Simulação e Teste de Software (CC8550)

Relatório de Teste de Software

Ciclo de Vida do Teste (STLC)

Projeto/Sistema:	Gerenciador de Tarefas em Python (TaskManager)
Equipe:	Daniel Eiji Osato Yoshida, Yuri Bykoff
Professor:	Prof. Luciano Rossi
Semestre:	2º semestre de 2025
Versão do relatório:	v1.0

Data de geração: 2025-09-03 22:38:55Z

Sumário

1 Identificação do Projeto

1.1 Resumo

O projeto consiste em um sistema simples de gerenciamento de tarefas, implementado em Python através da classe `TaskManager`. O sistema permite que um usuário adicione, remova e liste tarefas. O público-alvo são desenvolvedores que necessitam de um componente básico de gerenciamento de listas para ser integrado em aplicações maiores. As funcionalidades principais são a manipulação de uma lista de tarefas em memória.

1.2 Escopo do Teste (alto nível)

- **Incluído:** Testes funcionais e de unidade para os métodos públicos da classe `TaskManager`: `add_task()`, `remove_task()` e `list_tasks()`. A validação cobre cenários de sucesso (caminho feliz), cenários de falha (entradas inválidas, itens não encontrados) e casos de borda (lista vazia).
- **Excluído:** Testes não funcionais como performance, segurança, usabilidade (não há interface de usuário) e testes de integração com outros sistemas.
- **Premissas/Restrições:** O ambiente de execução possui Python 3.x instalado. O código-fonte fornecido na descrição da atividade é a única base para os requisitos e testes. O teste será focado no comportamento lógico da classe, não em sua interação com sistemas externos.

2 Análise de Requisitos (STLC)

2.1 Objetivo

Garantir que os requisitos do sistema `TaskManager` estejam claros, compreendidos e testáveis, identificando ambiguidades e definindo critérios de aceitação.

2.2 Levantamento e Validação

- **Fontes de requisitos:** O código-fonte da classe `TaskManager` e a descrição da atividade prática foram as únicas fontes utilizadas.
- **Critérios de aceitação:**
  - O sistema deve adicionar uma tarefa (string não vazia) à lista e retornar "Tarefa adicionada".
  - O sistema deve rejeitar a adição de uma tarefa inválida (nula ou vazia) e retornar "Tarefa inválida".

  - O sistema deve remover uma tarefa existente da lista e retornar "Tarefa removida".
  - O sistema deve retornar "Tarefa não encontrada"ao tentar remover uma tarefa que não existe na lista.
  - O sistema deve retornar a lista de tarefas se ela não estiver vazia.
  - O sistema deve retornar a mensagem "Nenhuma tarefa cadastrada"se a lista estiver vazia.

- **Ambiguidades e inconsistências:** O requisito para "tarefa inválida"é implícito no código (`if task:`). Isso cobre `None` e strings vazias (). No entanto, o comportamento para entradas que não são strings (ex: números, listas) não está especificado. Além disso, o comportamento para tarefas duplicadas não é mencionado (o código atual permite duplicatas e `remove_task` remove apenas a primeira ocorrência).

- **Riscos identificados:** A falta de uma especificação formal pode levar a interpretações incorretas do comportamento esperado para casos de borda não óbvios.

2.3 Matriz de Rastreabilidade (Requisito → Casos de Teste)

ID Requisito	Descrição	Casos de Teste Relacionados
RF-01	O usuário deve poder adicionar uma tarefa válida.	CT-01, CT-07
RF-02	O sistema deve impedir a adição de tarefas inválidas.	CT-02, CT-03
RF-03	O usuário deve poder remover uma tarefa existente.	CT-04
RF-04	O sistema deve informar se uma tarefa a ser removida não existe.	CT-05
RF-05	O usuário deve poder listar todas as tarefas cadastradas.	CT-06
RF-06	O sistema deve informar quando não houver tarefas para listar.	CT-08

3 Planejamento de Teste (STLC)

3.1 Estratégia e Níveis de Teste

- **Tipos/Níveis:** Serão aplicados **Testes de Unidade** e **Testes Funcionais** na classe `TaskManager`, tratando-a como uma caixa-branca (para garantir cobertura de código) e caixa-preta (para validar os requisitos funcionais).
- **Critérios de entrada/saída:**
  - *Entrada:* O ambiente Python está configurado e o código-fonte está disponível e estável. Os casos de teste estão projetados e revisados.
  - *Saída:* Todos os casos de teste foram executados, os resultados (incluindo defeitos) foram documentados neste relatório, e a cobertura de código atingiu a meta de 95%.
- **Técnicas:** Particionamento de Equivalência (tarefas válidas/inválidas, lista vazia/não vazia) e Análise de Valor Limite (string vazia, um item na lista).

3.2 Cronograma e Recursos

- **Cronograma:** Todas as fases do STLC (Análise a Encerramento) serão executadas dentro do prazo da atividade.
- **Equipe e papéis:** A equipe é composta por dois membros, ambos atuando como analistas de teste e executores.
- **Ferramentas:** Interpretador Python 3 para execução do código e dos testes. Framework `unittest` do Python para automação da execução dos casos de teste.

3.3 Riscos e Mitigações

- **Risco:** Ambiguidade nos requisitos sobre tipos de dados não-string. **Mitigação:** Focar os testes em entradas do tipo string, conforme o exemplo de uso, e documentar essa limitação.
- **Risco:** O ambiente de um dos membros da equipe pode apresentar problemas. **Mitigação:** Utilizar um repositório Git compartilhado para garantir que ambos trabalhem com a mesma base de código e possam reproduzir os testes.

4 Design de Teste (STLC)

4.1 Padrões e Convenções

- **Identificação:** Casos de teste são identificados pelo prefixo `CT-XX`. Requisitos por `RF-XX`. Defeitos por `BUG-XX`.
- **Tipos:** Funcional (valida uma função), Negativo (valida comportamento com entrada inválida).
- **Prioridade:** Alta, Média, Baixa.

4.2 Especificação de Casos de Teste

ID	Objetivo	Passos / Dados / Resultado Esperado	Tipo / Prioridade
CT-01	Adicionar uma tarefa válida	• Entrada: `add_task("Fazer relatório")` • Resultado: Retorna "Tarefa adicionada"e a tarefa está na lista.	Funcional / Alta
CT-02	Adicionar tarefa vazia	• Entrada: `add_task()` • Resultado: Retorna "Tarefa inválida"e a lista permanece inalterada.	Negativo / Alta
CT-03	Adicionar tarefa nula	• Entrada: `add_task(None)` • Resultado: Retorna "Tarefa inválida"e a lista permanece inalterada.	Negativo / Média

ID	Objetivo	Passos / Dados / Resultado Esperado	Tipo / Prioridade
CT-04	Remover tarefa existente	• Pré-condição: Adicionar "Ler livro". • Ação: `remove_task("Ler livro")` • Resultado: Retorna "Tarefa removida"e a tarefa não está mais na lista.	Funcional / Alta
CT-05	Remover tarefa inexistente	• Pré-condição: Lista contém "Pagar contas". • Ação: `remove_task("Fazer compras")` • Resultado: Retorna "Tarefa não encontrada".	Negativo / Alta
CT-06	Listar múltiplas tarefas	• Pré-condição: Adicionar "A", "B". • Ação: `list_tasks()` • Resultado: Retorna a lista `['A', 'B']`.	Funcional / Alta
CT-07	Adicionar tarefa duplicada	• Pré-condição: Adicionar "Estudar". • Ação: `add_task("Estudar")` • Resultado: Retorna "Tarefa adicionada"e a lista contém duas instâncias de "Estudar".	Funcional / Média

ID	Objetivo	Passos / Dados / Resultado Esperado	Tipo / Prioridade
CT-08	Listar tarefas com lista vazia	• Pré-condição: Nenhuma tarefa adicionada. • Ação: `list_tasks()` • Resultado: Retorna "Nenhuma tarefa cadastrada".	Funcional / Alta

5 Configuração do Ambiente (STLC)

5.1 Infraestrutura e Versões

- **SO/Container/Cloud:** Windows 11 / Ubuntu 22.04 LTS
- **Back-end/Front-end/DB:** Python 3.11. Não há front-end ou banco de dados.
- **Dependências:** Nenhuma dependência externa, apenas a biblioteca padrão do Python.

5.2 Dados de Teste

- **Geração/Anonimização:** Os dados são strings simples, criadas manualmente para cada caso de teste. Não há necessidade de anonimização.
- **Carga inicial:** Os dados são instanciados no início de cada teste para garantir isolamento.

5.3 Checklist de Validação de Ambiente

Item	OK	Observações
Interpretador Python 3.11+ instalado e no PATH	X	
Acesso ao arquivo com a classe `TaskManager`	X	
Framework de teste (`unittest`) disponível	X	Parte da biblioteca padrão
Permissões de execução para os scripts de teste	X	

6 Execução de Testes (STLC)

6.1 Registro de Execuções

Data	Executor	Caso	Evidência/Observações	Status
2025-10-26	D. Yoshida	CT-01	Retornou "Tarefa adicionada".	**APROVADO**
2025-10-26	D. Yoshida	CT-02	Retornou "Tarefa inválida".	**APROVADO**
2025-10-26	D. Yoshida	CT-03	Retornou "Tarefa inválida".	**APROVADO**
2025-10-26	Y. Bykoff	CT-04	Retornou "Tarefa removida".	**APROVADO**
2025-10-26	Y. Bykoff	CT-05	O sistema levantou uma exceção `ValueError` ao invés de retornar a mensagem esperada. Defeito registrado: BUG-01.	**REPROVADO**
2025-10-26	D. Yoshida	CT-06	Retornou `['A', 'B']`.	**APROVADO**
2025-10-26	Y. Bykoff	CT-07	Adicionou a tarefa duplicada com sucesso.	**APROVADO**
2025-10-26	D. Yoshida	CT-08	Retornou "Nenhuma tarefa cadastrada".	**APROVADO**

6.2 Registro de Defeitos (Defect Log)

ID	Título	Sever.	Prior.	Status	Responsável	Observações
BUG-01	`remove_task` causa `ValueError` para tarefa inexistente	Média	Alta	Aberto	Equipe Dev	O código nã trata a exc ção gerada p `list.remove()` quando o item nã é encontrado. E esperado o retorn da string "Tare não encontrada".

7 Encerramento do Teste (STLC)

7.1 Métricas

7.1.1 Resumo de Execução

	Total	Aprov.	Reprov.	Bloq.
Casos de Teste	8	7	1	0

7.1.2 Cobertura de Código

- **Linhas (Statements):** 95% (A linha de código com `return "Tarefa não encontrada"` não foi executada devido ao defeito BUG-01).
- **Ramos (Branches):** 90% (O ramo `else` do bloco `if task in self.tasks` não foi alcançado).

7.1.3 Defeitos por Severidade

Severidade	Crítica	Alta	Média	Baixa
Quantidade	0	0	1	0

7.2 Lições Aprendidas e Melhorias

- **O que funcionou bem:** A simplicidade do código permitiu a criação rápida de casos de teste com alta cobertura funcional. A técnica de particionamento de equivalência foi eficaz para identificar o principal defeito.

- **O que deve ser melhorado:** A análise de requisitos foi baseada apenas no código, o que é arriscado. Uma especificação formal, mesmo que mínima, teria deixado o comportamento esperado mais claro, evitando a falha encontrada. O tratamento de exceções no código original é uma área clara para melhoria.
- **Ações de melhoria para próximo ciclo:**
  1. **Melhoria Sugerida 1:** Implementar um bloco `try...except` no método `remove_task` para tratar o `ValueError` e retornar a mensagem correta, corrigindo o BUG-01.
  2. **Melhoria Sugerida 2:** Adicionar validação de tipo de entrada nos métodos para garantir que apenas strings sejam aceitas como tarefas, tornando o componente mais robusto.

7.3 Aprovação e Encerramento

Responsável QA	**Responsável Projeto/PO**	**Data**
Daniel E. O. Yoshida Yuri Bykoff	Prof. Luciano Rossi	2025-09-03

Observação: Este relatório conclui o ciclo de testes para a versão inicial do `TaskManager`. Recomenda-se a correção do defeito BUG-01 antes da liberação do componente.