



UNISAGRADO

Ensino Superior de Excelência

CIÊNCIA DA
COMPUTAÇÃO

PROJETO PRÁTICO

TESTES E QUALIDADE DE SOFTWARE — 2025

Duração sugerida: 8 semanas (2 meses)

Grupos: até 4 pessoas

Entrega: pelo Connect+

1. Objetivo

O trabalho tem como meta fazer vocês **vivenciarem um ciclo completo de testes de software**, desde o planejamento até a execução e apresentação final.

Vocês irão:

- Escolher um **sistema simples** para testar (login, operação principal, mensagens de erro e sucesso, limites de dados).
- Planejar e criar **casos de teste**.
- **Executar os testes** (manuais e automatizados).
- **Registrar falhas encontradas**.
- Medir a **qualidade do sistema** com métricas.
- Mostrar como o uso de **TDD (Test-Driven Development) e CI/CD (Integração Contínua/Entrega Contínua)** pode ser aplicado.

O foco é aprender o **processo de testes** e produzir **evidências claras** (documentos, planilhas, prints, vídeos, relatórios).

2. Sistema a ser testado (SUT)

Vocês devem escolher um sistema **simples, mas completo** para aplicar os testes.

Exemplos:

- Sistema de **inscrição em eventos** acadêmicos.
- **Gerenciador de tarefas** (criar, editar, excluir, listar).
- **Catálogo de itens** (com busca e filtro, mas sem pagamento).

O sistema escolhido deve permitir:

- ✓ Testes funcionais (se faz o que deveria)
 - ✓ Testes de integração/sistema/aceitação
 - ✓ Pelo menos 1 teste **não funcional** (exemplo: usabilidade ou desempenho simples).
-

3. O que deve ser entregue (Fases e Pontos)

◆ Cada fase vale pontos que somam **10,0** (nota do Projeto Prático).

Fases obrigatórias:

1. Descoberta e requisitos testáveis (1,0 ponto)

- Definir **o que o sistema deve fazer**.
- Escrever requisitos com critérios claros de aceitação (ex.: *Dado/Quando/Então*).
- Identificar fluxos principais e possíveis riscos.

2. Plano de Teste e gestão (1,0 ponto)

- Definir objetivos, tipos de teste, papéis no grupo, ferramentas, datas e critérios de início/fim.

3. Matriz de Rastreabilidade (0,7 ponto)

- Criar planilha ligando requisitos ↔ casos de teste ↔ evidências/defeitos.

4. Casos de Teste (1,8 pontos)

- Criar casos de teste completos com: ID, objetivo, pré-condições, passos, dados, resultado esperado e técnica aplicada.
- Usar **classes de equivalência, valores-limite e tabelas de decisão**.
- Incluir pelo menos **1 caso E2E** (fluxo do início ao fim) e **1 não funcional**.

5. Dados e ambiente (0,8 ponto)

- Preparar massa de dados (usuários, tarefas, eventos etc.).

- Criar guia de instalação do sistema (README).

6. Execução manual e defeitos (1,6 pontos)

- Executar os casos em **2 ciclos** (primeira vez e regressão).
- Registrar o resultado (Passou/Falhou).
- Registrar falhas em Jira/GitHub Issues (com título, passos, esperado/obtido, severidade, prioridade e evidências).

7. Automação mínima (1,6 pontos)

- Automatizar pelo menos **3 testes**:
 - Login válido/inválido.
 - Fluxo principal completo (E2E).
 - API (com Postman/Newman ou equivalente).

8. TDD e CI/CD (0,8 ponto)

- Mostrar 1 exemplo de TDD (teste falhando → código → refatoração).
- Configurar pipeline de integração contínua (ex.: GitHub Actions).

9. Métricas e relatório final (0,6 ponto)

- Apresentar dados como: cobertura de requisitos, taxa de aprovação, defeitos encontrados, riscos e recomendações.

10. Apresentação final (0,1 ponto)

- Apresentação de até **10 minutos**, mostrando todo o processo e os resultados.

4. Linha do Tempo Sugerida (8 semanas)

- **Semana 1:** escolha do sistema + rascunho dos requisitos.
 - **Semana 2:** plano de teste + início da matriz de rastreabilidade.
 - **Semana 3:** elaboração dos casos de teste + ambiente de dados.
 - **Semana 4:** finalização dessa etapa e checkpoint 1.
 - **Semana 5:** execução manual (ciclo 1).
 - **Semana 6:** execução regressão + automação parcial.
 - **Semana 7:** automação final + TDD/CI/CD + métricas (checkpoint 2).
 - **Semana 8:** apresentação e entrega final.
-

5. O que será entregue

📁 Os artefatos devem estar organizados em:

- Plano de Teste (DOCX/PDF).
 - Casos de Teste + Matriz de Rastreabilidade (planilha).
 - Scripts de automação (em pasta/repositório com README).
 - Relatórios (execução, defeitos, métricas e final).
 - Evidências (prints/vídeos nomeados com ID).
-

6. Regras

- Trabalho em grupo de **2 a 4 pessoas**.
 - Entrega apenas pelo **Connect+**.
 - Atraso desconta **0,5 ponto por dia (até -2,0)**.
 - Plágio = **nota zero**.
 - Mudança de sistema após a 2^a semana: só com autorização do professor.
-

EXEMPLO RESOLVIDO — “BikeShare UNISAGRADO”

Projeto Prático — Testes e Qualidade de Software

A) Descoberta e requisitos testáveis (1,0)

Visão & escopo (resumo): Alunos logam, visualizam bikes disponíveis, retiram uma, devolvem e veem histórico.

Fluxos críticos: Login, listar bikes, emprestar (se disponível), devolver, mensagens de erro/sucesso, limites (0/1/N).

Riscos: Autenticação falhar; inconsistência de estoque; mensagens pouco claras; lentidão ao listar.

Requisitos com critérios de aceitação (formato BDD):

1. Autenticação

- Dado que sou usuário cadastrado, Quando informo e-mail válido e senha correta, Então devo acessar o sistema e ver meu nome no topo.
- Dado que não sou cadastrado ou a senha está errada, Quando tento logar, Então devo ver “Credenciais inválidas”.

2. Listagem de bicicletas

- Dado que há bicicletas cadastradas, Quando acesso “Bicicletas”, Então devo ver nome, status (Disponível/Emprestada) e código.

3. Empréstimo

- Dado que a bike X está **Disponível**, Quando clico em “Emprestar”, Então devo ver “Empréstimo confirmado” e a bike muda para **Emprestada** para mim.
- Dado que a bike Y está **Emprestada**, Quando tento emprestar Y, Então devo ver “Bicicleta indisponível no momento”.

4. Devolução

- Dado que possuo uma bike emprestada, Quando clico em “Devolver”, Então devo ver “Devolução registrada com sucesso” e a bike volta a **Disponível**.

5. Limites e validações

- Dado que tento emprestar com campos vazios, Então devo ver mensagens de validação (“Selecione uma bicicleta”).
- Dado que não há nenhuma bike disponível (0), Quando abro a lista, Então devo ver a lista vazia e a mensagem “Nenhuma bicicleta disponível”.

B) Plano de Teste e gestão (1,0)

Objetivos: Verificar autenticação, regras de empréstimo/devolução, mensagens, limites; validar usabilidade simples (clareza de mensagens) e tempo de resposta básico (<2s na listagem local).

Níveis: Unitário, Integração, Sistema/E2E, Aceitação.

Tipos: Funcional + **Não funcional** (usabilidade simples).

Estratégia: Técnicas de equivalência/limites/decisão; 2 ciclos (execução e regressão); registro de bugs em GitHub Issues; automação mínima (UI + API).

Papéis: PO/QA Líder, QA Designer (casos), Executor, Dev/Automação.

Calendário: 8 semanas (seguir cronograma do enunciado).

Critérios de entrada/saída: Ambiente configurado / 100% casos críticos executados e sem severidade Alta aberta.

Ambiente & ferramentas: App web local (Node + SQLite + HTML), Playwright, Postman/Newman, GitHub Issues, GitHub Actions.

C) Matriz de Rastreabilidade (0,7)

REQ	Caso(s)	Evidência(s) / Bug(s)
REQ-001 Login válido	CT-001,	VID-001 (login ok), BUG-002
	CT-002	(mensagem truncada)
REQ-003 Emprestar bike disponível	CT-010,	VID-010 (E2E), IMG-010 (status)
	CT-011	
REQ-005 Emprestar bike já emprestada	CT-012	BUG-004 (status não atualiza)
REQ-007 Devolver bike	CT-015	VID-015, IMG-015
REQ-009 Lista vazia (0 bikes)	CT-020	IMG-020 (mensagem “Nenhum...”)

(IDs batendo com planilha de casos e links para prints/vídeos/bugs.)

D) Projeto de Casos de Teste (1,8)

Técnicas aplicadas: Classes de equivalência (e-mail válido/inválido), valores-limite (0/1/N bikes), tabela de decisão (estado da bike × ação do usuário).

Exemplos (resumo):

- **CT-001 (Login válido) — Equivalência**
 - Pré: usuário aluno@uni.br / 123456
 - Passos: abrir login → informar credenciais válidas → entrar
 - Esperado: saudação com nome + redireciona para “Bicicletas”.
- **CT-002 (Login inválido) — Equivalência**
 - Dados: e-mail válido + senha inválida
 - Esperado: “Credenciais inválidas”.
- **CT-010 (Emprestar bike disponível) — Decisão**
 - Pré: bike B-21 Disponível
 - Passos: selecionar B-21 → “Emprestar”

- Esperado: “Empréstimo confirmado” + status muda para **Emprestada**.
- **CT-012 (Tentar emprestar bike emprestada)** — *Decisão*
 - Pré: B-07 **Emprestada** por outro usuário
 - Esperado: “Bicicleta indisponível no momento”.
- **CT-015 (Devolução)** — *Decisão*
 - Pré: usuário possui B-21 emprestada
 - Esperado: “Devolução registrada com sucesso” + status **Disponível**.
- **CT-020 (Lista com 0 bikes)** — *Limits (0/1/N)*
 - Pré: base sem bicicletas
 - Esperado: mensagem de lista vazia.
- **CT-030 (E2E)** — *E2E completo*
 - Fluxo: login válido → listar → emprestar bike disponível → verificar status → devolver → verificar status.
- **CT-040 (Não funcional — usabilidade)**
 - Critério: mensagens claras e visíveis em 3s; botões com rótulos compreensíveis.

Cada caso tem **ID, objetivo, pré, passos, dados, esperado, técnica, cobertura (REQ-xxx)**.

E) Dados e ambiente (0,8)

Stack (exemplo): Node/Express + SQLite; páginas HTML + fetch.

README (passo a passo): instalar dependências, npm run seed (cria usuários e bikes), npm start (porta 3000).

Massa de dados (seed):

- **Usuários:** aluno@uni.br/123456; visitante@uni.br/123456.
- **Bikes:** B-01 (Disponível), B-07 (Emprestada), B-21 (Disponível), “Ø” (cenário 0 bikes).

IDs estáveis (data-testid): data-testid="login-email", "btn-emprestar-B21", etc.

Script reset: npm run reset repovoa a base entre ciclos.

F) Execução manual e defeitos (1,6)

Ciclo 1 — Resultados (amostra):

- CT-001 Passou (VID-001)
- CT-002 Passou (IMG-002)
- CT-010 **Falhou** — status visual não mudou até recarregar a página (IMG-010-F)
- CT-012 Passou
- CT-015 Passou
- CT-020 Passou
- CT-030 **Falhou** — mensagem de confirmação corta no mobile (VID-030-F)
- CT-040 **Falhou** — contraste baixo no botão “Emprestar”

Bugs registrados (GitHub Issues):

- **BUG-004** — *Status da bike não atualiza imediatamente após empréstimo*
 - Versão/ambiente: v0.2-local / Chrome 126
 - Passos: Emprestar B-21 → observar status
 - Esperado: muda para **Emprestada** sem recarregar
 - Obtido: só muda após F5
 - Severidade: Média | Prioridade: Alta
 - Evidência: IMG-010-F, console log
- **BUG-006** — *Mensagem de confirmação truncada no mobile*
 - Severidade: Baixa | Prioridade: Média
 - Evidência: VID-030-F

(Demais campos: título, passos, esperado/observado, anexos.)

Ciclo 2 (Regressão): Reexecutar casos afetados por correções + E2E completo.

G) Automação mínima (UI e API) (1,6)

UI (Playwright) — 2 cenários:

1. Login válido e inválido

- npx playwright test ui/login.spec.ts
- Checks: saudação aparece; mensagem “Credenciais inválidas”.

2. Fluxo E2E principal (emprestar e devolver)

- Seleciona B-21, clica “Emprestar”, verifica status; clica “Devolver”, verifica status.

API (Postman/Newman) — 1 cenário:

- Coleção api-bikeshare.postman_collection.json:
 - POST /login (200/401)
 - GET /bikes (lista)
 - POST /loans (emprestar disponível → 201; emprestar emprestada → 409 com mensagem)
- Execução: newman run api-bikeshare.postman_collection.json -e local.postman_environment.json

(*README traz comandos de execução local.*)

H) TDD e CI/CD (0,8)

TDD — 1 história de usuário:

“Como usuário, quero ver o **tempo estimado** para identificação do status após emprestar, para entender se preciso aguardar.”

Teste (falha) → código → refatoração:

- **Teste unitário (Jest):** getStatusUpdateETA(ambiente='local') deve retornar <= 2s.
- Inicialmente falha (retornava 5s).
- Código ajustado para usar evento de confirmação do back-end em vez de polling.
- Refator: extrair função utilitária e cobrir erro de timeout.

CI/CD (GitHub Actions) — pipeline simples:

- Dispara em push/PR.
 - Passos: npm ci → lint → testes unitários → **Playwright** headless → **Newman** → publicar relatório de testes (artifacts).
 - Gate: falha qualquer teste → PR bloqueado.
-

I) Métricas e relatório final (0,6)

Cobertura por requisito (amostra):

- REQ-001, 003, 005, 007, 009: **100%** cobertos por ao menos um caso.

Taxa de aprovação por ciclo:

- Ciclo 1: 75% (6/8).
- Ciclo 2: 100% (8/8) após correções.

Densidade de defeitos (Ciclo 1):

- 2 bugs / 8 casos = 0,25 bug/caso (pequena).

Tempo de correção (simulado):

- BUG-004: 6h; BUG-006: 2h.

Riscos residuais & recomendações:

- Risco: atualização visual depende de evento assíncrono — cobrir com mais testes de integração.
 - Recomendação: testes de acessibilidade automatizados (axe) e testes de desempenho de lista.
-

J) Apresentação final (0,1)

Estrutura (10 min, todos falam):

1. **SUT & escopo (1 min)**
 2. **Plano & técnicas (2 min)**
 3. **Casos e evidências (3 min)** — prints/vídeos rápidos
 4. **Automação & CI (2 min)** — mostrar rodando local/prints do Actions
 5. **Métricas & lições (2 min)** — o que aprendemos, próximos passos
-

Organização dos artefatos (como entregar)

- **Plano_de_Teste_BikeShare.pdf**
- **Casos_de_Teste.xlsx** (com aba **Matriz_Rastreabilidade**)
- **/automacao** (Playwright, Postman, README com comandos)
- **Relatorio_Execucao.xlsx** (status + links de evidência)
- **Relatorio_Defeitos.csv** (export Issues)
- **Relatorio_Final_BikeShare.pdf**
- **/evidencias** (IMG-..., VID-... com ID no nome)

Este exemplo segue o **mesmo escopo, fases, pesos e padrões** do seu enunciado (A–J, checkpoints, automação mínima, TDD/CI, métricas, artefatos e regras), apenas em um **contexto diferente** para os alunos enxergarem **como fica “pronto”**

📍 **Guia de Nomenclatura de IDs**

Para manter **organização e rastreabilidade**, todos os artefatos devem ser identificados com **códigos padronizados**.

1. Casos de Teste

- **Formato:** CT-XXX
 - **Exemplo:** CT-001 = Caso de Teste 1 (Login válido)
 - **Uso:** Identificação única de cada caso de teste.
-

2. Evidências (prints e vídeos)

- **Formato para imagens:** IMG-XXX
 - Exemplo: IMG-010 = print do Caso de Teste 10.
 - **Formato para vídeos:** VID-XXX
 - Exemplo: VID-015 = vídeo da execução do Caso de Teste 15.
 - **Uso:** Cada evidência deve ter o mesmo número do caso de teste a que se refere.
-

3. Bugs / Defeitos

- **Formato:** BUG-XXX
 - **Exemplo:** BUG-004 = Defeito nº 4 identificado.
 - **Uso:** Cada defeito reportado em Jira/GitHub Issues ou planilha deve ter esse código.
-

4. Requisitos

- **Formato:** REQ-XXX
 - **Exemplo:** REQ-003 = Requisito nº 3 (Emprestar bike disponível).
 - **Uso:** Serve para rastrear se todos os requisitos foram testados.
-

5. Relatórios

- **Execução:** REL-EXEC-GrupoX.xlsx
 - **Defeitos:** REL-BUGS-GrupoX.xlsx ou export do GitHub/Jira.
 - **Final:** REL-FINAL-GrupoX.pdf
-

Exemplo de rastreabilidade

- **REQ-003** → validado pelo **CT-010**
 - Evidência: **IMG-010** e **VID-010**
 - Defeito registrado: **BUG-004**
-

 Assim, tudo “conversa”: requisito → caso → evidência → defeito.