Somativa

Avaliação Somativa – Testes de Software

Disciplina: Testes de Software

Tipo: Avaliação Somativa (Consulta APENAS do MATERIAL)

Valor: 100 pontos

PARTE I – QUESTÕES CONCEITUAIS DISSERTATIVAS (40 pontos)

Questão 1 (10 pontos)

Explique como os princípios **"Testar cedo economiza tempo e dinheiro"** e **"Agrupamento de Defeitos (Efeito Pareto)"** podem ser combinados de forma estratégica para otimizar o planejamento de testes de um projeto com um cronograma apertado.

Questão 2 (15 pontos)

Imagine que você é o QA de um novo aplicativo de e-commerce. Usando a metodologia de **Análise de Risco**, identifique um **Risco de Produto** e um **Risco de Tecnologia** para este projeto.

- Descreva cada risco no formato: *"Se <evento> acontecer, <consequência>"*
- Justifique por que esses riscos seriam prioritários.

Questão 3 (15 pontos)

Compare as técnicas de **Caixa Branca** e **Caixa Preta**, detalhando o tipo de defeito que cada uma é mais eficaz em encontrar.

Explique por que o **Teste Unitário** é tipicamente associado à Caixa Branca, enquanto o **Teste de Aceitação** está mais associado à Caixa Preta.

PARTE II – QUESTÕES DE MÚLTIPLA ESCOLHA COM ANÁLISE DE CÓDIGO (60 pontos)

Questão 4 (10 pontos)

Os testes de Sistema e de Aceitação ocorrem em fases avançadas do projeto. Qual alternativa descreve a principal diferença de foco entre eles?

a) O Teste de Sistema é sempre manual, enquanto o de Aceitação é sempre automatizado.

b) O Teste de Sistema é executado pelo desenvolvedor para validar a lógica interna, e o de Aceitação pelo cliente para validar a interface.

c) O Teste de Sistema foca em verificar se o software atende aos requisitos funcionais e não funcionais especificados, enquanto o de Aceitação foca em garantir que o sistema atende às necessidades e expectativas do negócio e do usuário final.

d) O Teste de Sistema valida o fluxo completo (end-to-end), e o de Aceitação valida apenas módulos isolados.

Questão 5 (10 pontos)

Analise o método de teste abaixo:

```
public function test() {
    $calc = new Calculator();
    $this->assertEquals(5, $calc->media([2, 2, 2]));
    $this->assertEquals(10, $calc->media([3]));
}
```

Qual é o **principal problema estrutural** neste teste, de acordo com as boas práticas de testes unitários?

a) O teste não segue a estrutura Arrange-Act-Assert (AAA) e a função `assertEquals` é inadequada para comparar valores numéricos.

b) O teste não cobre casos de erro e a classe `Calculator` deveria ser instanciada no método `tearDown()` para preparar o ambiente.

c) O teste falha em ser independente, pois o resultado do segundo `assert` depende diretamente do sucesso do primeiro.

d) **O nome do método é genérico e ele testa múltiplos cenários em um único método, violando a prática de "um teste por funcionalidade"**

Questão 6 (10 pontos)

Uma empresa está migrando seu sistema de um banco de dados MySQL para PostgreSQL. A equipe precisa garantir que, após a migração, as consultas retornam exatamente os mesmos resultados que retornavam no sistema antigo. Qual técnica de teste é a mais indicada para essa validação?

a) Teste de Recuperação.

b) Teste de Estresse.

c) Teste de Usabilidade.

d) Teste de Paralelo.

Questão 7 (10 pontos)

Um risco de "vulnerabilidade de segurança em API de pagamento" tem baixa probabilidade de ocorrer, mas um impacto muito alto (danos à reputação e perda de receita). Na Matriz de Risco, este cenário exigiria:

a) Atenção prioritária, pois o impacto catastrófico justifica o investimento em testes, mesmo com baixa probabilidade.

b) Baixa prioridade, pois a probabilidade de ocorrência é pequena.

c) Monitoramento ocasional, pois o risco é considerado moderado.

d) Aceitação do risco sem ação, pois os custos de mitigação seriam muito altos.

Questão 8 (10 pontos)

Qual é a principal justificativa técnica para utilizar o método **setUp()** para inicializar objetos, em vez de criá-los diretamente no construtor da classe de teste ou no início de cada método?

a) Melhora a performance geral da suíte de testes, pois os objetos são criados apenas uma vez.

b) É a única maneira de passar dependências para a classe de teste.

c) Permite que todos os testes compartilhem a mesma instância de um objeto, economizando memória.

d) Garante a independência dos testes, assegurando que cada método de teste seja executado com um estado limpo e uma nova instância do objeto, evitando que o resultado de um teste interfira no outro.

Questão 9 (10 pontos)

O Teste de Regressão é fundamental para a manutenção da qualidade. Em qual cenário sua automação e inclusão em uma pipeline de CI/CD traz o maior benefício?

a) Apenas uma vez por mês, para validar o estado geral do sistema.

b) Somente antes de grandes lançamentos para produção, como uma verificação final.

c) De forma contínua, sendo executado automaticamente após cada novo commit ou merge de código, para detectar quebras de funcionalidade o mais cedo possível.

d) Apenas no ambiente de desenvolvimento local do programador, antes de ele enviar o código para o repositório.

Questão 10 (10 pontos)

Observe o método **dividir()** e seu teste:

```
// Código da Classe Calculator
public function dividir($a, $b) {
    if ($b == 0) {
        return null; // Retorna nulo em caso de divisão por zero
    }
    return $a / $b;
}

// Código do Teste
public function testDivisaoComNumerosValidos() {
    $calc = new Calculator();
    $resultado = $calc->dividir(10, 2);
    $this->assertEquals(5, $resultado);
```

```
}
```

Qual cenário crítico, além do "happy path" já testado, é essencial para garantir a robustez deste método, de acordo com as boas práticas?

a) Testar a divisão de dois números negativos.

b) Testar o caso de divisão por zero, verificando se o método retorna null conforme a regra de negócio (`assertNull`).

c) Testar a divisão de um número por ele mesmo.

d) Testar a performance do método com números muito grandes.