

3.2.4 Sincronizar Dados GitHub

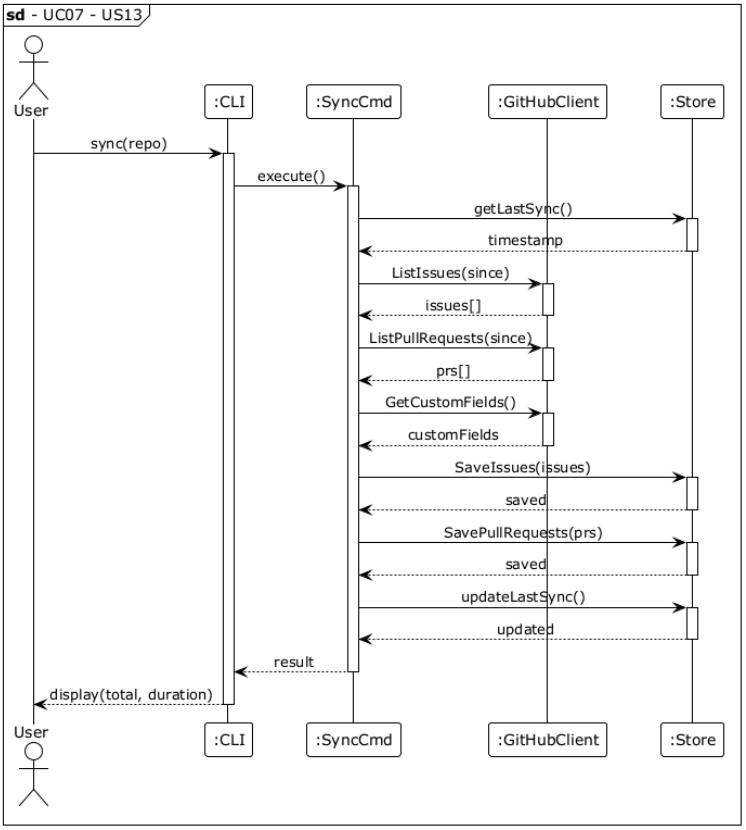


Figura 3.5 - Diagrama de Sequência: Sincronizar Dados GitHub

3.3 Diagramas de Comunicação

Os diagramas de comunicação complementam os diagramas de sequência, focando na estrutura das relações entre objetos e a ordem das mensagens trocadas.

3.3.1 Diagrama de Comunicação - Estimar Tarefa

#TODO:Ainda trabalhando nele

3.4 Arquitetura

Esta seção apresenta a modelagem da arquitetura do sistema SET CLI utilizando o modelo C4 (Context, Containers, Components, Code), que fornece uma abordagem hierárquica para visualizar a arquitetura de software em diferentes níveis de abstração.

A arquitetura do sistema SET CLI foi projetada seguindo os princípios de Clean Architecture, garantindo separação de responsabilidades, baixo acoplamento e alta coesão. O sistema utiliza Go como linguagem principal, aproveitando suas características de performance, concorrência e facilidade de distribuição multiplataforma.

3.4.1 Diagrama de Contexto (C4 Level 1)

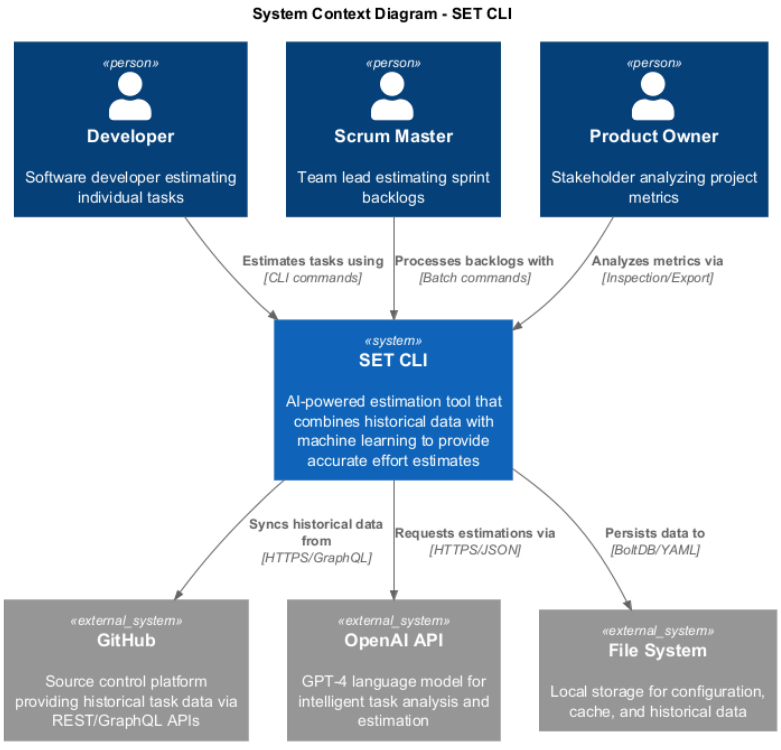


Figura 3.10 - Diagrama de Contexto C4

O diagrama de contexto mostra a visão de mais alto nível do sistema SET CLI, identificando os atores principais (Developer, Scrum Master, Product Owner) e os sistemas externos com os quais interage (GitHub API, OpenAI API, File System).

3.4.2 Diagrama de Contêineres (C4 Level 2)

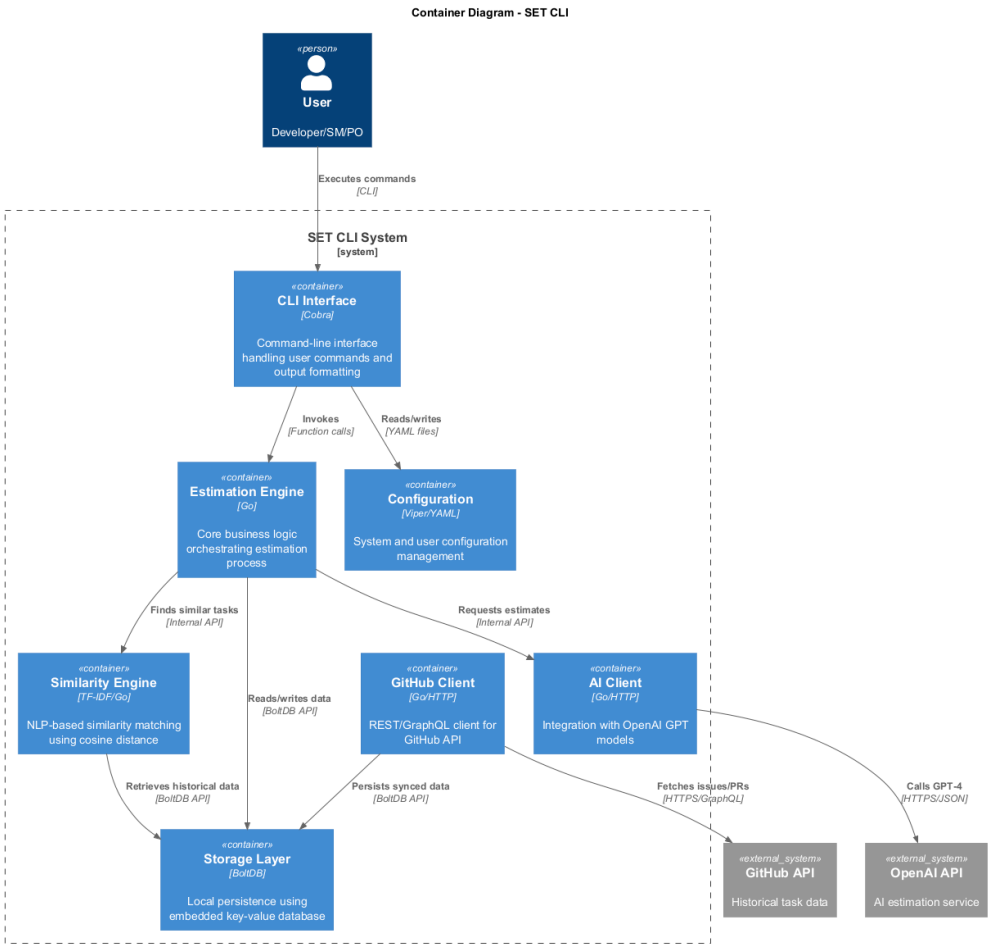


Figura 3.11 - Diagrama de Contêineres C4

O diagrama de contêineres decompõe o sistema SET CLI em seus principais componentes executáveis, mostrando como a CLI Interface, Estimation Engine, Similarity Engine, AI Client, GitHub Client, Storage Layer e Configuration trabalham em conjunto para fornecer funcionalidade de estimativa.

3.4.3 Padrões Arquiteturais Adotados

Clean Architecture: O sistema segue os princípios da Clean Architecture, organizando o código em camadas concêntricas onde as dependências apontam sempre para dentro. As regras de negócio estão isoladas das implementações específicas de infraestrutura.

Repository Pattern: Utilizado para abstrair o acesso a dados, permitindo diferentes implementações de persistência (BoltDB para dados locais, potencial extensão para bancos remotos).

Dependency Injection: Implementado através de interfaces Go, facilitando testes unitários e flexibilidade na troca de implementações.

Strategy Pattern: Aplicado nos algoritmos de estimativa, permitindo diferentes abordagens de cálculo conforme o contexto.

3.4.4 Tecnologias e Frameworks

Linguagem Principal: Go 1.21+

- **Vantagens:** Performance superior, concorrência nativa com goroutines, facilidade de distribuição (binário único), excelente suporte para CLI
- **Frameworks:** Cobra (CLI), Viper (configuração), BoltDB (persistência)

Persistência Local:

- **BoltDB:** Banco de dados embarcado para dados históricos e estimativas
- **JSON/YAML:** Arquivos de configuração e cache estruturado
- **Vantagens:** Zero configuração, performance local, portabilidade

Integração Externa:

- **GitHub API v4:** Para coleta de dados de repositórios via GraphQL/REST
- **OpenAI API:** Para análise de linguagem natural e geração de estimativas
- **Rate Limiting:** Implementado para respeitar limites das APIs externas

3.4.5 Fluxo de Dados e Processamento

Estimativa Individual:

1. CLI recebe comando e parâmetros do usuário
2. Estimation Service processa a descrição da tarefa
3. Similarity Engine busca tarefas similares no histórico local
4. AI Client envia contexto para análise de IA
5. Resultado consolidado é retornado e formatado para exibição

Análise em Lote:

1. CLI carrega arquivo de tarefas (JSON/CSV)
2. Core Engine processa cada tarefa usando goroutines para paralelização
3. Resultados são agregados e estatísticas calculadas
4. Export Engine gera arquivos de saída nos formatos solicitados

Sincronização de Dados:

1. GitHub Client busca dados incrementais do repositório
2. Cache Manager otimiza requisições repetitivas
3. Repository persiste novos dados no BoltDB local
4. Similarity Engine reindexiza dados para buscas futuras

3.4.6 Estratégias de Performance

Concorrência: Utilização de goroutines para processamento paralelo de estimativas em lote e requisições de API.

Caching:

- Cache em memória para dados frequentemente acessados
- Cache de arquivos para resultados de APIs externas
- TTL configurável para balancear freshness e performance

Indexação: Índices otimizados no BoltDB para buscas rápidas de tarefas similares baseadas em keywords e metadados.

3.4.7 Segurança e Configuração

Autenticação: Tokens GitHub armazenados com criptografia AES-256 local.

Configuração: Sistema hierárquico de configuração (defaults < arquivo config < variáveis ambiente < parâmetros CLI).

Validação: Validação rigorosa de entrada para prevenir injection attacks e garantir integridade dos dados.

3.5 Diagramas de Estados

Os diagramas de máquina de estados descrevem os diferentes estados pelos quais as entidades principais do sistema passam durante sua execução.

3.5.1 Diagrama de Estados - Estimativa

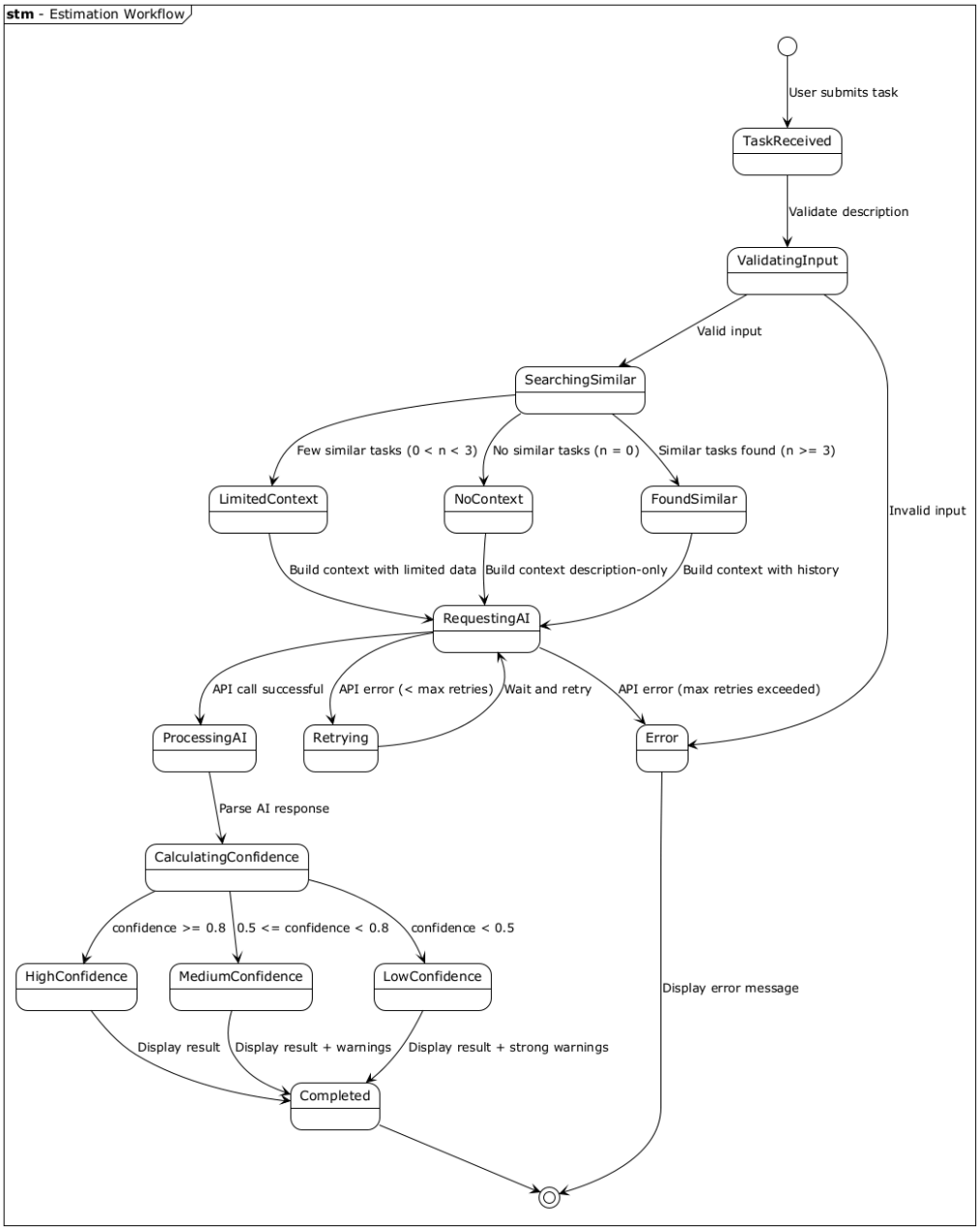


Figura 3.8 - Diagrama de Estados: Workflow de Estimativa

Este diagrama mostra os estados pelos quais uma solicitação de estimativa passa, desde a submissão até a conclusão ou erro.

3.6 Diagrama de Componentes e Implantação.

#TODO:Ainda trabalhando neles

4. Projeto de Interface com Usuário

Como o SET é uma ferramenta CLI (Command Line Interface), esta seção apresenta exemplos de interface de terminal para os principais comandos do sistema.

4.1 Interface CLI Comum a Todos os Atores

#TODO:Colocar print de cada comando

set estimate --task "Implementar login" --repo "user/project"
set analyze --project "user/project" --timeframe "last-6-months"
set configure --github-token "token" --repo "user/project"
set report --type estimation --output csv
set help

5. Glossário e Modelos de Dados

5.1 Glossário de Termos

Termo	Definição
CLI	Command Line Interface - Interface de linha de comando
IA	Inteligência Artificial
API	Application Programming Interface - Interface de Programação de Aplicações
Scrum	Framework ágil para desenvolvimento de software

GitHub	Plataforma de hospedagem de código e versionamento
Sprint	Período fixo de desenvolvimento no Scrum (geralmente 1-4 semanas)
Story Points	Unidade de medida para estimar o esforço relativo de tarefas
Velocity	Quantidade de story points completados por sprint
Planning Poker	Técnica de estimativa colaborativa usada em metodologias ágeis
Accuracy	Precisão das estimativas (quão próximas estão do tempo real)
Confidence Score	Nível de confiança da estimativa (0.0 a 1.0)
TF-IDF	Term Frequency-Inverse Document Frequency - algoritmo de processamento de texto
Cosine Similarity	Medida de similaridade entre vetores de texto
BoltDB	Banco de dados embarcado key-value em Go
GraphQL	Linguagem de consulta para APIs

Token	Chave de autenticação para APIs
Rate Limit	Limite de requisições por período de tempo

5.2 Modelos de Dados

5.2.1 Entidade: Task

Atributo	Tipo	Descrição
id	string	Identificador único da tarefa
title	string	Título descritivo da tarefa
description	string	Descrição detalhada da tarefa
labels	[]string	Lista de labels para categorização
estimated_hours	float64	Horas estimadas pela IA
actual_hours	float64	Horas realmente gastas (histórico)
story_points	int	Story points atribuídos
size	string	Tamanho da tarefa (XS, S, M, L, XL)
created_at	timestamp	Data de criação
completed_at	timestamp	Data de conclusão
developer	string	Desenvolvedor responsável
repository	string	Repositório de origem

5.2.2 Entidade: Estimation

Atributo	Tipo	Descrição
id	string	Identificador único da estimativa
task_id	string	Referência para a tarefa
estimated_hours	float64	Horas estimadas
story_points	int	Story points calculados
size	string	Classificação de tamanho
confidence	float64	Nível de confiança (0.0-1.0)
reasoning	string	Explicação da IA sobre a estimativa
similar_tasks	[]SimilarTask	Lista de tarefas similares encontradas
warnings	[]string	Avisos sobre limitações
created_at	timestamp	Data da estimativa
model	string	Modelo de IA usado (ex: gpt-4)

5.2.3 Entidade: SimilarTask

Atributo	Tipo	Descrição
task_id	string	ID da tarefa similar
title	string	Título da tarefa similar
similarity_score	float64	Score de similaridade (0.0-1.0)
estimated_hours	float64	Horas que foram estimadas
actual_hours	float64	Horas realmente gastas

accuracy_rate	float64	Taxa de acerto da estimativa original

5.2.4 Entidade: Issue (GitHub)

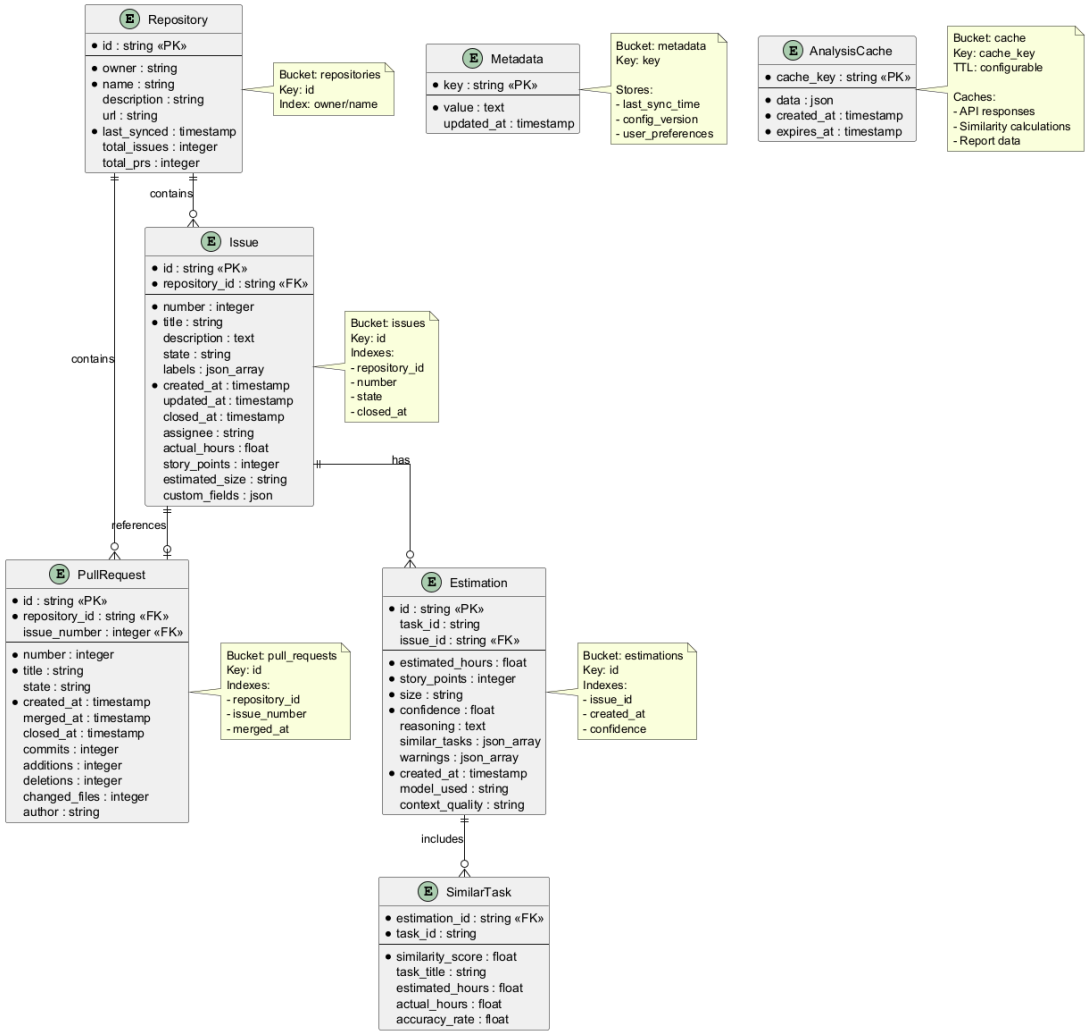
Atributo	Tipo	Descrição
id	string	ID único do GitHub
number	int	Número da issue no repositório
repository_id	string	Referência ao repositório
title	string	Título da issue
description	text	Corpo da issue
state	string	Estado (open, closed)
labels	json_array	Labels da issue
created_at	timestamp	Data de criação
closed_at	timestamp	Data de fechamento
actual_hours	float64	Horas gastas (de custom field)
story_points	int	Story points (de custom field)
estimated_size	string	Tamanho estimado (de custom field)
custom_fields	json	Outros campos customizados

5.2.5 Entidade: PullRequest

Atributo	Tipo	Descrição
id	string	ID único do GitHub

number	int	Número do PR no repositório
repository_id	string	Referência ao repositório
issue_number	int	Issue relacionada (se houver)
title	string	Título do PR
state	string	Estado (open, closed, merged)
created_at	timestamp	Data de criação
merged_at	timestamp	Data de merge
commits	int	Número de commits
additions	int	Linhas adicionadas
deletions	int	Linhas removidas
changed_files	int	Arquivos alterados
author	string	Autor do PR

5.3 Diagrama Entidade-Relacionamento



O diagrama acima mostra a estrutura do banco de dados BoltDB, organizada em buckets (equivalentes a tabelas) e os relacionamentos entre as entidades.

6. Casos de Teste

Esta seção apresenta os casos de teste elaborados para validar o sistema SET CLI. Na Seção 6.1, são detalhados os testes de aceitação, projetados para assegurar que as funcionalidades desenvolvidas atendam às necessidades dos usuários conforme especificado no documento de visão. A Seção 6.2 descreve os testes de integração, focando em verificar a interação entre o sistema e os componentes externos, como GitHub API e serviços de IA.

Os testes foram organizados para cobrir as quatro necessidades principais identificadas no documento de visão: precisão em estimativas de esforço, automatização do processo de estimativa, utilização de contexto histórico e integração com ferramentas existentes. Cada caso de teste inclui

pré-condições, dados de entrada, ações a serem executadas e resultados esperados, garantindo cobertura completa das funcionalidades críticas.

6.1 Testes de Aceitação

Os testes de aceitação (TA) foram elaborados com base nas necessidades identificadas no documento de visão e nos casos de uso apresentados. As necessidades testadas são:

- **N1**: Precisão em Estimativas de Esforço
- **N2**: Automatização do Processo de Estimativa
- **N3**: Utilização de Contexto Histórico
- **N4**: Integração com Ferramentas Existentes

Para a escrita dos testes, utiliza-se uma tabela com seis campos: Identificador (padrão TAXX), Necessidade, Caso de teste, Pré-condições, Dados de entrada, Ações e Resultado esperado.

6.1.1 Necessidade 1 – Precisão em Estimativas de Esforço

Esta necessidade garante que o sistema forneça estimativas precisas baseadas em dados históricos e análise de IA, reduzindo significativamente os desvios entre estimado e executado.

Identificador	TA01
Necessidade	Precisão em Estimativas de Esforço
Caso de teste	Gerar estimativa com alta confiança para tarefa similar
Pré-condições	• Sistema configurado com GitHub token válido • Base histórica contém pelo menos 5 tarefas similares • Usuário autenticado
Dados de entrada	Descrição da tarefa: "Implementar autenticação OAuth com Google"
Ações	1. Executar comando set estimate --task "Implementar autenticação OAuth com Google" 2. Sistema processa a descrição 3. Sistema busca tarefas similares 4. IA analisa e gera estimativa
Resultado esperado	• Estimativa gerada entre 4-12 horas • Confiança ≥ 80% • Pelo menos 3 tarefas similares encontradas • Tempo de resposta ≤ 30 segundos

Identificador	TA02
Necessidade	Precisão em Estimativas de Esforço
Caso de teste	Gerar estimativa com baixa confiança para tarefa inédita
Pré-condições	• Sistema configurado com GitHub token válido • Base histórica não contém tarefas similares
Dados de entrada	Descrição da tarefa: "Implementar blockchain para rastreamento"
Ações	1. Executar comando com tarefa inédita 2. Sistema não encontra similaridades 3. IA faz estimativa baseada apenas na descrição
Resultado esperado	• Estimativa gerada com confiança ≤ 50% • Aviso sobre limitação de dados históricos • Recomendação para revisar estimativa manualmente

ToDO: Add more cases

6.1.2 Necessidade 2 – Automatização do Processo de Estimativa

Esta necessidade visa reduzir o tempo gasto em reuniões de planning e permitir estimativas independentes da disponibilidade da equipe completa.

Identificador	TA03
Necessidade	Automatização do Processo de Estimativa
Caso de teste	Processar lote de tarefas para planning sprint
Pré-condições	• Sistema configurado • Arquivo JSON com 15 tarefas do backlog
Dados de entrada	Arquivo sprint-backlog.json contendo lista de tarefas
Ações	1. Executar set estimate --batch --file sprint-backlog.json 2. Sistema processa todas as tarefas em paralelo 3. Sistema gera relatório consolidado
Resultado esperado	• 15 estimativas geradas com sucesso • Tempo total de processamento ≤ 2 minutos • Relatório com total de horas estimadas • Arquivo CSV de saída gerado

ToDO: Add more cases

6.1.3 Necessidade 3 – Utilização de Contexto Histórico

Esta necessidade garante que dados valiosos de projetos anteriores sejam aproveitados para melhorar a precisão das estimativas futuras.

Identificador	TA04
Necessidade	Utilização de Contexto Histórico

Caso de teste	Analisar performance histórica da equipe
Pré-condições	• Base histórica com dados de 6 meses • Múltiplos desenvolvedores no histórico
Dados de entrada	Filtros: --team backend --period last-6-months
Ações	1. Executar set analyze --team backend --period last-6-months 2. Sistema processa dados históricos 3. Sistema calcula métricas de performance
Resultado esperado	• Accuracy média da equipe exibida • Identificação de padrões de super/subestimação • Ranking de desenvolvedores por precisão • Tendências temporais de melhoria

ToDO: Add more cases

6.1.4 Necessidade 4 – Integração com Ferramentas Existentes

Esta necessidade assegura que o sistema se integre naturalmente com GitHub e ferramentas Scrum sem causar disrupção no fluxo de trabalho existente.

Identificador	TA05
Necessidade	Integração com Ferramentas Existentes
Caso de teste	Sincronizar dados do GitHub automaticamente
Pré-condições	• Sistema configurado com GitHub • Repositório ativo com commits recentes
Dados de entrada	Comando de sincronização manual
Ações	1. Executar set sync --repository user/project --since last-week 2. Sistema acessa GitHub API 3. Sistema coleta issues, commits, PRs
Resultado esperado	• Dados sincronizados com sucesso • Issues fechadas adicionadas à base • Commits analisados para métricas • Rate limits respeitados

ToDO: Add more cases

7. Cronograma e Processo de Implementação

Esta seção apresenta o cronograma detalhado para implementação do sistema SET CLI e descreve o processo que será seguido durante o desenvolvimento. O cronograma foi estruturado considerando as datas marco estabelecidas e organizando as atividades em sprints quinzenais para facilitar o acompanhamento e entregas incrementais.

7.1 Cronograma de Implementação

Sprint	Período	Marco	Atividades Principais	Entregáveis
Sprint 1	16/09 - 29/09	**A2 - Boilerplate**	• **Setup do Projeto e Estrutura Inicial** • Configuração do repositório GitSetup do ambiente Go com módulos • Implementação da estrutura CLI com Cobra	• Projeto Go inicializado • Comandos CLI básicos funcionais • README com instruções de setup
Sprint 2	30/09 - 13/10		• **Sistema de Configuração e GitHub Integration** • Implementação do Configuration Service • Desenvolvimento do GitHub Client • Sistema de autenticação com tokens • Comandos de configuração inicial • Validação de conectividade	• Comando set configure funcional • Integração GitHub API implementada • Validação de tokens funcionando • Testes unitários das integrações • Documentação da configuração
Sprint 3	14/10 - 20/10	**A3 - Core Funcional**	• **Core de Estimativas e Persistência Local** • Implementação do EstimationService • Desenvolvimento do BoltRepository • Sistema de busca de similaridade • Comando básico de estimativa individualEstrutura de dados históricos	• Comando set estimate funcional • Persistência local operacional • Algoritmo de similaridade básico • Testes de integração com BoltDB • Documentação das APIs internas

Sprint 4	21/10 - 03/11		• **Integração com IA e Análise Avançada** • Implementação do AI Client (OpenAI) • Desenvolvimento do Similarity Engine • Integração IA + dados históricos • Sistema de confiança das estimativas • Tratamento de erros e fallbacks	• IA integrada às estimativas • Sistema de confiança funcional • Tratamento robusto de erros • Testes com mocks da API OpenAIMétricas de accuracy implementadas
Sprint 5	04/11 - 10/11	**A4 - Implementação Finalizada**	• **Funcionalidades Avançadas e Otimização** • Implementação do comando batch • Desenvolvimento do AnalysisService • Sistema de relatórios básico • Cache e otimização de performance • Comando de calibração	• Comando set estimate --batch funcional • Análise histórica implementada • Sistema de cache operacional • Relatórios básicos funcionando • Performance otimizada
Sprint 6	11/11 - 24/11		• **Relatórios Executivos e Export** • Implementação do ReportService completo • Sistema de exportação (CSV, PDF)Dashboards e visualizações • Comandos de análise avançada • Testes de integração completos	• Sistema completo de relatórios • Exportação funcionando • Todos os comandos implementados • Suite completa de testes • Cobertura de testes >85%
Sprint 7	18/11 - 24/11	**A5 - Revisão Final**	• **Testes Finais e Documentação** • Execução de todos os casos de teste • Correção de bugs identificados • Documentação técnica completa • Guias de usuário e instalação	• Todos os testes passando • Documentação completa • Guias de instalação e uso • Binários para múltiplas plataformas

				• Revisão de código finalizada
Sprint 8	25/11 - 30/11		• **Finalização e Entrega** • Testes finais de aceitação • Preparação da apresentação • Release final do sistema • Empacotamento para distribuição • Documentação de deployment	• Sistema 100% funcional • Release v1.0.0 publicada • Apresentação preparada • Documentação final entregue • Projeto pronto para uso