Curso: Sistemas de Informação – Unidade São Gabriel
Disciplina: Fundamentos de Testes de Software – Período: 5º - Turno: Noite
Professor: Claudiney Vander Ramos
Data de Entrega: 02/06/2020
Lista 2 - Teste de Software Orientado por Objetos; Teste de Aplicações Web; *Test Driven Development* (TDD); Refatoração (*refactoring*).

1) O que são *drivers* (pseudo-controladores) e *stubs* (pseudo-controlados)?
2) Explique o que são testes de integração e as diferenças entre as metodologias *top-down* e *bottom-up*.
3) Explique, através de um pequeno exemplo, como o teste de programas orientados a objetos pode ser diferente do teste de programas estruturados.
4) Descreva com suas próprias palavras por que a classe é a menor unidade razoável de teste de um sistema OO.
5) Por que temos de testar novamente subclasses instanciadas a partir de uma classe existente, se a classe existente já foi testada rigorosamente? Podemos usar os casos de teste projetados para a classe existente?
6) Explique resumidamente o propósito/objetivo de cada uma das seções em um plano de testes de software.
7) Uma especificação de um programa declara que ele aceita de 2 a 6 entradas, que são números inteiros de quatro dígitos, maiores que 5000. Defina as partições de equivalência necessárias para os testes de caixa-preta para este programa.
8) Discutir as principais características das aplicações Web que as tornam diferentes das aplicações convencionais.
9) Quais as diferenças entre os testes de aplicações Web e os testes de aplicações convencionais?
10) Compatibilidade é uma importante dimensão de qualidade. O que precisa ser testado para garantir que exista compatibilidade em uma aplicação Web?
11) Que erros tendem a ser mais sérios – erros do lado do cliente ou erros do lado do servidor? Por quê?
12) Que elementos de aplicações Web podem ser submetidos ao teste de unidade? Que tipos de testes devem ser conduzidos apenas depois que os elementos da aplicação Web tenham sido integrados?
13) Quais os tipos de testes possíveis de serem aplicados a um software que está sendo desenvolvido? Quais critérios devem ser observados para definir os tipos de testes que serão aplicados?
14) Descreva a sequência de atividades de um ciclo de TDD (*Test Driven Development*).
15) Explique como os objetos *mock* podem ser utilizados no contexto do TDD.

16) Explique o significado do termo refatoração (*refactoring*). Mostre exemplos de situações onde é necessário o uso da refatoração de código. Descreva os principais benefícios da refatoração de código.

17) Considere um programa que recebe como entrada um valor inteiro referente ao mês de nascimento de uma pessoa. Considerando a técnica de teste de caixa-preta, elabore um conjunto de casos de teste informando os valores que seriam utilizados como entrada, utilizando:

a) Partição em classes de equivalência.

b) Análise de valor limite.

18) Considere a implementação dos algoritmos apresentadas em **i)** e **ii)**:

i)

```
public int calculaMaior (int [] vetor) {
  int i = 0;
  int maior;
  if ((v[0]==v[1]) && (v[1]==v[2]))
    Console.Write ("Todos os elementos são iguais");
  else
  {
    maior = v[0];
    for (i = 1; i < vetor.length; i++)
    {
      if (v[i] > maior)
        maior = v[i];
    }
    Console.Write ("Maior elemento: " + maior);
  }
}
```

ii)

```
public int verificaNota (Aluno [] aluno) {
  int i = 0; int j = 0; double media = 0;
  while (j < Aluno.length)
  {
    for (i = 0; i < aluno.nota.length; i++)
      media += aluno.nota[i];
    media = media / aluno.nota.length;
    if (media >= 6)
      Console.Write ("Aluno aprovado");
    else if (media >= 4)
      Console.Write ("Exame especial");
    else
      Console.Write ("Aluno reprovado");
    j++;
  }
}
```

Para cada algoritmo, faça:

a) Construa um grafo de fluxo de controle para o algoritmo.

b) Calcule a complexidade ciclomática do grafo construído.

c) Elabore um conjunto de casos de teste que garanta a cobertura de comandos/instruções para o algoritmo.
d) Elabore um conjunto de casos de teste que garanta a cobertura de decisões para o algoritmo.

19) Observe a classe abaixo que calcula o consumo de um aparelho em reais. Crie um caso de teste (código) para esta classe. Utilizar objetos *mock*.

```
public class Consumo {
  public double getPotencia (Medidor medidor, Aparelho aparelho)
  {
        double potenciaAparelho = aparelho.getPotencia();
        double precoEnergia = 0,30;
        double tempoFuncionamentoEmHoras;

        tempoFuncionamentoEmHoras =
                    medidor.getTempoFuncionamentoEmHoras();

        double valorEmkWh = potenciaAparelho / 1000;

        //Cálculo do consumo
        return valorEmkWh * precoEnergia;
  }
}
```

20) Analise cada código abaixo, identifique e aplique a refatoração adequada:

a.

```
class Empregado {

      int id;
      string nome
      double salario;

      public Empregado (int id)
      {
            this.id = id;
      }

      public void setNome (string nome)
      {
            this.nome = nome;
      }

      public double getSalario ()
      {
            return salario;
      }

}

class Vendedor extends Empregado {

      string setor;
```

```
        public string getSetor ()
        {
                return this.setor;
        }
}
```

b.

```
public void imprimeValorParcela () {

        double preco = item.getPreco ();
        double juros = 0.1;
        double precoFinal = 0.0;

        // calcula parcela
        precoFinal = calculaParcela(preco, juros);

        // imprime resultados
        System.out.println ("Preço Final: " + precoFinal);

}

double calculaParcela (double a, double b)
    { return a + (a * b); }
```

c.

```
class Conta {
        double getAnuidade () {
                switch (_type) {
                        case POUPANCA:
                                return 0.0;
                        case CORRENTE:
                                return 15.0;
                        case ESPECIAL:
                                return 30.0;
                }
                throw new RunTimeException (“Tipo desconhecido!”);
        }
}
```