UNIVERSIDADE FEDERAL DO AMAZONAS
INSTITUTO DE CIÊNCIAS EXATAS E TECNOLOGIA
CURSO DE ENGENHARIA DE SOFTWARE

MANUTENÇÃO E INTEGRAÇÃO DE SOFTWARE
SEMESTRE 2025-2

MANUTENÇÃO CORRETIVA

Instruções

- O trabalho deve ser feito preferencialmente **em grupo** com no mínimo **4 pessoas**.
- Todo o material deverá ser entregue diretamente no repositório GitHub do projeto.
- A organização das pastas e documentos será avaliada.

Descrição do Trabalho

Este trabalho tem como objetivo aplicar conceitos de manutenção corretiva de software no sistema que o grupo já desenvolveu, passando por todas as etapas modernas de manutenção: especificação, identificação de bugs, documentação em *bug reports*, rastreamento em sistema de *issues*, correção, testes e evidências.

Etapa 1 – Apresentação do Sistema

Antes de iniciar a manutenção corretiva, cada grupo deve criar um diretório no repositório chamado **/docs** contendo:

- **Descrição breve do sistema** (1 a 2 parágrafos explicando propósito e principais funcionalidades).
    - Exemplo: *O sistema X é uma aplicação web para gerenciamento de estoque de papelarias. Ele permite cadastrar produtos, controlar movimentações de entrada/saída e gerar relatórios de vendas.*
- **Arquitetura resumida**: breve explicação da estrutura (pode ser um diagrama simples de classes, containers ou módulos).
- **Como executar o sistema**: instruções claras para rodar localmente (dependências, comandos, etc.)

Etapa 2 – Identificação e Classificação de Bugs

O grupo deve revisar o próprio código e encontrar **pelo menos 3 bugs reais** (se não houver, devem simular plausivelmente).

- Cada bug deve ser classificado em uma das categorias:
    - **Sintaxe** (ex.: parêntese faltando, variável não declarada).
    - **Lógica** (ex.: condição incorreta, cálculo errado).
    - **Runtime** (ex.: erro ao acessar índice inexistente).
    - **Concorrência** (ex.: race condition em acesso a recurso compartilhado).
    - **Performance** (ex.: laço ineficiente que trava em grandes entradas).
    - **Segurança** (ex.: SQL Injection, falta de validação de entrada).

Exemplo de bug encontrado:

- Tipo: **Lógico**
- Local: Carrinho.java, método getTotal()
- Descrição: ao calcular o valor do carrinho, o desconto é subtraído duas vezes, resultando em valor incorreto.

Etapa 3 – Registro dos Bugs (Bug Reports)

Cada bug identificado deve ser registrado como uma **Issue no GitHub** do projeto, seguindo um **template padronizado**.

Template mínimo sugerido:

- **Título:** Resumo curto (ex.: "Erro de cálculo no carrinho de compras").
- **Descrição:** Breve explicação do problema.
- **Passos para reproduzir:** Lista de ações que geram o erro.
- **Resultado obtido:** O que realmente acontece.
- **Resultado esperado:** O que deveria acontecer.
- **Evidência:** Logs, prints ou trechos de código.
- **Classificação:** Tipo de bug (lógico, runtime, etc.).

Etapa 4 – Rastreamento em Bug Tracking System

Os bugs registrados devem ser **organizados no GitHub Issues** com:

- Labels (ex.: bug, critical, low-priority).
- Responsável atribuído (quem do grupo vai corrigir).
- Status atualizado (Open - In Progress - Closed).

Exemplo prático:

- Issue #3: *Erro de cálculo no carrinho*
    - Label: bug, high-priority
    - Responsável: João
    - Status: *In Progress*

Etapa 5 – Correção dos Bugs

Depois que os bugs forem identificados e registrados como issues no GitHub, cada grupo deve corrigir os bugs seguindo boas práticas de versionamento e colaboração.

1. Criar uma branch para cada bug

- Cada correção deve ser feita em **uma branch separada** (não no main).

- O nome da branch deve indicar o bug que será corrigido.
    - Exemplo:
    - git checkout -b fix/bug-calculo-carrinho

---

2. Fazer a correção do bug

- Alterar o código necessário para corrigir o bug.
- Rodar testes (se existirem) para confirmar que o erro foi corrigido.

👉 Aqui entra a parte de **manutenção corretiva** na prática.

---

3. Comitar a correção com mensagem clara

- Usar **commits atômicos** (um commit = uma mudança lógica).
- Seguir a convenção **Conventional Commits**:

Exemplos:

fix: corrige cálculo duplicado no método getTotal() do Carrinho

fix: trata erro de índice fora da lista em Pedido.java

👉 Assim, a mensagem já deixa claro que o commit é uma **correção de bug**.

---

4. Relacionar a issue no commit

- Ao comitar, a equipe deve mencionar a issue correspondente usando a palavra-chave fixes ou closes.
- Exemplo:

fix: corrige cálculo duplicado no método getTotal() do Carrinho (fixes #3)

👉 Isso faz com que, quando o PR for aceito, a **issue seja automaticamente fechada pelo GitHub**.

---

5. Abrir um Pull Request (PR)

- Após corrigir o bug na branch, a equipe deve **abrir um PR** para a branch main.
- O título do PR deve resumir a correção:
    - Ex.: fix: corrigir cálculo no carrinho de compras (issue #3)
- A descrição do PR deve conter:
    - Resumo da correção.
    - Link para a issue correspondente.
    - Evidências (prints, logs, descrição do teste que valida a correção).

👉 Isso simula como times reais discutem e revisam código antes de integrar na branch principal.

---

6. Revisão do PR e merge

- Dentro do grupo, um colega pode revisar o PR (fazer comentários, sugerir ajustes).
- Depois de aprovado, o PR deve ser **mergeado** na main.
- A issue correspondente será automaticamente fechada (se usada a palavra-chave fixes).

Etapa 6 – Evidência de Validação

Para cada bug corrigido, deve haver evidência de validação:

- **Teste automatizado novo** cobrindo o caso corrigido, ou
- **Relatório manual de teste** (descrição dos passos e resultado correto).

Exemplo de evidência:

- Bug #3 corrigido:
    - Adicionado teste unitário CarrinhoTest.testCalculoComDesconto() que confirma cálculo correto.

Etapa 7 – Relatório Final

- Breve descrição do sistema.
- Lista dos bugs identificados e suas classificações.
- Links para as issues correspondentes no GitHub.
- Links para os commits/PRs que corrigem cada bug.
- Evidências dos testes de validação

Critérios de Avaliação

1. Clareza na apresentação do sistema – 1,0
2. Identificação e classificação correta de bugs – 2,0
3. Qualidade dos bug reports (issues bem escritos) – 2,0
4. Uso correto do GitHub Issues (labels, status, responsáveis) – 1,5
5. Qualidade das correções (commits claros, boas práticas) – 2,0
6. Evidências de validação (testes ou relatórios) – 1,5

Total: 10 pontos.