Boas práticas de desenvolvimento de software

Material enriquecido também pela disciplina Fundamentos de Engenharia de Software da Prof.ª Maria Augusta Vieira Nelson

Dicas, sugestões, métodos e técnicas

C / C++

UML

Documentação

- Código Fonte
- Requisitos
- Arquitetura
- Para o Usuário

Motivação para documentação do código-fonte

A documentação do código-fonte não pode ser uma alternativa a um descaso para escolha de identificadores (variáveis, constantes, funções, *etc.*), como demonstrado nos exemplos abaixo:

```
int i; // representa a idade

 // verifica se está o aluno aprovado ou não a partir da nota parametrizada
bool verifica(double n)
{
        return n>= 60;
}
```

Motivação para documentação do código-fonte

Ao contrário, a prioridade deve ser buscar alto grau de clareza com boas escolhas para os identificadores (variáveis, constantes, funções, *etc.*)

```
int idade;

 // verifica se está o aluno aprovado ou não a partir da nota
bool aprovado(double nota)
{
        return nota>= 60;
}
```

Motivação para documentação do código-fonte

Comentários devem ser inseridos:

- para as partes do código que aparentemente não estão claras, como no exemplo abaixo:

```
int fatorial(int n) {
        int resultado;

        // Caso base: o fatorial de 0 é definido como 1
        if (n == 0) {
                resultado = 1;
        }
        else { // Caso indutivo: n! = n * (n-1)!
                resultado = n * fatorial(n - 1);
        } // fim if()
        return resultado;
} // fim fatorial()
```

Uma correlação do código-fonte com modelagem da solução

$$
\text{fatorial}(n) = \begin{cases} 0, \text{ para } n<0 \text{ (não definido)} \\ 1, \text{ para } n=0 \text{ (caso base)} \\ \\ n * \text{fatorial}(n-1), \text{ para } n>0 \text{ (caso indutivo)} \end{cases}
$$

```
int fatorial(int n) {
        int resultado=0; // Caso fatorial não definido

        // Caso base: o fatorial de 0 é definido como 1
        if (n == 0) {
                resultado = 1;
        }
        else { // Caso indutivo: n! = n * (n-1)!
                resultado = n * fatorial(n - 1);
        } // fim if()
        return resultado;
} // fim fatorial()
```

Motivação para documentação do código-fonte

Comentários devem ser inseridos:

- quando se deseja acelerar a compreensão de um trecho, por exemplo, explicando o laço que iniciará logo à frente.

```
// Listar as notas acima da média
printf("Notas acima da média:\n");
for (int i = 0; i < numEstudantes; i++)
{               if (notas[i] > media) {
                        printf("Nota do estudante %d: %.2f\n", i + 1, notas[i]);
                }
}
```

Documentação em arquivo (biblioteca, classe, estrutura, *etc.*)

Busque documentar os propósitos de cada arquivo de código-fonte

Gere um cabeçalho de forma que todo arquivo inicie com um conjunto de comentários que descrevam seu objetivo e conteúdo

```
/*H*****************************************************************
* FILENAME :        fmcompres.c              DESIGN REF: FMCM00
*
* DESCRIPTION :
*       File compression and decompression routines.
*
* PUBLIC FUNCTIONS :
*       int     FM_CompressFile( FileHandle )
*       int     FM_DecompressFile( FileHandle )
*
* NOTES :
*       These functions are a part of the FM suite;
*       See IMS FM0121 for detailed description.
*
*       Copyright A.N.Other Co. 1990, 1995.  All rights reserved.
*
* AUTHOR :    Arthur Other        START DATE :    16 Jan 99
*
* CHANGES :
*
* REF NO  VERSION DATE    WHO     DETAIL
* F21/33  A.03.04 22Jan99 JR      Function CalcHuffman corrected
*
*H*/
```

Documentação em funções

Muita atenção ao nomear a função:

- semântica vinculada a quem usará a função – e não a quem implementou

Comente:
- O objetivo da função
- O conjunto de dados (e seus tipos) que ela recebe – parâmetros
- O dado (e seu tipo) gerado pela função – valor de retorno

Documentação em funções

```
/* Função que implementa o cálculo da área de um retângulo
 * Dados de entrada:
 *         Base do retângulo, um valor real
 *         Altura do retângulo, um valor real
 * Valor gerado:
 *         A área do retângulo, um valor real
 * Autor: Gabriel Pires, em 01/06/2022
 * Alterada por: Lucas Porto, em 04/12/2024
 *
 */
```

```
/*****************************************************************
* NAME :            int KB_GetLine(pKbdBuf, MaxChars)
*
* DESCRIPTION :     Input line of text from keyboard
*
* INPUTS :
*       PARAMETERS:
*           int     MaxChars                max chars to read before beeping
*       GLOBALS :
*           struct  Terminal                Terminal description (in termdata.h)
*                   char    .BackspaceCode  Code for backspace
*                   char *  .CharSet        keyboard conversion tables
* OUTPUTS :
*       PARAMETERS:
*           char    * pKbdBuf               -> buffer for keyboard chars
*       GLOBALS :
*            None
*       RETURN :
*            Type:   int                    Error code:
*            Values: VALID_DATA             valid read
*                    KB_BAD_DATA            invalid kbd data
*                    KB_DISCONNECTED        keyboard not present
* PROCESS :
*                   [1]  Clear keyboard buffer
*                   [2]  Do
*                   [3]    Get character
*                   [4]    Translate characters
*                   [5]  Until CR or buffer full
*
* NOTES :           Unknown characters returned as '*'
*                   Backspace is the only editing allowed.
* CHANGES :
```

Motivação para UML
Linguagem de Modelagem Unificada

Em OO, o processo de codificação deve ser precedido por um Diagrama de Classes – ao longo do Curso, ele lhe acompanhará em cada etapa.

Sugestões para iniciar:

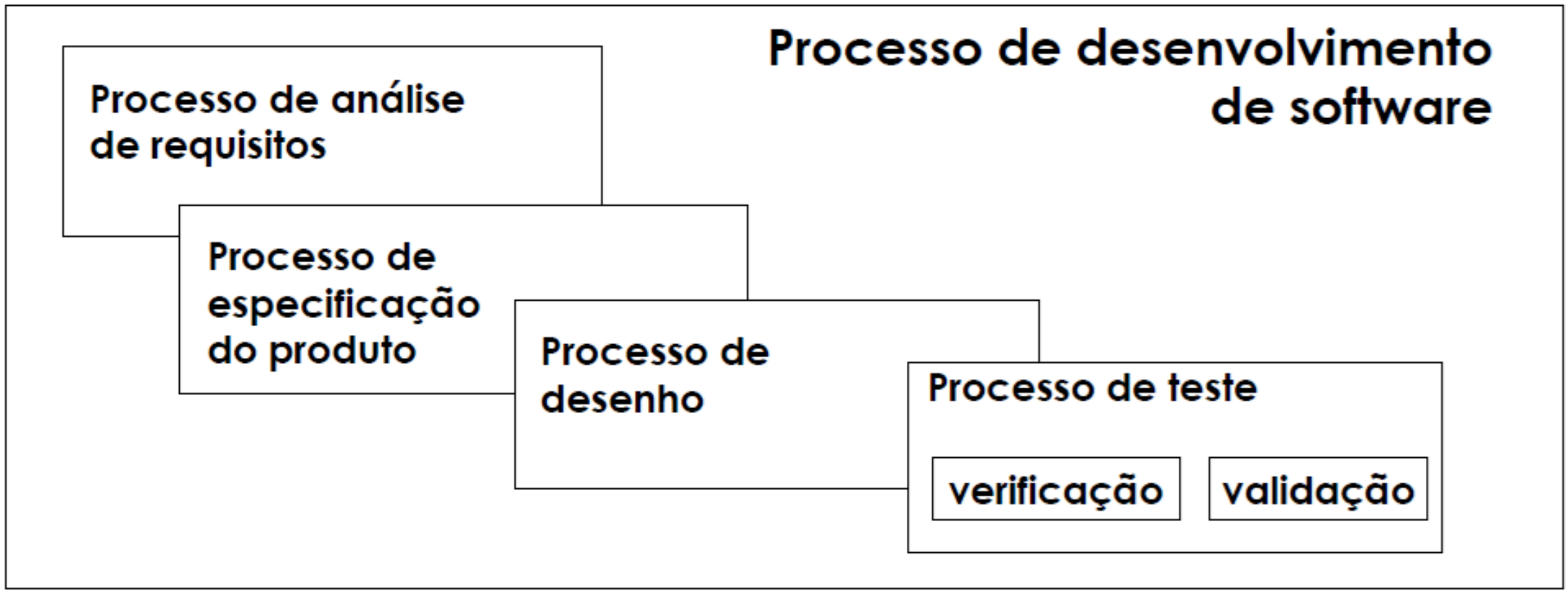
www.drawio.com
staruml.io
argouml-tigris-org.github.io/tigris/argouml

Motivação para teste de software

O *Teste de Software* é um processo fundamental para o desenvolvimento de software com qualidade industrial.

Colabora fundamentalmente para alcançar importantes níveis de qualidade de software, em especial:

- Corretude
- Confiabilidade
- Usabilidade
- Eficiência
- Manutenibilidade
- Portabilidade
- Robustez


Processo de desenvolvimento de software
Processo de análise de requisitos
Processo de especificação do produto
Processo de desenho
Processo de teste
verificação
validação

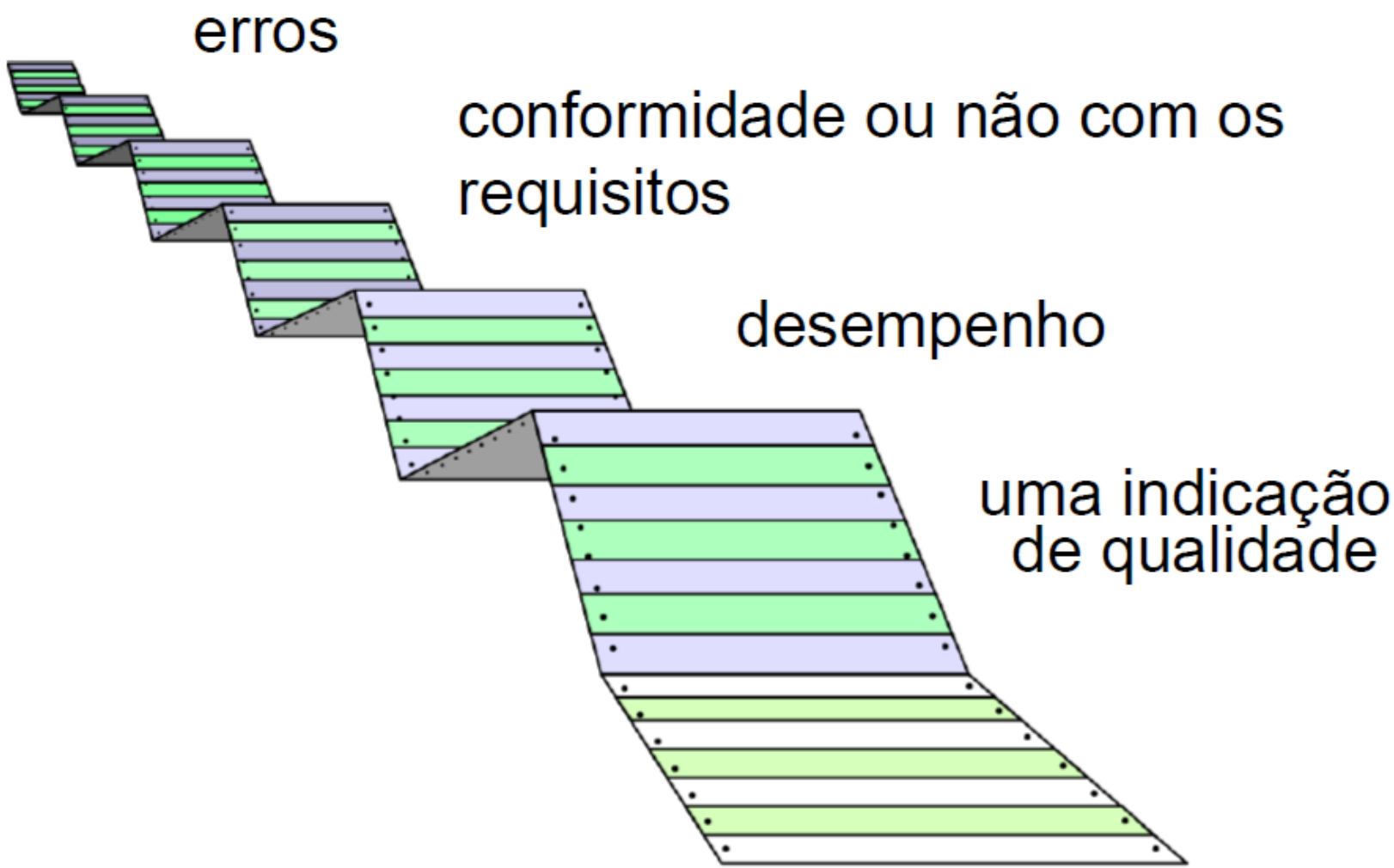
Teste de Software

Testar é a atividade de executar o programa com o intuito específico de encontrar erros antes da sua entrega ao usuário final. Observar o ciclo de vida.

Tarefa:

Validar com diferentes perfis de usuários
Ampliar com diferentes parâmetros de entrada
Validar em diferentes ambientes

O que os testes podem mostrar?



Quando realizar o processo de Teste de Software

Testar o código o mais cedo possível tende a diminuir, e muito, o retrabalho

Lembre-se: o custo com teste, em geral, é bem menor que o requerido em retrabalho

Elementos para o teste do código

- Um conjunto (diverso) de entradas
- Condições de execução
- Saídas esperadas

Trabalho Prático Final

Construa o Diagrama de Classes (UML)

Trabalho Prático Final

- Documente o código
- Teste seu CRUD
- Implemente o *shift -1*

Trabalho Prático Final

Implemente persistência:

- escolha vetor de objetos para instrução única
- ou varra o *array* para armazenar e varra o arquivo para preencher o *array*

Recordando estratégias para persistência

```
// Salvando o vetor em uma única instrução
FILE* arq = fopen("pessoas.dat", "w");


fwrite(Pessoas, sizeof(Pessoa), TAM, arq);


fclose(arq);
```

Recordando estratégias para persistência

```
// Vetor de ponteiros: salvar um elemento por vez
FILE* arq = fopen("pessoas.dat", "w");

   for(int i=0; i<TAM; i++){
     fwrite(Pessoas[i], sizeof(Pessoa), 1, arq);
   }

   fclose(arq);
```

Dicas, sugestões, métodos e técnicas

C / C++

UML

O laço *for-each*

```
#include <iostream>
using namespace std;

int main() {
    int array[]={1,2,3,4,5};
    for( int x : array) {
            printf("\n%d", x);
    } // fim for(x)
    return 0;
} // fim main()
```

O laço *for-each*

```
#include <iostream>
using namespace std;

int main() {
    string array[]={”Lucas”, ”Gabriel”, ”Ana”, ”Mateus”};
    for( string nome : array) {
            cout << endl << nome;
    } // fim for(nome)
    return 0;
} // fim main()
```

Motivação para documentação do código-fonte

Obs: Sempre que provido pela linguagem, opte pelo for each quando for necessário varrer por completo um arranjo. E comente o código.

```
...
  cout << "Notas acima da média:“ << endl;

  // Listar as notas acima da média armazenadas no vetor notas
  for (double nota : notas) {
            if (nota > media) {
                      cout << endl << nota;
            }
  }
```

O laço *for-each*

```
class Aluno {
    private :
            string nome;
    public   :
            Aluno(){};
            Aluno(string nome){
                setNome(nome);
            }
            string getNome(){
                return this->nome;
            }
            void setNome(string nome){
                this->nome = nome;
            }
};
```

```
#include <iostream>
using namespace std;

int main() {
    Aluno* array[3];
    array[0] = new Aluno("Lucas");
    array[1] = new Aluno("Gabriel");
    array[2] = new Aluno("Any");

    // Listando toda a turma
    for( Aluno* aluno : array) {
            cout << aluno->getNome() << endl;
    }
    return 0;
} // fim main()
```

Trabalho Prático Final

Em todos os casos que for viável, opte pelo *for-each* para implementar os laços de varredura de um vetor...

e comente.

Dicas, sugestões, métodos e técnicas

C / C++

UML

Número aleatório em C

```c
#include <stdlib.h>  // Para rand()

int main() {

    int X = rand ();

    printf("\n%d", X);

    return 0;
} // fim main()
```

Número aleatório em C

```c
#include <stdlib.h>  // Para rand()
#include <time.h>  // Para time()

int main() {
   srand( time(0) ); //inicializa a semente com o tempo atual
   int X = rand ();

   printf("\n%d", X);

   return 0;
} // fim main()
```

Número aleatório em C

```c
#include <stdlib.h>  // Para rand()
#include <time.h>  // Para time()


int main() {
    srand( time(0) ); //inicializa a semente com o tempo atual
    int X = rand () % 11;    // Limita a intervalo entre 0 e 10

    printf("\n%d", X);

    return 0;
} // fim main()
```

Número aleatório em C++

```
#include <iostream>  // Para cout
#include <cstdlib>  // Para rand()
#include <ctime>  // Para time()

int main() {
   srand( time(0) ); //inicializa a semente com o tempo atual
   int X = rand () % 11;    // Limita a intervalo entre 0 e 10

   cout << endl <<  X);

   return 0;
} // fim main()
```

Tente

1. Um vetor deve armazenar uma coleção de 10 números. Os números precisam estar em um intervalo fechado de 0 a 100. Sorteie os números para o vetor e o escreva.

2. Construa um programa que sugira seis números para o jogo da Megasena.

Dicas, sugestões, métodos e técnicas
C++

Tratamento de Exceções

Inspiração

- Sistemas críticos
- Qualidade industrial
- Grau de Robustez
- Linguagem ADA

Motivação para Programação Orientada por Eventos

Tipicamente, operações são requeridas em um *software* a partir de um fluxo de controle.

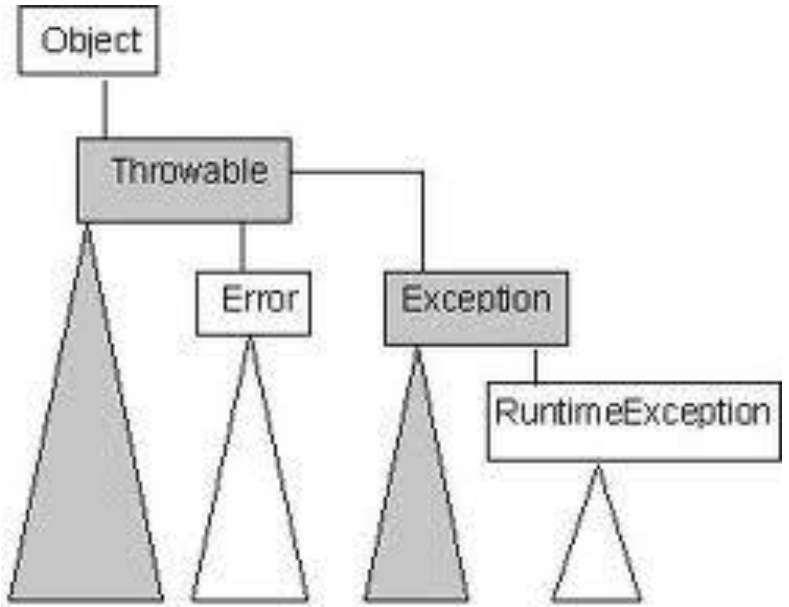
Considere a hipótese de uma operação ser requerida pelo fato de algum evento acontecer.

Ex. de motivação para Programação Orientada por Eventos

Criação de interfaces com o usuário

Ex. de motivação para Programação Orientada por Eventos

Aumentar o grau de robustez de um *software*



Linguagens e o tratamento de exceções

Os principais mecanismos para tratamento de exceções foram incorporados à linguagem

Ada;
C++, que se inspirou no mecanismo de Ada;
Java, que se inspirou no mecanismo de C++.

Conceito de exceções

Uma exceção é um sinal que indica que algum tipo de **condição excepcional** ocorreu durante a **execução**.

Por isto as exceções estão associadas a condições de erro **não tratadas em tempo de compilação**, mas em **tempo de execução**.

Com o tratamento de exceções, depois de lidar com um **problema**, um programa pode continuar **executando em vez de encerrar**.

Ex. de ganho com Tratamento de Exceções

O grau de **robustez** adquirido colabora para o que é chamado de

computação de missão crítica.

Escrever programas *robustos* e *tolerantes a falha*.

Alguns exemplos de exceções

- acessar um elemento de um arranjo depois de sua **última posição** *(a depender da linguagem)*;
- receber uma referência **null** onde se espera um objeto;
- ler um tipo diferente do programado;
- realizar uma divisão por zero.

Imagine um código assim escrito para ampliar o grau de robustez:

Realize uma tarefa

Se a tarefa anterior não tiver sido executada corretamente

Realize processamento de erro

Realize a próxima tarefa

Se a tarefa anterior não tiver sido executada corretamente

Realize processamento de erro

...

Considere:

Mecanismo para Tratamento de Exceções em C++

Alternativa:
Mecanismo para tratamento de exceções

Permite tratar uma exceção antes de abortar a execução do código.

Para isto, ao gerar um erro, uma exceção é lançada.

Palavras-chave:

try

throw

catch

Formam a instrução **try..catch()**

#include <stdexcept> // arquivo de cabeçalho **stdexcept** contém **runtime_error**

using std::runtime_error; // classe runtime_error da biblioteca-padrão do C++

Lembre-se das bolas nos quintais das casas...

```
int main() {
  int x = -1;
  cout << “\nInstrucao antes do bloco try\n";
  try {
      cout << “\nInstrucao protegida pelo bloco try \n";
      if (x < 0) {
        throw x;
        cout << “Instrução apos um throw: nunca executada\n";
      }
  }
  catch (int x) {
      cout << “\nExcecao capturada por um catch\n";
  }
  cout << "\nCodigo apos instrução try..catch executada\n";
  return 0;
}
```