

Projeto Interdisciplinar – Comedoria da Tia

Relatório de Qualidade e Testes de Software

1. Introdução

O presente relatório apresenta a aplicação dos conceitos de Qualidade de Software e Testes de Software no contexto do projeto Aplicativo Mobile – Comedoria da Tia, desenvolvido no âmbito do Projeto Interdisciplinar do curso de Análise e Desenvolvimento de Sistemas.

O sistema tem como objetivo otimizar o atendimento da cantina da FECAP, permitindo que os alunos realizem pedidos e pagamentos antecipadamente, reduzindo filas e melhorando a experiência de consumo. Além disso, a cantina contará com uma interface administrativa para gerenciar o cardápio, pedidos e relatórios.

2. Qualidade de Software

2.1. Modelo de qualidade de software

Clique [aqui](#) para baixar a imagem do diagrama ou baixe pelo github.

2.2. Processo (plano) de gerenciamento de qualidade de software

O processo de qualidade será conduzido considerando as seguintes práticas:

- Análise de requisitos para garantir cobertura das necessidades dos alunos e da cantina.
- Boas práticas de codificação (padrões de projeto e modularidade).
- Testes contínuos (unitários, integração, usuário e carga).
- Validação de usabilidade com protótipos e feedback de alunos.
- Documentação técnica e relatórios de testes.

2.3. Identificação de atributos de qualidade da norma 25010

Os atributos da ISO/IEC 25010 aplicáveis ao projeto são:

- Usabilidade → interface intuitiva e responsiva.
- Confiabilidade → o sistema deve funcionar sem falhas durante os pedidos.
- Eficiência de desempenho → resposta rápida em consultas ao cardápio e finalização de pedidos.

- Segurança → proteção dos dados de login e transações financeiras.
- Manutenibilidade → código modular e testável.
- Portabilidade → aplicação inicial para Android, com possibilidade de expansão para iOS.

2.4. Relatório sobre a aplicação da ISO/IEC 25010

A norma ISO/IEC 25010 foi utilizada como referência para definir métricas e critérios de qualidade no desenvolvimento do aplicativo.

A usabilidade será garantida por meio de prototipação, testes com usuários e adoção de boas práticas de UX.

A segurança será atendida pela integração com APIs de pagamento certificadas.

A eficiência de desempenho será validada com testes de carga.

A confiabilidade será garantida por testes unitários e de integração.

A manutenibilidade será garantida por código modular, reutilizável e com comentários.

3. Teste de Software

3.1. Plano de Teste

Escopo: validar cadastro, login, pedidos, pagamentos e histórico.

Ferramentas: JUnit (Java), JMeter (carga), Android Emulator e Postman.

Método: testes unitários → integração → usuário → carga.

Critério de aceitação: todas as funcionalidades devem estar implementadas e operacionais sem falhas críticas.

3.2. Testes Unitários

3.2.1. Teste Unitário 1

```
import org.junit.Test;
import static org.junit.Assert.*;

public class SistemaTestesUnitarios {

    // 1. Teste unitário de login correto
    @Test
    public void testLoginCorreto() {
        String usuario = "aluno";
        String senha = "123";
        assertTrue(usuario.equals("aluno") && senha.equals("123"));
    }
}
```

3.2.2. Teste Unitário 2

```
// 2. Teste unitário de login incorreto
@Test
public void testLoginIncorreto() {
    String usuario = "aluno";
    String senha = "errada";
    assertFalse(usuario.equals("aluno") && senha.equals("123"));
}
```

3.2.3. Teste Unitário 3

```
// 3. Teste unitário de cálculo de pedido
@Test
public void testCalculoPedido() {
    double item1 = 5.0;
    double item2 = 7.5;
    double totalEsperado = 12.5;
    double totalCalculado = item1 + item2;
    assertEquals(totalEsperado, totalCalculado, 0.01);
}
```

3.2.4. Teste Unitário 4

```
// 4. Teste unitário de histórico de pedidos
@Test
public void testHistoricoPedidos() {
    String[] historico = {"Coxinha", "Suco"};
    assertEquals(2, historico.length);
    assertEquals("Coxinha", historico[0]);
}
}
```



Resultado esperado: todos ficam **verdes** no JUnit. Clique [aqui](#) para tutorial.

3.3. Testes de Integração

3.3.1. Teste de integração 1

```
public class SistemaTestesIntegracao {

    // 1. Integração login + cardápio
    @Test
    public void testFluxoLoginCardapio() {
        boolean login = true; // simula login válido
        String cardapio = login ? "Lista de itens" : null;
        assertNotNull(cardapio);
    }
}
```

3.3.2. Teste de integração 2

```
// 2. Integração pedido + pagamento
@Test
public void testFluxoPedidoPagamento() {
    boolean pedidoCriado = true;
    boolean pagamentoEfetuado = true;
    String status = (pedidoCriado && pagamentoEfetuado) ? "Pago" : "Falha";
    assertEquals("Pago", status);
}
}
```

Resultado esperado: fluxo completo validado/aprovado.

3.4. Teste de Usuário

Cenário: O aluno abre o aplicativo, insere login e senha válidos, visualiza o cardápio, escolhe “Coxinha” e “Suco”, confirma o pedido e realiza o pagamento.

Resultado esperado: o sistema retorna “Pedido confirmado – retirar no balcão em 5 minutos”.

Critério de aceitação: fluxo concluído em até 30 segundos, sem erros.

3.5. Teste de carga

A seguir, uma representação do teste de carga.

Usuários Simultâneos	Tempo médio de resposta (ms)	Taxa de erro (%)
100	120	0
250	150	0
500	200	2
750	280	3
1000	400	5

4. ATENÇÃO

Realizamos um teste prático de nosso sistema. Você pode assistir baixando o vídeo pelo GitHub ou clicando [aqui](#).

5. Conclusão (Observação Final)

Os testes apresentados neste relatório têm caráter **simulatório**, uma vez que o sistema ainda não se encontra em estágio de implementação completo. Foram utilizados cenários hipotéticos, códigos **mock** em **JUnit** e dados fictícios para representar o funcionamento esperado das funcionalidades principais. Dessa forma, o objetivo foi **demonstrar o entendimento conceitual** sobre qualidade de software, planejamento de testes e aplicação prática das normas, garantindo que, em etapas futuras de desenvolvimento, estes testes possam ser executados de forma real no sistema. **No entanto**, como mencionado anteriormente, você pode

assistir ao teste prático pelo vídeo disponível no YouTube ou no GitHub, conforme dito anteriormente.

6. Referências Bibliográficas

SOMMERVILLE, I. Engenharia de Software. 11^a Edição. São Paulo: Pearson Addison-Wesley, 2017.

ISO/IEC 25010:2011. Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models.