UNIVERSIDADE ESTADUAL DO NORTE FLUMINENSE
LABORATÓRIO DE ENGENHARIA E EXPLORAÇÃO DE PETRÓLEO

PROJETO ENGENHARIA
DESENVOLVIMENTO DO SOFTWARE

CÁLCULO DA PERDA DE CARGA DISTRIBUIDA DE UM FLUIDO NO DECORRER DO ESCOAMENTO

TRABALHO DA DISCIPLINA PROGRAMAÇÃO PRÁTICA

Versão 4:

Lucas Isaac Vieira Oliveira

Mateus Nascimento Gonçalves da Rocha

Prof. André Duarte Bueno

MACAÉ - RJ

Novembro - 2018

Sumário

Capítulo 1

Introdução

No presente projeto de engenharia desenvolve-se o software "Cálculo da perda de carga distribuída de um fluido no decorrer do escoamento". O mesmo tem como finalidade calcular a perda de carga distribuída de um fluido no decorrer do escoamento ao longo da tubulação. Os cálculos são realizados a partir dos dados fornecidos pelo usuário. O software é aplicado a engenharia de petróleo e utiliza o paradigma da orientação a objetos e a linguagem C++.

1.1 Escopo do problema

Como a realização destes cálculos manualmente é um processo tedioso e propenso a erros, o desenvolvimento de um programa para este fim é uma ferramenta útil ao aprendizado e aplicação destes conceitos.

As características e funções necessárias para "Cálculo da perda de carga distribuída de um fluido no decorrer do escoamento" incluem o cálculo das perdas de carga distribuída nas tubulações e acessórios do escoamento. Os cálculos são de grande importância, pois influenciam no dimensionamento dos projetos, máquinas e fluxos do escoamento dos fluidos.

A dissipação de energia por unidade de peso acarreta em uma diminuição da pressão estática do escoamento, influenciando a velocidade do fluido e variações de elevação. O efeito do atrito age no sentido de diminuir a pressão, isto é, o de causar uma "perda" de pressão comparada com a do caso ideal de escoamento livre de fricção.

1.2 Objetivos

Os objetivos deste projeto de engenharia são:

- Objetivo geral:
  - Desenvolver um projeto de engenharia de software para calcular a perda de carga distribuída de um fluido no decorrer do escoamento ao longo da tubulação.
- Objetivos específicos:
  - Modelar física e matematicamente o problema.
  - Modelagem estática do software (diagramas de caso de uso, de pacotes, de classes).
  - Modelagem dinâmica do software (diagramas exemplificando os fluxos de processamento).
  - Implementação dos algoritmos.
  - Criar classe para conversão de unidades.
  - Calcular dados do escoamento (vazão, fator de atrito, número de Reynolds).
  - Salvar dados em disco, para possível comparação de materiais usados.
  - Simular (realizar simulações para teste do software desenvolvido).
  - Documentação do software.

Capítulo 2

Especificação

Apresenta-se neste capítulo do projeto de engenharia a concepção e a especificação do sistema a ser modelado e desenvolvido.

2.1 Especificação do programa -descrição dos requisitos

- O projeto a ser desenvolvido consiste em um software que primeiramente pede pelos dados iniciais (comprimento, velocidade do fluido, diâmetro do tubo, massa especifica, diâmetro hidráulico, viscosidade dinâmica, viscosidade cinemática). O programa então recebe o nome do arquivo no qual contem os materiais e suas devidas rugosidades, assim será definido pelo usuário o material que o tubo é feito e então é calculado a rugosidade relativa, o coeficiente de Reynold, que permite avaliar o tipo do escoamento, o fator de atrito e por conseqüência a perda de carga distribuída e a vazão, sendo tais dados citados acima retornados ao usuário.
- O conceito de programação orientada a objeto será utilizado no desenvolvido do software.
- A licença é a GPL 2.0

2.2 Especificação do software - requisitos

2.2.1 Nome do sistema / produto

Nome	Cálculo da perda de carga de um fluido no decorrer do escoamento
Componentes principais	Métodos para cálculo de perda de carga distribuída, tipo de escoamento e vazão
Missão	Cálculo de propriedades da mêcanica dos fluidos

2.2.2 Requisitos funcionais

Apresenta-se a seguir os requisitos funcionais.

RF-01	O sistema deve conter uma base de dados com nomes de materiais e determinadas rugosidades.

RF-02	O usuário deverá ter liberdade para escolher com qual material que se deseja trabalhar.

RF-03	Deve permitir a escolha do tipo de escoamento.

RF-04	O usuário poderá plotar a perda de carga para determinada rugosidade do material. O gráfico poderá ser salvo como imagem ou ter seus dados exportados como texto.

2.2.3 Requisitos não funcionais

RNF-01	O programa deverá ser multi-plataforma, podendo ser executado em Windows, GNU/Linux ou Mac OS X.

2.3 Casos de uso

Nome do caso de uso:	Cálculo de uma propriedade do escoamento.
Resumo/Descrição:	Cálculo de uma propriedade do escoamento a partir de propriedades do meio e do fluido.
Etapas:	1. Criar objeto da perda de carga de um fluido no decorrer do escoamento. 2. Entrar com dados da tubulação: Comprimento do tubo, Diâmetro do Tubo, Rugosidade. 3. Entrada dados do escoamento: Velocidade do Fluido, Massa Especifica do fluido, Diâmetro Hidráulico, Viscosidade Dinâmica, Viscosidade Dinâmica, Viscosidade Cinemática. 4. Definir o material para analise das propriedades do fluido. 5. Cálcular rugosidade relativa. 6. Cálcular coeficiente de Reynold. 7. Cálcular vazão. 8. Cálcular fator de atrito 9. Analisar resultados. 10. Cálcular propriedade principal (perda de carga distribuída). 11. Gerar gráfico: Perda de carga x Material.
Cenários alternativos:	1. Entrada errada do usuário(por exemplo: a função é logarítmica, e o intervalo vai de -1 a 10). 2. O software apresentará um bug quando for determinar o logaritmo de -1.

2.3.1 Diagrama de caso de uso geral

O diagrama de caso de uso geral da Figura 2.1 mostra o usuário acessando os sistemas de ajuda do software, calculando a área de uma função ou analisando resultados.

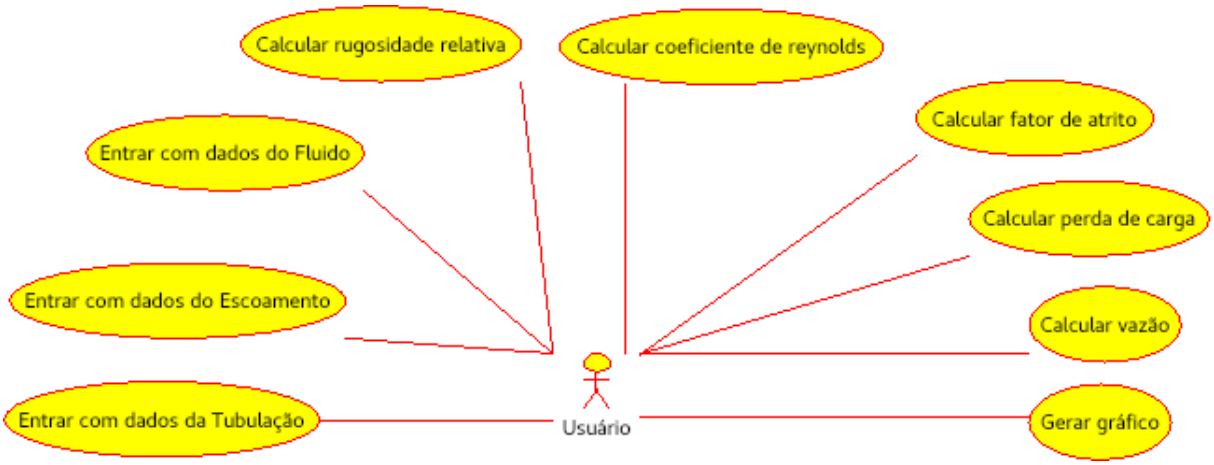

Figura 2.1: Diagrama de caso de uso – Caso de uso geral

Capítulo 3

Elaboração

Neste capítulo faremos a análise de domínio, identificação e diagrama de pacotes.

3.1 Análise de domínio

O desenvolvimento deste software tem como área principal a mecânica dos fluidos, sendo o ramo da mecânica que estuda o comportamento físico dos fluidos e suas propriedades. Os aspectos teóricos e práticos da mecânica dos fluidos são importantes para a solução de diversos problemas encontrados na engenharia, sendo suas principais aplicações destinadas ao estudo de escoamentos de líquidos e gases, máquinas hidráulicas, aplicações de pneumática e hidráulica industrial, entre outros.

Outra área relacionada é a elevação e escoamento em engenharia de petróleo, associado ao processo através do qual os líquidos produzidos por um reservatório são transportados do fundo do poço até a cabeça do poço e deste ponto até a plataforma, na superfície.

3.2 Formulação teórica

3.2.1 Tipos de Escoamento

• Escoamento Laminar:

Aquele em que as partículas de fluido deslocam-se em lâminas individualizadas (camadas adjacentes de fluido movem-se paralelamente entre si), sem trocas de massa entre elas.

Exemplos: escoamento de água numa torneira pouco aberta, injeção de fluido em reservatórios, etc.

• Escoamento Turbulento:

Aquele em que as partículas apresentam um movimento caótico (aleatório) macroscópico, isto é, a velocidade apresenta componentes transversais ao movimento geral do fluido, e

consequentes trocas de massa (transversais) entre as partículas de fluido. É o mais comum na natureza.

Exemplos: água jorrando de uma torneira totalmente aberta, tubulações industriais, fluxo de fluido em um meio poroso na região próxima ao poço produtor, etc.

3.2.2 Perda de carga

A perda de carga distribuída é aquela que ocorre ao longo de condutos retos e de seção constante devido aos atritos no escoamento. Ela será calculada utilizando a Equação 3.1:

$$h\mathtt{f} = f(\frac{L}{D\mathtt{H}})v^{\mathtt{2}} \tag{3.1}$$

Na qual $hf[m]$ representa a perda de carga distribuida, f o fator de atrito, $L[m]$ o comprimento da tubulação, $D\mathtt{H}[m]$ o diâmetro hidráulico do tubo e $v[m^2/s]$ a velocidade média da seção.

Para o cálculo da perda de carga distribuida é necessário a utilização e compreensão de conceitos que serão apresentados a seguir.

3.2.3 Rugosidade ($\mathscr{E}[mm]$)

Conjunto de irregularidades (asperezas) nas paredes internas dos condutos, isto é, desvios de altura nas paredes internas devido às mesmas não serem perfeitamente lisas, como representado na Figura 3.1.

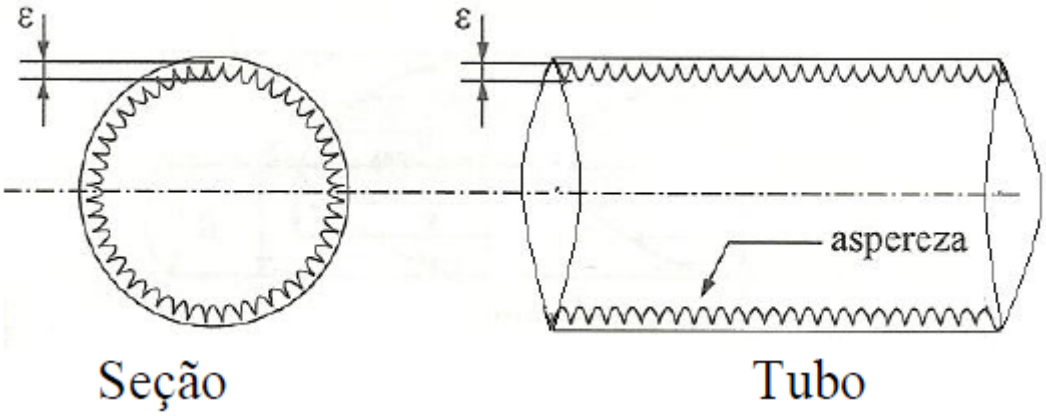

Figura 3.1: Tubo com rugosidades

A rugosidade gera dois efeitos no escoamento:

1. Gera perdas de carga devido ao atrito entre o fluido e as rugosidades.
2. Favorece turbulências no escoamento devido às colisões das partículas de fluido com as rugosidades, seguidas de movimentos transversais das partículas.

Nos cálculos do fator de atrito(f) utilizaremos a rugosidade relativa que é a razão entre a rugosidade média e o diâmetro do conduto representada pela Equação 3.2.

$$R\mathtt{DH} = \frac{\mathscr{E}}{D\mathtt{H}} \tag{3.2}$$

Os materiais que compõem os tubos utilizados na indústria possuem rugosidade conhecida, a Tabela 3.1 representa a relação entre material e rugosidade.

Tabela 3.1: Material x Rugosidade

Tubo	Rugosidade(mm)
Aço rebitado (rebite)	4,5
Concreto (manilha)	1,5
Madeira	0,6
Ferro Fundido	0,26
Ferro galvanizado ou forjado (eletrólise)	0,15
Aço comercial (Ferro + Carbono)	0,046
Vidro, acrílico e PVC	0,0 (liso)

3.2.4 Fator de atrito (f)

O fator de atrito é uma das formas de expressar a perda de carga distribuída em termos de uma tensão de cisalhamento adimensional na parede do tubo. É definido empiricamente de acordo com a Equação 3.3

$$f = \frac{\tau}{\frac{1}{8}v^{\mathbf{2}}\rho} \tag{3.3}$$

No qual $v[\frac{m}{s}]$ é a velocidade média na seção, $\tau[\frac{kgf}{m^2}]$ é a tensão de cisalhamento na parede interna do tubo e $\rho[\frac{kg}{m}]$ é a massa especifica do fluido.

3.2.5 Numero de Reynolds

O número de Reynolds é adimensional e serve para classificar os escoamentos. É dado pela combinação de um comprimento de escala $L[m]$, da velocidade $v[m/s]$, da viscosidade cinemática do fluido $V[\frac{m^2}{s}]$, viscosidade dinâmica $\mu[N.s/m^2]$, é representado de acordo com as Equações 3.4 e 3.5

$$Re = \frac{vL}{V} \tag{3.4}$$

$$Re = \frac{\rho vL}{\mu} \tag{3.5}$$

Classificação Parede da Tubulação

As paredes do tubo também possuem classificações de acordo com a Tabela 3.2

Parede	Região		Limites	Perfil $u(y)$
Hidraulicamente Lisa	Região da parede	Subcamada Visosa	$0 \leq \frac{u_t.y}{v} \leq 5$	$\frac{u}{u_t} = \frac{u_t.y}{v}$
		Zona intermediária	$5 \leq \frac{u_t.y}{v} \leq 30$	$\frac{u}{u_t} = g(y)(sódadosempiricos)$
		Região Turbulenta	$30 \leq \frac{u_t.y}{v}$ e $\frac{y}{r_0} \leq 0,15$	$\frac{u}{u_t} = 2,44.ln(\frac{u_t.y}{v}) + 4,9$
Hidraulicamente rugosa	Região da parede	Subcamada Visosa	$\frac{y}{r_0} \leq 0,15$	$\frac{u}{u_t} = 2,44.ln(\frac{y}{\varepsilon}) + 8,5$
		Zona intermediária		
		Região Turbulenta		
Hidraulicamente lisa, rugosa (ou transição)	Região externa à parede		$30 \leq \frac{u_t.y}{v}$ e $\frac{y}{r_0} \leq 1$	$\frac{u_{MAX}-u}{u_t} = 2,44.ln(\frac{r_0}{y}) + 0,8$

Tabela 3.2: Tabela Tipo de Parede da Tubulação e Distribuição das velocidades [APOSTILA DE MECÂNICA DOS FLUIDOS PARA O CURSO DE ENGENHARIA DE EXPLORAÇÃO E PRODUÇÃO DE PETRÓLEO]

3.2.6 Diagrama de Moody-Rouse

O diagrama de Moody-Rouse é um diagrama empírico que relaciona o fator de atrito com o número de Reynolds do escoamento e a rugosidade relativa do conduto. Este diagrama é aplicado em escoamentos nas mesmas condições da Figura 3.1 e também a outras geometrias dos condutos fazendo-se uso do diâmetro hidráulico.

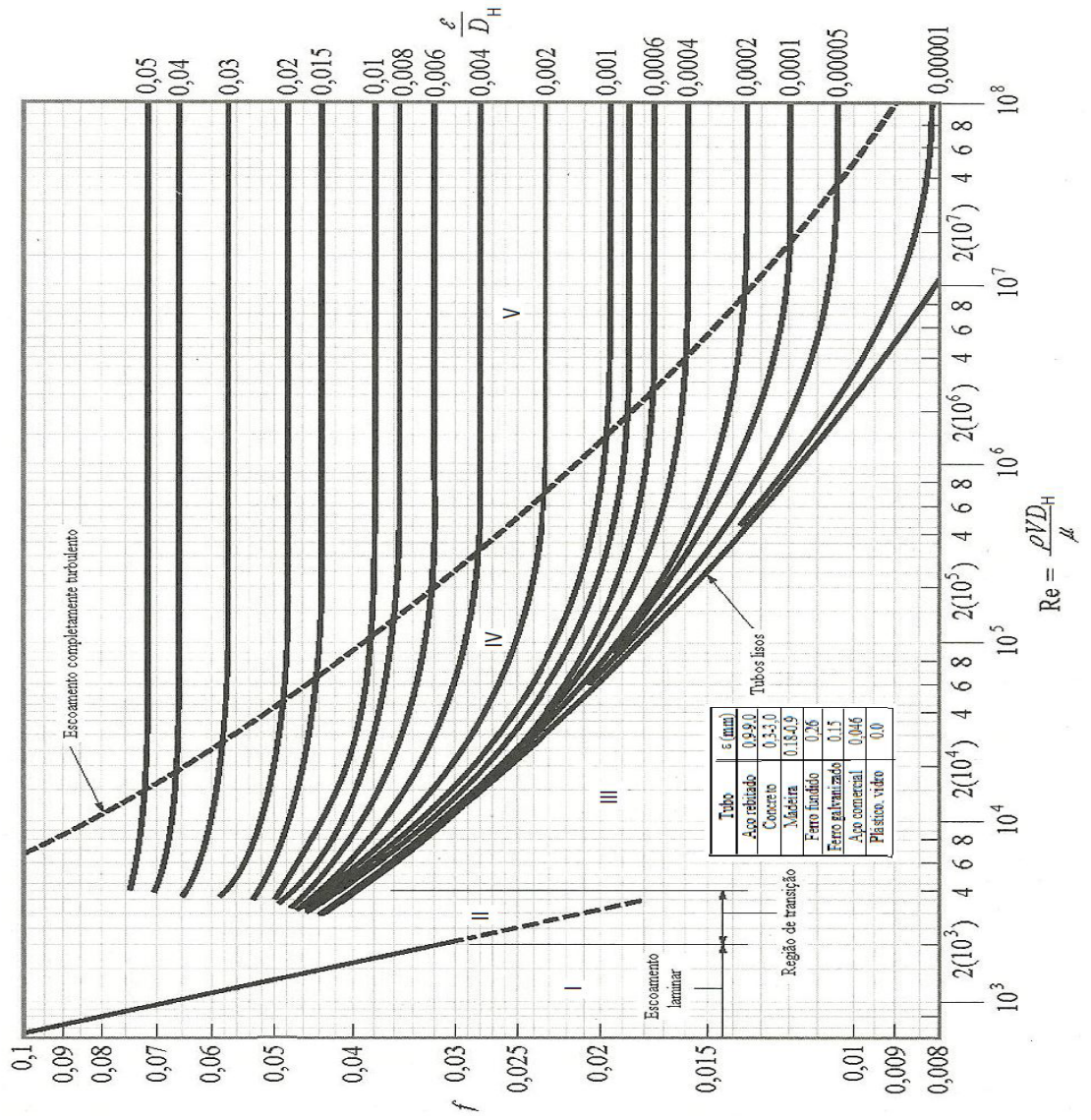

Figura 3.2: Diagrama de Moody-Rouse[APOSTILA DE MECÂNICA DOS FLUIDOS PARA O CURSO DE ENGENHARIA DE EXPLORAÇÃO E PRODUÇÃO DE PETRÓLEO]

Sintese teoria

A Tabela 3.3 representa as equações do fator de atrito de acordo com o tipo de escoamento e numero de Reynolds.

Tabela 3.3: Análises fisicas do Diagrama de Moody

Região	Nº de Reynolds	Descrição	Equação
I	$Re < 2000$ (Laminar)	Forças viscosas (cisalhamento) elevadas em relação às demais, exercendo impacto bem maior que a rugosidade para as perdas de carga.	$f = \frac{64}{Re}$
II	$2000 < Re < 4000$ (Transição)	Forças viscosas ora elevadas, ora desprezíveis em relação às demais, exercendo impacto ora maior, ora menor que a rugosidade para as perdas de carga.	Faltam dados empíricos
III	$Re > 4000$ (Turbulento. Hid. liso)	Forças viscosas significativas, sendo a rugosidade da parede coberta pela SCLL, com contribuição da rugosidade desprezível para as perdas de carga.	$\frac{1}{\sqrt{f}} = 0,86.ln(Re.\sqrt{f}) - 0,8$
IV	$Re > 4000$ (Turbulento. Transição)	A rugosidade é comparável à SCLL, contribuindo para as perdas de carga.	- $\frac{1}{\sqrt{f}} = 0,86ln(\frac{\varepsilon}{3,7.D_H} + \frac{2,51}{Re.\sqrt{f}})$
V	$Re >> 4000$ (Turbulento. Hid. rugoso)	Forças viscosas desprezíveis em relação às demais, e a rugosidade ultrapassa totalmente a SCLL contribuindo fortemente para as perdas de carga.	$\frac{1}{\sqrt{f}} = -0,86.ln(\frac{\varepsilon}{3,7.D_H})$

[Fonte: APOSTILA DE MECÂNICA DOS FLUIDOS PARA O CURSO DE ENGENHARIA DE EXPLORAÇÃO E PRODUÇÃO DE PETRÓLEO]

3.3 Identificação de pacotes – assuntos

- Banco de Dados: Pacote que armazena e modifica unidades dos dados dos fluidos, materiais e escoamento. Possui métodos para gravar e ler dados em disco, além de mostrar em tela para o usuário.
- Simulação: Gerencia a simulação.

3.4 Diagrama de pacotes – assuntos

O diagrama de pacotes da Figura 3.3 mostra a relação entre os diferentes pacotes do software desenvolvido.

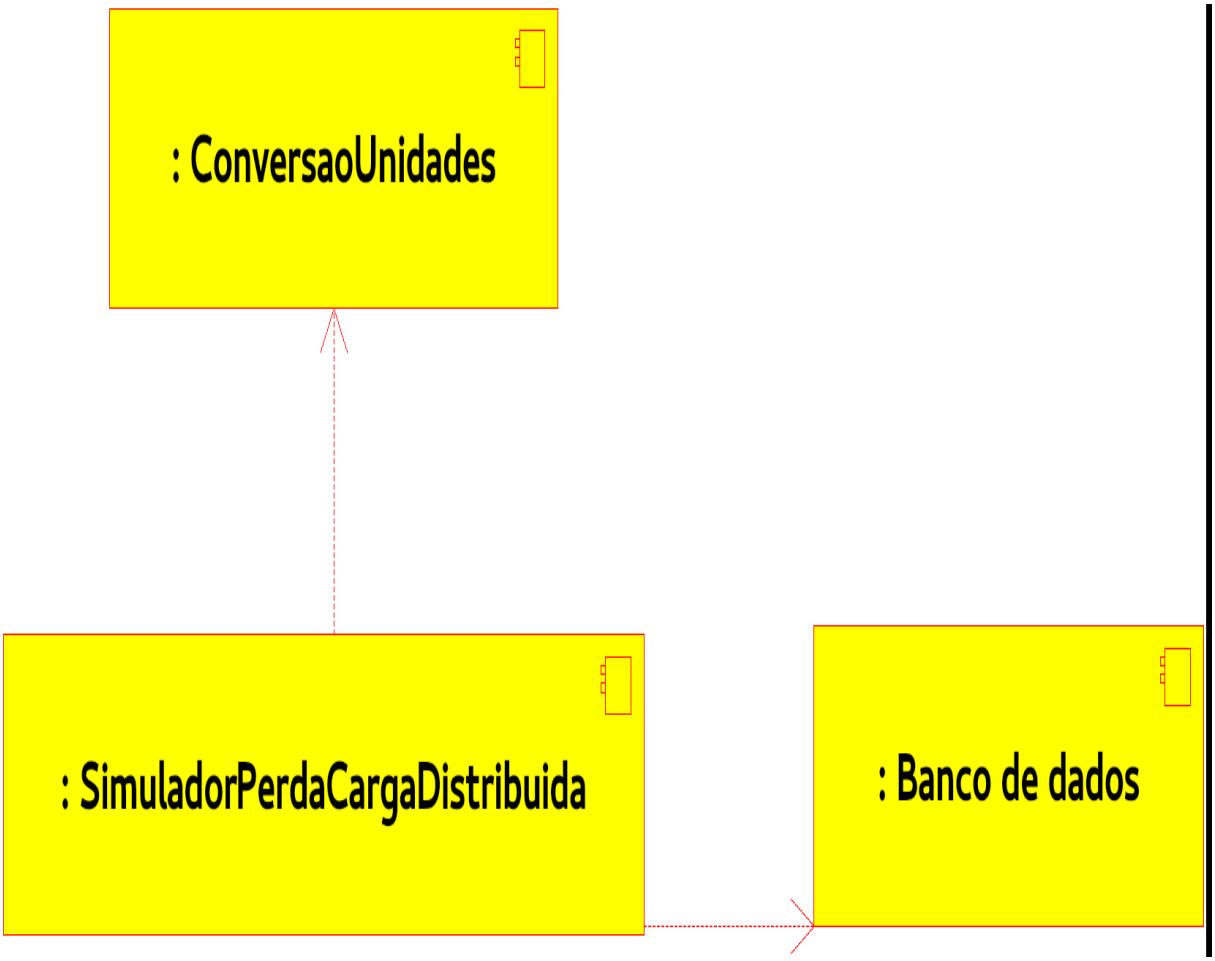

Figura 3.3: Diagrama de Pacotes

Capítulo 4

AOO – Análise Orientada a Objeto

A terceira etapa do desenvolvimento de um projeto de engenharia, no nosso caso um software aplicado a engenharia de petróleo, é a AOO – Análise Orientada a Objeto. A AOO utiliza algumas regras para identificar os objetos de interesse, as relações entre as classes, os atributos, os métodos, as heranças, as associações, as agregações, as composições e as dependências. O modelo de análise deve ser conciso, simplificado e deve mostrar o que deve ser feito, não se preocupando como isso será realizado. O resultado da análise é um conjunto de diagramas que identificam os objetos e seus relacionamentos.

4.1 Diagramas de classes

O diagrama de classes é apresentado na Figura 4.1.

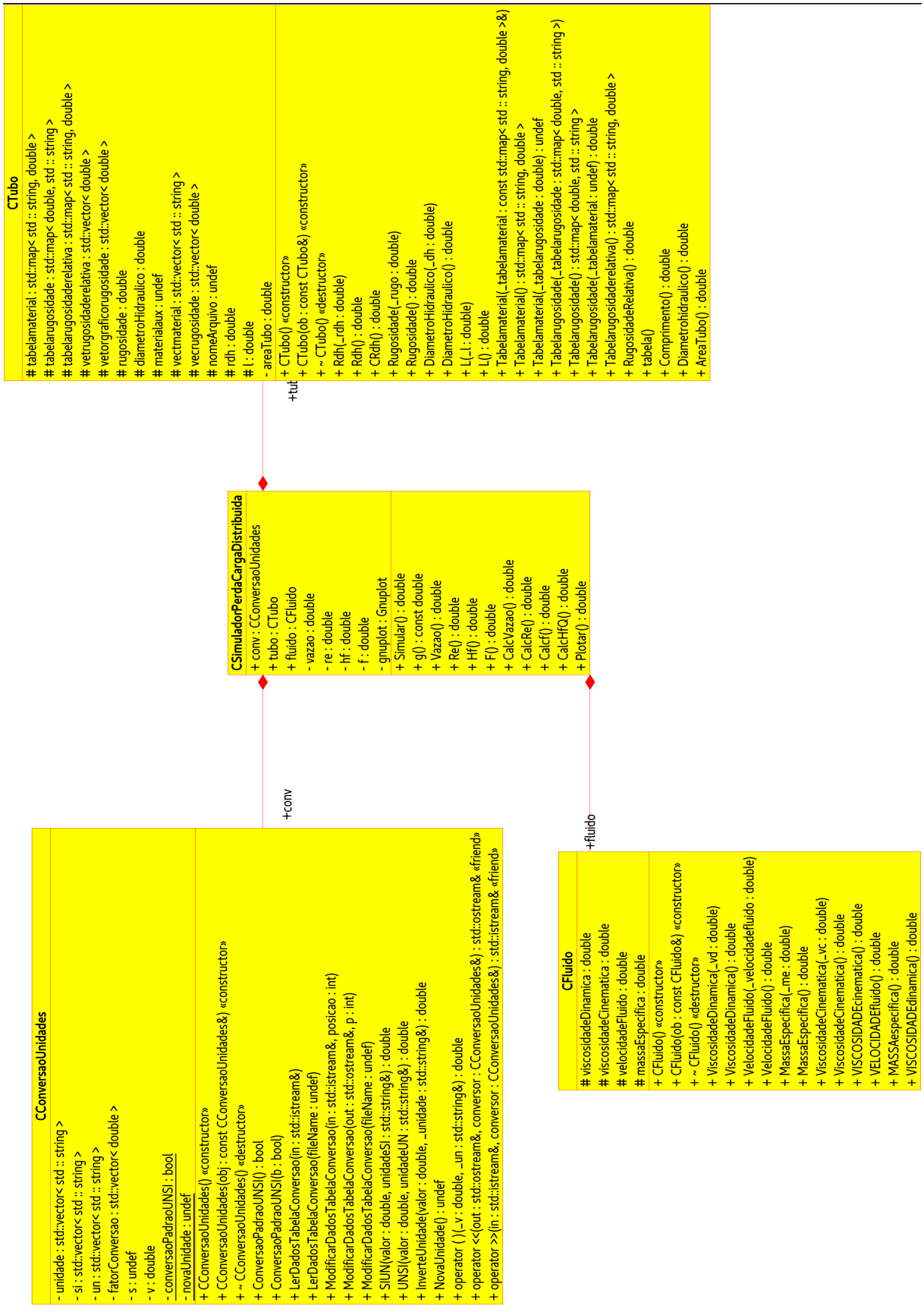

Figura 4.1: Diagrama de classes

4.1.1 Dicionário de classes

- Classe CConversaoUnidades: Muda unidades americanas CGS para unidades MKS
- Classe CTubo: Representa o tubo com suas propriedades.
- Classe CFluido: Representa o fluido com suas propriedades.
- Classe CSimuladorPerdaCargaDistribuida: Realiza o cálculo do escoamento, a simulação de todos os processos, gerenciando a simulação. Gerencia a simulação.
- Classe CGnuplot: Plota o gráfico Material x Perda de carga.

4.2 Diagrama de sequência – eventos e mensagens

O diagrama de seqüência enfatiza a troca de eventos e mensagens e sua ordem temporal. Contém informações sobre o fluxo de controle do software. Costuma ser montado a partir de um diagrama de caso de uso e estabelece o relacionamento dos atores (usuários e sistemas externos) com alguns objetos do sistema.

4.2.1 Diagrama de sequência

No diagrama de sequência o usuário faz uso da classe CSimuladorPerdaCargaDistribuida, que utiliza as classes CTubo, CFluido, CConversaoUnidades e CGnuplot.

Após o cálculo da perda de carga distribuída é gerado um gráfico que é mostrado em tela para o usuário. Veja o diagrama de seqüência na Figura 4.2.

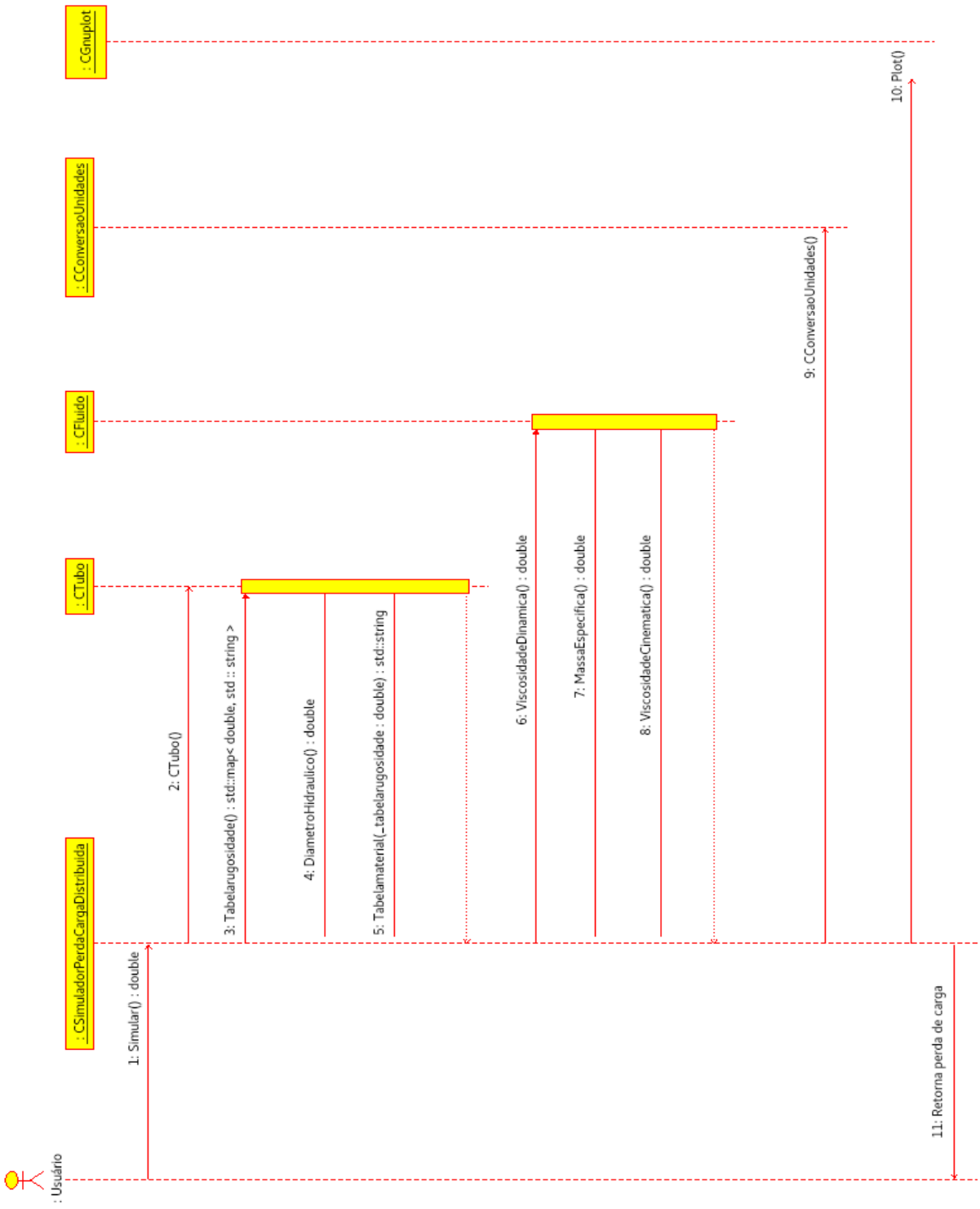

Figura 4.2: Diagrama de sequência

4.3 Diagrama de comunicação-colaboração

No diagrama de comunicação da Figura 4.3 o foco é a interação e a troca de mensagens e dados entre os objetos. Observe que o diagrama de comunicação pode ser desenvolvido como uma extensão do diagrama de caso de uso, detalhando o caso de uso por meio da inclusão dos objetos, mensagens e parâmetros trocados entre os objetos.

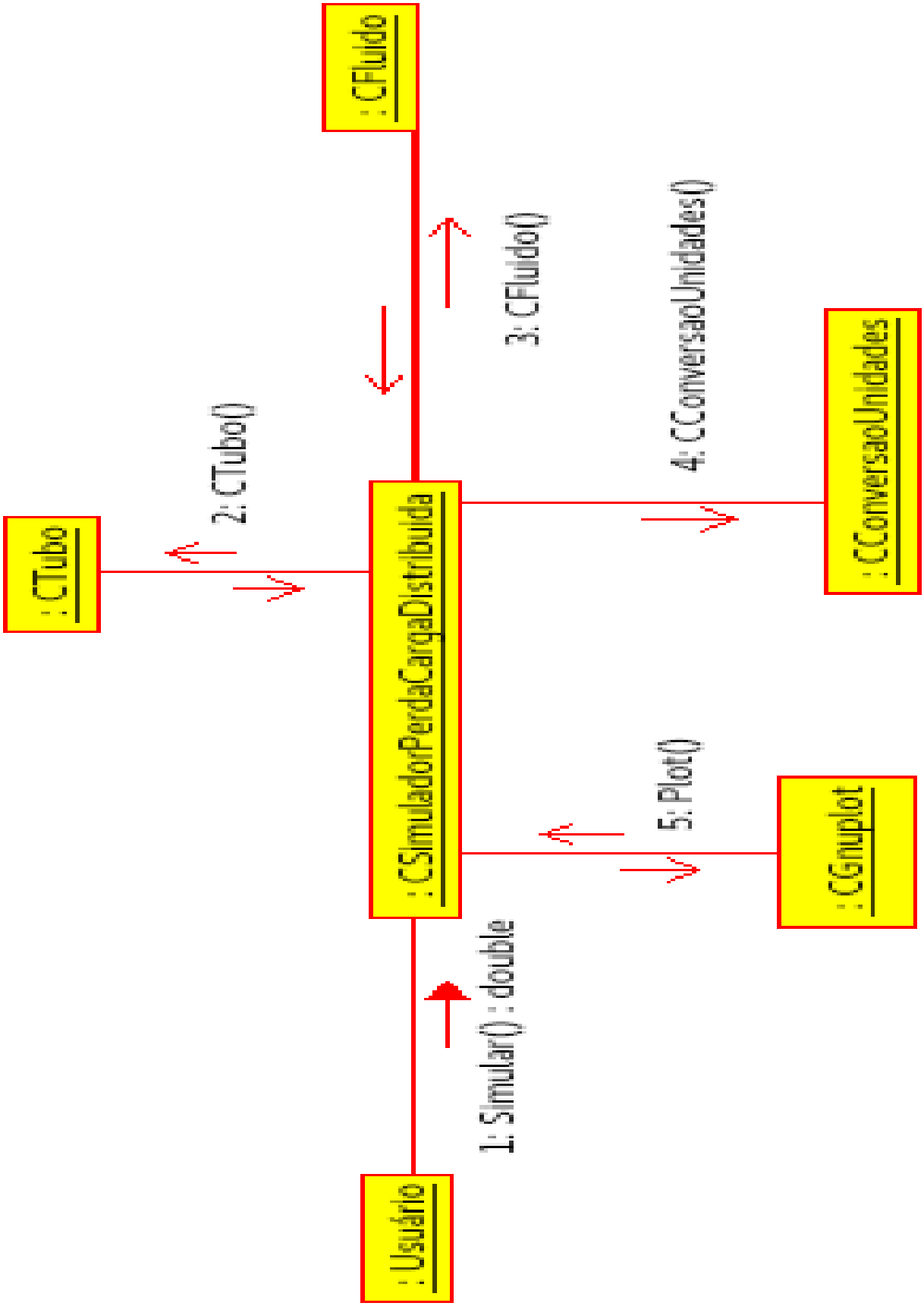

Figura 4.3: Diagrama de Comunicação

4.4 Diagrama de máquina de estado

Um diagrama de máquina de estado representa os diversos estados que um objeto assume e os eventos que ocorrem ao longo de sua vida ou mesmo ao longo de um processo (histórico do objeto). É usado para modelar aspectos dinâmicos do objeto.

Veja na Figura 4.4 o diagrama de maquina de estado para o objeto fator de atrito, por meio dos métodos CalcRe( ) e Calcf( ).

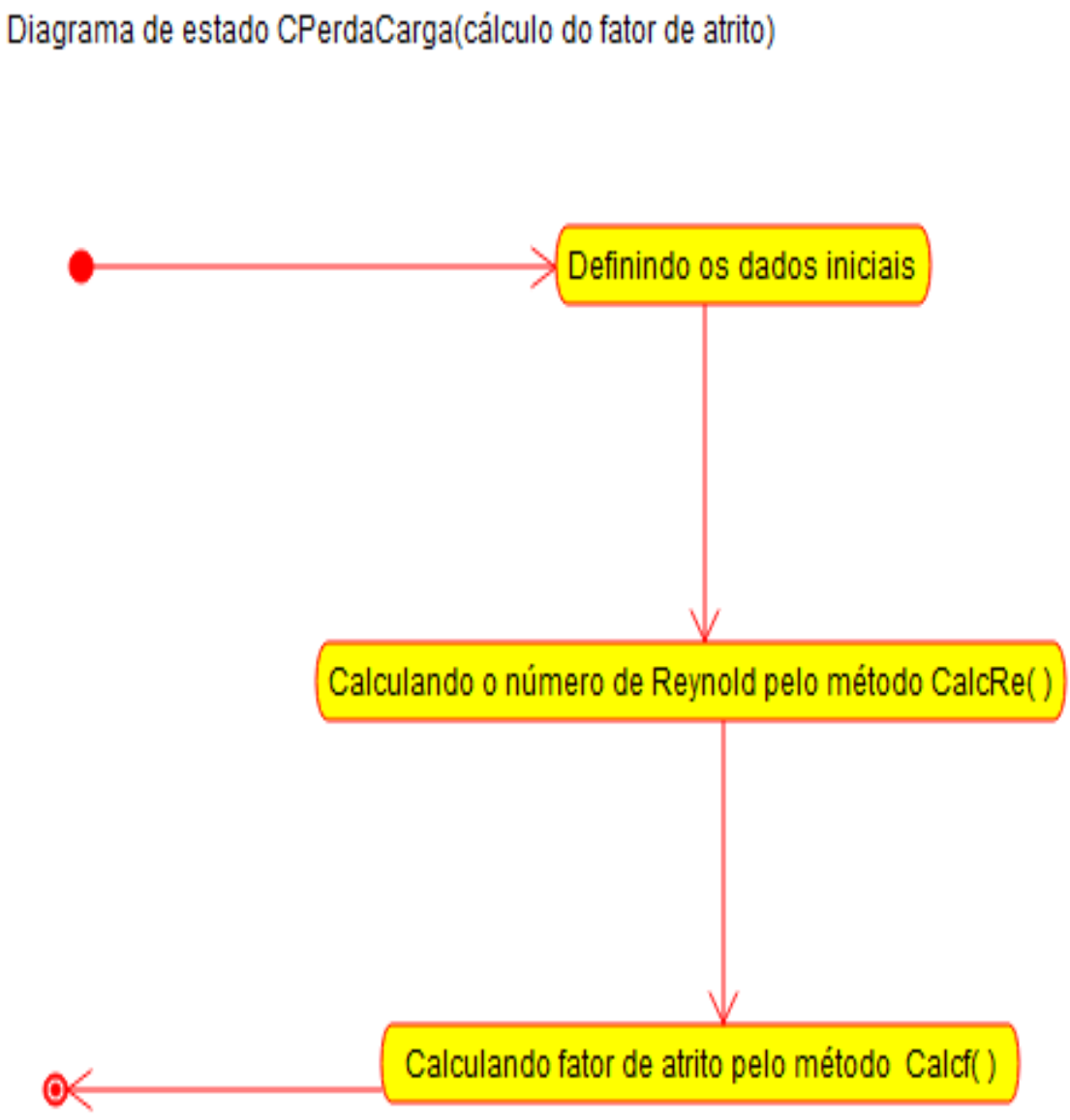

Figura 4.4: Diagrama de máquina de estado

4.5 Diagrama de atividades

Veja na Figura 4.5 o diagrama de atividades correspondente a uma atividade específica do diagrama de máquina de estado. Observe que a partir dos dados iniciais ocorre o cálculo do número de Reynolds, e da rugosidade relativa, os mesmos serão usados na determinação do tipo de escoamento (laminar, turbulento hidraulicamente liso, turbulento, transição entre hidraulicamente liso e rugoso, completamente turbulento e rugoso).

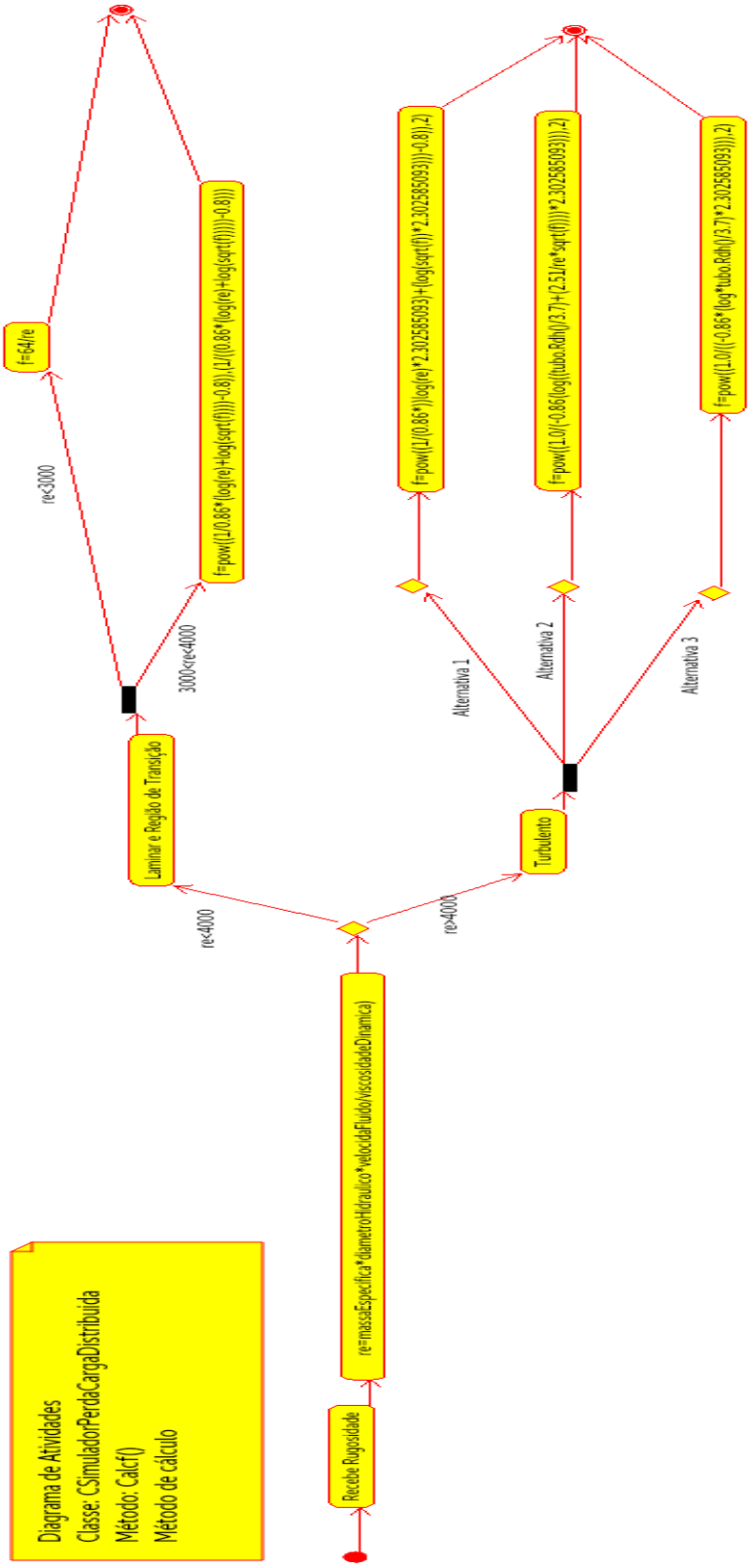

Figura 4.5: Diagrama de atividades

Capítulo 5

Projeto

Neste capítulo do projeto de engenharia veremos questões associadas ao projeto do sistema, incluindo protocolos, recursos, plataformas suportadas, implicações nos diagramas feitos anteriormente, diagramas de componentes e implantação. Na segunda parte revisamos os diagramas levando em conta as decisões do projeto do sistema.

5.1 Projeto do sistema

1. Protocolos

    - O software terá entrada de dados importando arquivos no formato ASCII com extensão .txt e receberá dados via teclado.
    - A interface utilizada será em modo texto.
    - O software irá gerar saída de arquivos no formato ASCII com extensão no .txt.

2. Recursos

    - Identificação e alocação dos recursos globais, como os recursos do sistema serão alocados, utilizados, compartilhados e liberados. Implicam modificações no diagrama de componentes.
    - Neste projeto o programa irá necessitar de utilizar os componentes internos do computador, como por exemplo HD, processador, mouse, teclado.
    - Neste projeto será utilizado uma tabela material vs rugosidade, como banco de dados para o software.
    - Neste projeto será utilizado duas tabelas para conversão de unidades, na qual uma contem unidades no sistema internacional (SI) e a segunda unidades mais utilizadas pelos usuários.
    - Neste projeto não se aplica sistemas de armazenamento em massa.
    - Os gráficos serão gerados no programa externo Gnuplot.

3. Controle

   - Neste projeto o controle será sequencial.
   - Não há necessidade de otimização. Pois, o software e seus componentes trabalham com dados pequenos.
   - Identificação e definição de *loops* de controle e das escalas de tempo.
     - Não se aplica.

4. Plataformas

   - O software irá funcionar nos sistemas operacionais Windows e GNU/Linux, sendo desenvolvido no GNU/Linux, Ubunto, e testado no Windows e GNU/Linux.
   - A linguagem de programação utilizada é C++.

5. Padrões de projeto

   - Normalmente os padrões de projeto são identificados e passam a fazer parte de uma biblioteca de padrões da empresa. Mas isto só ocorre após a realização de diversos projetos.
     - Não se aplica.

5.2 Projeto orientado a objeto – POO

Efeitos do projeto no modelo estrutural

- Neste projeto foram utilizadas as seguintes bibliotecas
  - `string`, para utilização de palavras.
  - `map`, para criar uma associação entre key e correspondente.
  - `vector`, para utilização de vetores contendo dados.
  - `fstream`, utilizado para gravar e ler arquivos de disco.
  - `<cmath>`, utilizada para realizar calculos que precisem de funções especificas.
  - `iostream`, para entrada e saida de dados, pelo teclado e tela respectivamente.
  - `iomanip`, para utilização de ferramentas de formatação do layout de tabelas.
- Novas classes e associações oriundas das bibliotecas selecionadas e da linguagem escolhida devem ser acrescentadas ao modelo.
  - Após a análise e o projeto do sistema surgiu a necessidade da criação de novas classes e associações. Problemas como esse poderão surgir durante a implementação do banco de dados, sendo assim passível de modicação ou criação de novas classes, atributos e métodos.

Efeitos do projeto nas associações

- Neste projeto o usuário acessa o arquivo material .txt oferecendo a chave, nome do material, e esse arquivo retorna o valor da rugosidade média do material oferecido.

5.3 Diagrama de componentes

Neste projeto temos as classes CTubo, CFluido, CConversaoUnidades, CSimulador-PerdaCargaDistribuida e geram objetos respectivos de cada classe.

A classe CConversaoUnidades depende dos objetos da classe CTubo e CFluido, da mesma forma que a classe CSimuladorPerdaCargaDistribuida depende dos objetos criados pela classe CConversaoUnidades.

O simulador é o que faz a junção entre os objetos da classe CTubo, CFluido e resultados matemáticos. Assim como está representado na Figura 5.1.

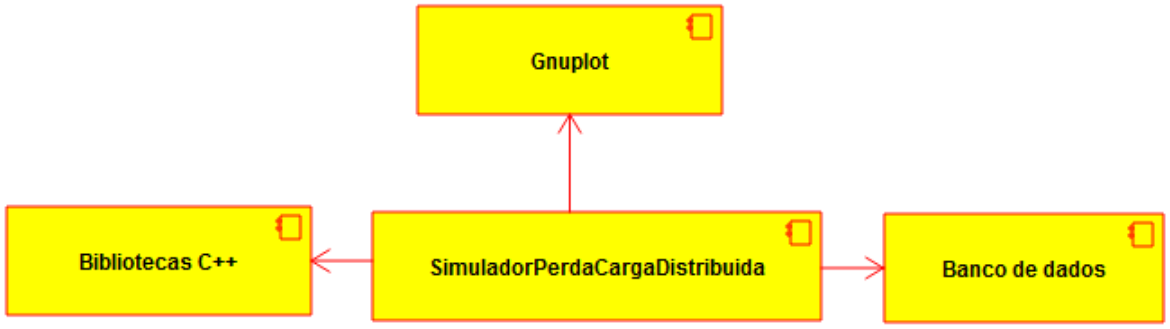

Figura 5.1: Diagrama de componentes

Capítulo 6

Implementação

Neste capítulo do projeto de engenharia apresentamos os códigos fonte que foram desenvolvidos.

6.1 Código fonte

Apresenta-se a seguir um conjunto de classes (arquivos .h e .cpp) além do programa `main`.

Apresenta-se na listagem 6.1 o arquivo com código da classe CConversaoUnidades.

Listing 6.1: Arquivo de cabeçalho da classe CConversaoUnidades.

```
1#ifndef CConversaoUnidades_h
2#define CConversaoUnidades_h
3#include <iostream>
4#include <iomanip>
5#include <fstream>
6#include <string>
7#include <vector>
8
9//Autor: Prof. Andre Duarte Bueno, Dr. Eng.
10class CConversaoUnidades
11{
12private : // Atributos
13std :: vector < std :: string > unidade; ///< vetor com unidades
14std :: vector < std :: string > si; ///< vetor com unidades si
15std :: vector < std :: string > un; ///< vetor com unidades un
16std :: vector < double > fatorConversao; ///< fatorConversao de SI para
    UN
17std::string s;
18double v;
19static bool conversaoPadraoUNSI; ///< A conversao padrao e de UNSI
20static std :: string novaUnidade; ///< Unidade da ultima conversao
    realizada
21public :
```

```
22 // Construtor vazio
23 CConversaoUnidades ( ) {LerDadosTabelaConversao ( "unidadesPrimarias.dat
     " );LerDadosTabelaConversao ( "unidadesSecundarias.dat" );}
24 // Construtor sobrecarregado
25 CConversaoUnidades ( std :: vector < std :: string > _unidade , std ::
     vector < std :: string > _si ,
26 std :: vector < std :: string > _un , std :: vector <double >
     _fatorConversao ): unidade(_unidade ), si( _si ), un(_un),
     fatorConversao ( _fatorConversao ) { }
27 // Construtor copia
28 CConversaoUnidades ( const CConversaoUnidades & obj): unidade(obj.
     unidade ), si(obj .si), un( obj .un), fatorConversao (obj .
     fatorConversao ) {}
29 // Destrutor
30 ~ CConversaoUnidades ( ) {};
31 // Retorna conversaoPadraoUNSI ( Get )
32 bool ConversaoPadraoUNSI () { return conversaoPadraoUNSI ; }
33 // Seta conversaoPadraoUNSI ( Set )
34 void ConversaoPadraoUNSI ( bool b) { conversaoPadraoUNSI = b; }
35 // Adiciona nos vetores : unidade , si , bg e fatorConversao a partir
     uma ifstream
36 void LerDadosTabelaConversao ( std :: istream & in ) {
37 std :: string s; double v;
38 in >> s; unidade.push_back(s); /// Adiciona no final do vetor
39 in >> s; si.push_back (s);
40 in >> s; un.push_back (s);
41 in >> v; fatorConversao.push_back (v);
42 }
43 // Adiciona nos vetores : unidade , si , bg e fatorConversao a partir
     arquivo disco
44 void LerDadosTabelaConversao (std :: string fileName ) {
45 std :: ifstream fin( fileName.c_str () );
46 if( fin.fail () )
47 std :: cerr << "␣Nome␣de␣arquivo␣invalido␣ou␣nao␣tem␣acesso␣leitura␣.\n"
     ;
48 while ( ! fin.eof () )
49 LerDadosTabelaConversao ( fin );
50 fin. close ();
51 }
52 // Modifica informacoes de conversao de uma dada posicao
53 void ModificarDadosTabelaConversao ( std :: istream & in , int posicao =
      -1)
54 {
55 if(posicao>=0 and posicao<unidade.size())
56 {
57 in >> unidade [ posicao ];
58 in >> si[ posicao ];
59 in >> un[ posicao ];
```

```
60 in >> fatorConversao [ posicao ];
61 }
62 else std :: cerr << "\nPosicaoo␣invalida␣!\n";
63 }
64 /** Retorna linha com informacoes : unidade unidadeSI unidadeBG
       fatorConversao */
65 void ModificarDadosTabelaConversao ( std :: ostream & out , int p= -1) {
66 if(p>=unidade.size() or p<0)
67 std :: cerr << "\n-->␣Posicao␣invalida␣!\n";
68 else
69 out << std :: setw (25) << unidade [p] << std :: setw (15) << si[p]
70 << std :: setw (15) << un[p] << std :: setw (15) << fatorConversao [p]
       << '\n';
71 }
72 /** Salva informacoes em disco */
73 void ModificarDadosTabelaConversao ( std :: string fileName ) {
74 std :: ofstream fout ( fileName . c_str () );
75 if( fout )
76 for( int p=0; p<unidade.size(); p++)
77 ModificarDadosTabelaConversao (fout ,p);
78 else std :: cerr << "␣Nome␣de␣Arquivo␣␣Invalido␣ou␣acesso␣negado␣.\n";
79 fout . close ();
80 }
81 /** Realiza conversao SI2BG */
82 double SIUN ( double valor , std :: string & unidadeSI ){
83 for( int p=0; p<unidade.size();p++)
84 if ( unidadeSI == si[p]){
85 novaUnidade = un[p];
86 return valor = valor / fatorConversao [p];
87 }
88 novaUnidade = unidadeSI ; // Mantem a unidade
89 return valor ;
90 }
91 /** Realiza conversao UNSI */
92 double UNSI ( double valor , std :: string & unidadeUN ){
93 for( int p=0; p<unidade.size(); p++)
94 if ( unidadeUN == un[p]) {
95 novaUnidade = si[p];
96 return valor = valor * fatorConversao [p];
97 }
98 novaUnidade = unidadeUN ; // Mant~A @m a unidade
99 return valor ;
100 }
101 /** Realiza inversao da unidade ; se for SI converte para BG e vice -
       versa */
102 double InverteUnidade ( double valor , std :: string & _unidade ){
103 for( int p=0; p<unidade.size();p++) {
104 if ( _unidade == si[p]){ novaUnidade = un[p];
```

```
105return valor /= fatorConversao [p]; // Converte para un
106 }
107if ( _unidade == un[p]){ novaUnidade = si[p];
108return valor *= fatorConversao [p]; // Converte para si
109}
110}
111novaUnidade = _unidade ; // Mantem  a unidade
112std :: cerr << "\a\a\a\nUnidade␣nao␣encontrada!␣Conversao␣nao␣foi␣
     realizada!\n";
113return valor ;
114}
115/** Retorna nova unidade */
116std :: string NovaUnidade () { return novaUnidade ; }
117// Operadores
118/** Realiza UNSI se conversaoPadraoUNSI = true ; usa sobrecarga operador
      ( como função membro ) */
119// Uso: double valorConvertido = objetoConversao ( valorAntesConversao )
     ;
120double operator ()( double _v , std :: string & _un ){
121return conversaoPadraoUNSI ? UNSI(_v , _un): SIUN (_v , _un);
122}
123/** Funcao amiga , envia para dipositivo de saida dados da tabela de
     convers~A o .*/
124friend std :: ostream & operator << ( std :: ostream & out ,
     CConversaoUnidades & conversor ){
125for ( int p=0; p<conversor.unidade.size(); p++ )
126conversor . ModificarDadosTabelaConversao (out ,p);
127return out;
128}
129/** Funcao amiga , recebe de dipositivo de entrada dados da tabela de
     conversao .*/
130friend std :: istream & operator >>( std :: istream & in ,
     CConversaoUnidades & conversor ){
131while ( ! in. eof () ) conversor . LerDadosTabelaConversao (in);
132return in;
133}
134};
135# endif
```

Apresenta-se na listagem 6.2 o arquivo de implementação da classe CConversaoUnidades.

Listing 6.2: Arquivo de implementação da classe CConversaoUnidades.

```
136#include "CConversaoUnidades.h"
137#include <string>
138std :: string CConversaoUnidades :: novaUnidade = "␣[␣nenhuma␣conversao␣
     realizada␣!]";
139bool CConversaoUnidades :: conversaoPadraoUNSI = true ;
```

Apresenta-se na listagem 6.3 o arquivo com código da classe CTubo.

Listing 6.3: Arquivo de cabeçalho da classe CTubo.

```
141#ifndef CTUBO_H
142#define CTUBO_H
143# include "CConversaoUnidades.h"
144#include <iostream>
145#include <fstream>
146#include <map>
147#include <string>
148#include <math.h>
149#include <cstdlib>
150#include <iomanip>
151#include <vector>
152class CTubo
153{
154protected:
155        std::map< std::string , double > tabelamaterial;
156        std::map< double , std::string > tabelarugosidade;
157        std::map<std::string,double> tabelarugosidaderelativa;
158        std::vector<double>vetrugosidaderelativa;
159        std::vector<double>vetorgraficorugosidade;
160        double rugosidade;
161        double diametroHidraulico;
162        std::string materialaux;
163        std::vector<std::string> vectmaterial;
164        std:: vector<double> vecrugosidade;
165        double areaTubo=1.00;
166
167
168        std::string nomeArquivo;
169        double rdh;
170        double l;
171
172public:
173
174  //Empty Constructor
175  CTubo(){Entrada(); areaTubo=DiametroHidraulico()*DiametroHidraulico()
       *3.141529/4; std::cout << "AREA TUBO: "<<areaTubo << '\n'; std::
       cout << "AREA HIDRAUKLCI: "<<DiametroHidraulico() << '\n';}
176
177  //Construtor de copia
178  CTubo(const CTubo& ob): vetrugosidaderelativa(ob.vetrugosidaderelativa
       ), vetorgraficorugosidade(ob.vetorgraficorugosidade), rugosidade(ob
       .rugosidade), diametroHidraulico(ob.diametroHidraulico),
       materialaux(ob.materialaux),vectmaterial(ob.vectmaterial),
       vecrugosidade(ob.vecrugosidade),nomeArquivo(ob.nomeArquivo),rdh(ob.
       rdh),l(ob.l){}
179
180  //Construtor sobrecarregado
```

```
181  CTubo( std::vector<double>_vetorgraficorugosidade, double _rugosidade,
          double _diametroHidraulico, std::string _materialaux, std::vector
        <std::string> _vectmaterial, double _vecrugosidade, std::string
        _nomeArquivo, double _rdh, double _l ): vetorgraficorugosidade(
        _vetorgraficorugosidade), rugosidade(_rugosidade),
        diametroHidraulico(_diametroHidraulico),materialaux(_materialaux),
        vectmaterial(_vectmaterial),vecrugosidade(_vecrugosidade),
        nomeArquivo(_nomeArquivo), rdh(_rdh),l(_l){}
182
183  //Empty Destructor
184  virtual ~CTubo(){}
185
186  void Entrada();
187
188  //Set the value of rdh
189  //void Rdh (double _rdh) {rdh=_rdh;}
190
191  double Rdh(){return rdh;}
192
193  //Get the value of rdh
194  double CRdh(){
195        for (int i=0; i<2; i++){
196           std::cin.get();
197        }
198        std::cout << "Digite␣o␣nome␣do␣material␣específico␣(sem␣o␣uso␣de
              ␣letras␣maiusculas␣ou␣espaços):\n";
199        std::string _material; std::getline(std::cin,_material);
200        for (auto& element: tabelarugosidaderelativa){
201              std::cout<<element.first<<element.second<<'\n';
202        }
203        std::cout << "O␣valor␣é␣" << tabelarugosidaderelativa[_material]
               << '\n';
204        rdh = tabelarugosidaderelativa[_material];
205  }
206
207  //Set the value of rugosidade
208  void Rugosidade (double _rugo) {rugosidade= _rugo;}
209
210  //Get the value of Rugosidade
211  double Rugosidade () {return rugosidade;}
212
213  //Set the value of DiametroHidraulico
214  void DiametroHidraulico (double _dh)   {diametroHidraulico = _dh;}
215
216  //Get the value of DiametroHidraulico
217  double DiametroHidraulico ()   {return diametroHidraulico;}
218
219  double AreaTubo() {return areaTubo;}
```

```
220
221   // Set the value of l
222   void L (double _l) {l=_l;}
223
224   //Get the value of L
225   double L() {return l;}
226
227   //Seta o map tabelamaterial
228   void Tabelamaterial (const std::map< std::string , double >&
          _tabelamaterial){tabelamaterial=_tabelamaterial;}
229   //retorna o map tabelamaterial
230   std::map< std::string , double > Tabelamaterial(){return
          tabelamaterial;}
231   //Seta o map tabelarugosidade
232   void Tabelarugosidade (std::map< double , std::string >
          _tabelarugosidade){tabelarugosidade=_tabelarugosidade;}
233   //Retorna o map mAbertura
234   std::map< double ,std::string > Tabelarugosidade( )const{return
          tabelarugosidade;}
235
236    //Dado rugosidade retorna material correspondente
237   std::string Tabelamaterial (double _tabelarugosidade){return
          tabelarugosidade[_tabelarugosidade];}
238   // Dada material retorna rugosidade correspondente
239   double Tabelarugosidade (std::string _tabelamaterial){ return
          tabelamaterial [ _tabelamaterial ]; }
240   //usuario cria tabela
241   void tabela ();
242   //Metodo para leitura de dados (l,diametro hidraulico)
243   double Comprimento();
244   //Metodo para leitura de dados (Diametro Hidraulico)
245   double Diametrohidraulico( );
246
247   //Sobrecarga de operadores
248   friend std::istream& operator>>(std::istream& is , CTubo& ob);
249   friend std::ostream& operator<<(std::ostream& out , CTubo& ob);
250   friend std::ifstream& operator>>(std::ifstream& is , CTubo& ob);
251   friend std::ofstream& operator<<(std::ofstream& out , CTubo& ob);
252};
253#endif
```

Apresenta-se na listagem 6.4 o arquivo de implementação da classe CTubo.

Listing 6.4: Arquivo de implementação da classe CTubo.

```
257#include "CTubo.h"
258
259//Entrada e conversao de Dados
260void CTubo::Entrada(){
261        Comprimento();
```

```
262         Diametrohidraulico();
263         tabela();
264         CRdh();
265 }
266
267 double CTubo::Comprimento ()
268 {
269   CConversaoUnidades conversor;
270   std::string unidade;
271   std::cout << "Entre␣com␣o␣comprimento␣do␣tubo␣e␣em␣seguida␣sua␣unidade
        ␣[Ex:␣35␣m].\n␣";
272   std::cin >> l;
273   std::cin >> unidade;
274   l = conversor.UNSI (l, unidade);
275   std::cout << "Transformando␣para␣SI␣----->␣" << l << "␣";
276   std::cout << conversor.NovaUnidade () << std::endl;
277   return l;
278 }
279 //Entrada e conversao de Dados
280 double CTubo::Diametrohidraulico ()
281 {
282   CConversaoUnidades conversor;
283   std::string unidade;
284   std::cout << "Entre␣com␣o␣valor␣do␣␣Diametro␣Hidraulico␣e␣em␣seguida␣
        sua␣unidade␣[Ex:␣2␣␣m].␣\n␣";
285   std::cin >> diametroHidraulico;
286   std::cin >> unidade;
287   diametroHidraulico = conversor.UNSI (diametroHidraulico, unidade);
288   std::cout << "Transformando␣para␣SI␣----->␣" << diametroHidraulico <<
        "␣";
289   std::cout << conversor.NovaUnidade () << std::endl;
290   return diametroHidraulico;
291 }
292
293 //Salva tabela nos maps
294 std::ifstream & operator>> (std::ifstream& in, CTubo& tabela)
295 {
296   CTubo graficorug;
297   tabela.tabelamaterial.clear ();
298   tabela.tabelarugosidade.clear ();
299   std::string material;
300   double rugosidadeabsoluta;
301   while (true)                    // le linhas e adiciona ao map
302     {
303       in >> material;
304       if (in.good ())
305         in >> rugosidadeabsoluta;
306       else
```

```
307         {
308           in.clear ();
309           return in;
310         }
311       if (in.good ())
312         {
313           tabela.tabelamaterial.
314             insert (make_pair (material, rugosidadeabsoluta));
315           tabela.tabelarugosidade.
316             insert (make_pair (rugosidadeabsoluta, material));
317         }
318       else
319         {
320           in.clear ();
321           return in;
322         }
323     }
324   return in;
325 }
326
327 //Constroi e Le a tabela do disco   MATERIAL - RUGOSIDADE
328 void CTubo::tabela ()
329 {
330   std::ifstream in;
331   std::cout <<"\nSe deseja criar uma nova tabela digite 1. \nCaso deseje
        utilizar o banco de dados digite 2.\n";
332   int alternativa;
333   std::cin >> alternativa; std::cin.get();
334
335  // Criacao de nova tabela pelo usuario
336   if (alternativa == 1)
337         {
338                 std::cout <<"\nEntre com o nome para nomear o novo banco
                      de dados.\nObs: Sem espacos ou caracteres especiais
                      .\n";
339                 std::cin >> nomeArquivo;
340                 std::cout<<"\nQuantos dados de cada elemento deseja
                      preencher a tabela?\n";
341                 int d;
342                 std::cin>>d;
343                 std::cout << "\nEntre os dados da Tabela   Material x
                      Rugosidade no seguinte formato: \nMaterial enter
                      Rugosidade enter.\n";
344                 //limpa vetores
345                 vectmaterial.clear();
346                 vecrugosidade.clear();
347                 //Recebe dados do teclado do usuario
348                 for(int i=0;i<d;i++)
```

```
349                 {
350                         std::cin >> materialaux;
351                         vectmaterial.push_back(materialaux);
352                         std::cin >> rugosidade;
353                         vecrugosidade.push_back(rugosidade);
354                 }
355                 //Apaga ultimo dado do vetor
356                 vectmaterial.pop_back () ;
357                 vecrugosidade.pop_back () ;
358                 //Mostra tabela criada pelo usuario na tela
359                 std::cout<<"\nA␣tabela␣criada␣e:␣\n\n";
360                 for(int i=0; i<d;i++)
361                 {
362                         std::cout<<vectmaterial[i]<<std::setw(30)<<
                            vecrugosidade[i]<<std::endl;
363                 }
364 //salva tabela em disco
365         std::ofstream fout (nomeArquivo.c_str ()) ;
366                 for( int i = 0; i < d; i ++)
367                 {
368                         fout << vectmaterial[i] << std::setw(15) <<
                            vecrugosidade[i] << std::endl;
369                 }
370         }
371
372
373 //Le tabela do banco de dados ja criada
374   if (alternativa == 2)
375         {
376                 std::ifstream in;
377                 //Verificacao nome do arquivo do banco de dados
378                 do
379                 {
380                     std::cout <<"\nEntre␣com␣nome␣do␣arquivo␣com␣dados␣
                        padroes␣dos␣materiais␣e␣rugosidades:␣";
381                     std::cin >> nomeArquivo;
382                     std::cin.get ();
383                     in.open (nomeArquivo.c_str ());
384                     if (!in)
385                         std::cout << "\nNome␣invalido,␣tente␣novamente.\
                            n";
386                 } while (!in);
387
388                 //Limpa vetores
389                 vectmaterial.clear();
390                 vecrugosidade.clear();
391                 //Enquanto nao chegou ao final da tabela continua
                    recebendo dados e salvando nos vetores
```

```
392                 while (!in.eof())
393                 {
394                         in>> materialaux;
395                         vectmaterial.push_back(materialaux);
396                         in>> rugosidade;
397                         vecrugosidade.push_back(rugosidade);
398                 }
399                 in.close () ;
400
401                 vectmaterial.pop_back () ; vecrugosidade. pop_back () ;
402
403                 //Mostra tabela carregada do disco na tela
404                 std::cout<<"\nA tabela carregada do disco e: \n\n";
405                 for(int i=0; i<vectmaterial.size(); i++) //ok
406                 {
407                         std::cout<<vectmaterial[i]<<std::setw(30)<<
                            vecrugosidade[i]<<std::endl;
408                 }
409                 vectmaterial.pop_back () ; vecrugosidade.pop_back () ;
410         }
411         for (int i=0; i<vectmaterial.size(); i++){
412                 tabelamaterial.insert(std::make_pair(vectmaterial[i],
                    vecrugosidade[i]));
413                 tabelarugosidade.insert(std::make_pair(vecrugosidade[i],
                     vectmaterial[i]));
414                 tabelarugosidaderelativa.insert(std::make_pair(
                    vectmaterial[i],vecrugosidade[i]/diametroHidraulico))
                    ;
415         }
416}
```

Apresenta-se na listagem 6.5 o arquivo com código da classe CFluido.

Listing 6.5: Arquivo de cabeçalho da classe CFluido.

```
418#ifndef CFluido_H
419#define CFluido_H
420# include "CConversaoUnidades.h"
421#include <iostream>
422#include <fstream>
423class CFluido
424{
425protected:
426//Atributos
427         double viscosidadeDinamica;
428         double viscosidadeCinematica;
429         double velocidadeFluido;
430         double massaEspecifica;
431
432public:
```

```
433  //Construtor vazio
434  CFluido() {
435         viscosidadeDinamica = VISCOSIDADEdinamica();
436         viscosidadeCinematica = VISCOSIDADEcinematica();
437         velocidadeFluido = VELOCIDADEfluido();
438         massaEspecifica = MASSAespecifica();
439  };
440
441  //Construtor de copia
442  CFluido(const CFluido & ob): viscosidadeDinamica(ob.
        viscosidadeDinamica), viscosidadeCinematica(ob.
        viscosidadeCinematica),velocidadeFluido(ob.velocidadeFluido),
        massaEspecifica(ob.massaEspecifica){};
443
444  //Destrutor
445  virtual~CFluido(){}
446
447  void Entrada();
448
449  //Set the value of ViscosidadeDinamica
450  void ViscosidadeDinamica (double _vd)   {viscosidadeDinamica = _vd;}
451
452  //Get the value of ViscosidadeDinamica
453  double ViscosidadeDinamica ()   {return viscosidadeDinamica;}
454
455  //Set the value of velocidadeFluido
456  void VelocidadeFluido (double _velocidadefluido) {velocidadeFluido =
        _velocidadefluido;}
457
458  //Get the valur of velocidadeFluido
459  double VelocidadeFluido () {return velocidadeFluido;}
460
461  //Set the value of MassaEspecifica
462  void MassaEspecifica (double _me)   {massaEspecifica = _me;}
463
464  //Get the value of MassaEspecifica
465  double MassaEspecifica ()   {return massaEspecifica;}
466
467  //Set the value of viscosidadeCinematica
468  void ViscosidadeCinematica (double _vc)  {viscosidadeCinematica = _vc
        ;}
469
470  //Get the value of viscosidadeCinematica
471  double ViscosidadeCinematica() { return viscosidadeCinematica;}
472
473  //Metodo para leitura de dados
474  double VISCOSIDADEcinematica ( );
475  double VELOCIDADEfluido ( );
```

```
476   double MASSAespecifica ( );
477   double VISCOSIDADEdinamica ( );
478
479   friend std::istream& operator>>(std::istream& is , CFluido& ob);
480
481   friend std::ostream& operator<<(std::ostream& out , CFluido& ob);
482 };
483 #endif
```

Apresenta-se na listagem 6.6 o arquivo de implementação da classe CFluido.

Listing 6.6: Arquivo de implementação da classe CFluido.

```
487 #include "CFluido.h"
488 using namespace std;
489
490 //Entrada e conversao de Dados
491 double CFluido::VISCOSIDADEdinamica()
492 {
493         CConversaoUnidades conversor ;
494         std :: string unidade ;
495         std::cout<<"Entre␣com␣o␣valor␣da␣Viscosidade␣Dinamica␣e␣em␣
              seguida␣sua␣unidade.␣[Ex:␣0.1␣N.s/m2]␣\n␣";
496         std::cin>>viscosidadeDinamica;
497         std::cin>>unidade;
498         viscosidadeDinamica=conversor.UNSI(viscosidadeDinamica,unidade);
499         std::cout<<"Transformando␣para␣SI␣----->␣"<<viscosidadeDinamica
              <<"␣";
500         std::cout<<conversor.NovaUnidade() <<std::endl;
501         return viscosidadeDinamica;
502 }
503 //Entrada e conversao de Dados
504 double CFluido::MASSAespecifica()
505 {
506         CConversaoUnidades conversor ;
507         std :: string unidade ;
508         std::cout<<"Entre␣com␣o␣valor␣da␣␣Massa␣Especifica␣do␣fluido␣e␣
              em␣seguida␣sua␣unidade␣[Ex:␣3␣g/cm3].␣\n␣";
509         std::cin>>massaEspecifica;
510         std::cin>>unidade;
511         massaEspecifica=conversor.UNSI(massaEspecifica,unidade);
512         std::cout<<"Transformando␣para␣SI␣----->␣"<<massaEspecifica<<"␣"
              ;
513         std::cout<<conversor.NovaUnidade() <<std::endl;
514         return massaEspecifica;
515 }
516 //Entrada e conversao de Dados
517 double CFluido::VELOCIDADEfluido()
518 {
519         CConversaoUnidades conversor ;
```

```
520         std :: string unidade ;
521         std::cout<<"Entre␣com␣o␣valor␣da␣Velocidade␣do␣Fluido␣e␣em␣
                seguida␣sua␣unidade␣[Ex:␣10␣m/s].␣␣\n␣";
522         std::cin>>velocidadeFluido;
523         std::cin>>unidade;
524         velocidadeFluido=conversor.UNSI(velocidadeFluido,unidade);
525         std::cout<<"Transformando␣para␣SI␣----->␣"<<velocidadeFluido<<"␣
                ";
526         std::cout<<conversor.NovaUnidade() <<std::endl;
527         return velocidadeFluido;
528 }
529 //Entrada e conversao de Dados
530 double CFluido::VISCOSIDADEcinematica()
531 {
532         CConversaoUnidades conversor ;
533         std :: string unidade ;
534         std::cout<<"Entre␣com␣o␣valor␣da␣Viscosidade␣Cinematica␣e␣em␣
                seguida␣sua␣unidade␣[Ex:␣10␣␣m2/s].\n␣";
535         std::cin>>viscosidadeCinematica;
536         std::cin>>unidade;
537         viscosidadeCinematica=conversor.UNSI(viscosidadeCinematica,
                unidade);
538         std::cout<<"Transformando␣para␣SI␣----->␣"<<
                viscosidadeCinematica<<"␣";
539         std::cout<<conversor.NovaUnidade() <<std::endl;
540         return viscosidadeCinematica;
541 }
```

Apresenta-se na listagem 6.7 o arquivo com código da classe CCalculoEscoamento.

Listing 6.7: Arquivo de cabeçalho da classe CSimuladorPerdaCargaDistribuida.

```
542 #ifndef CSIMULADORPERDACARGADISTRIBUIDA_H
543 #define CSIMULADORPERDACARGADISTRIBUIDA_H
544 #include <cstdlib>
545 #include <string>
546 #include <map>
547 #include <math.h>
548 #include <iostream>
549 #include <fstream>
550 #include <map>
551 #include <cmath>
552 # include "CConversaoUnidades.h"
553 #include "CTubo.h"
554 #include "CFluido.h"
555 #include "CGnuplot.h"
556
557 using namespace std;
558
559 ////////////////////////////////////
```

```
560 //Autores: Mateus Nascimento Gonçalves Rocha, Lucas Isaac Vieira
      Oliveira
561
562 class CSimuladorPerdaCargaDistribuida
563 {
564
565 protected:
566           //Atributos
567           double vazao=1.00;
568           double re=1.00;
569           double hf=0.00;
570           double f=1.00;
571       CConversaoUnidades* conversor;
572       CTubo* tubo;
573       CFluido* fluido;
574
575 public:
576
577 const double g=9.81;
578
579   //Empty Constructor
580   CSimuladorPerdaCargaDistribuida (){
581         conversor = new CConversaoUnidades();
582         cout << *conversor;
583         tubo = new CTubo();
584         fluido = new CFluido();
585   }
586
587   //Construtor de copia
588   //CSimuladorPerdaCargaDistribuida(CTubo& _tubo,CFluido& _fluido,const
        CSimuladorPerdaCargaDistribuida& ob): tubo(_tubo),fluido(_fluido),
        vazao(ob.vazao), re(ob.re), hf(ob.hf), f(ob.f),areaTubo(ob.areaTubo
        ){}
589
590   //Construtor sobrecarregado
591   //CSimuladorPerdaCargaDistribuida(CTubo& _tubo,CFluido& _fluido,
        double _vazao,  double _re, double _hf, double _f, double _g,
        double _areaTubo, double _diametrohidraulico ):tubo(_tubo),fluido(
        _fluido),vazao(_vazao), re(_re),hf(_hf), f(_f),areaTubo(_areaTubo)
        {}
592
593   //Empty Destructor
594   virtual ~CSimuladorPerdaCargaDistribuida (){}
595
596
597   //Metodo set
598   void Vazao (double _vaz)   {vazao = _vaz;}
599
```

```
600   //Metodo get
601   double Vazao ()   {return vazao;}
602
603   //Metodo set
604   void Re (double _re)   {re = _re;}
605
606   //Metodo get
607   double Re ()   {return re;}
608
609   //Metodo set
610   void Hf (double _hf)   {hf = _hf;}
611
612   //Metodo get
613   double Hf ()   {return hf;}
614
615   //Metodo set
616   void F (double _f)   {f = _f;}
617
618   //Metodo get
619   double F ()   {return f;}
620
621   //Metodos de calculo
622
623   double CalcVazao ( );
624
625   double CalcRe ( );
626
627   double Calcf  ();
628
629   double CalcHfQ ();
630
631   void Plotar();
632
633   //Sobrecarga de operador
634   friend std::ifstream& operator>>(std::ifstream& is ,
         CSimuladorPerdaCargaDistribuida& ob);
635   friend std::ofstream& operator<<(std::ofstream& out ,
         CSimuladorPerdaCargaDistribuida& ob);
636
637   void Simular();
638
639 };
640 #endif
```

Apresenta-se na listagem 6.8 o arquivo de implementação da classe CCalculoEscoamento.

Listing 6.8: Arquivo de implementação da classe CSimuladorPerdaCargaDistribuida.

```
641 #include "CSimuladorPerdaCargaDistribuida.h"
642
643 using namespace std;
644
645 void CSimuladorPerdaCargaDistribuida::Simular()
646 {
647         CalcVazao();
648         CalcRe();
649         Calcf();
650         CalcHfQ();
651         Plotar();
652 }
653
654 //Calculo a vazao
655 double CSimuladorPerdaCargaDistribuida::CalcVazao()
656 {
657         vazao=tubo->AreaTubo()*fluido->VelocidadeFluido();
658         return vazao;
659 }
660
661 //Calculo numero de Reynold
662 double CSimuladorPerdaCargaDistribuida::CalcRe()
663 {
664         re = ((tubo->DiametroHidraulico()* fluido->VelocidadeFluido())/
               fluido->ViscosidadeCinematica());
665         return re;
666 }
667
668 //Calculo fator de atrito
669 double CSimuladorPerdaCargaDistribuida::Calcf()
670 {
671                 std::cout<< "\nReynold vale: " << CalcRe() <<std::endl;
672
673                 if(re<=2000)
674   {
675                 f= 64/re;
676   }
677
678         if(re>2000 && re<3000)
679         {
680                 std::cout<<"\nNao e possivel calcular o fator de atrito
                    (f), pois esta na regiao de transicao. Aproximando
                    para Reynolds < 2000 temos:\n";
681                         f= 64/re;
682                 std::cout<<"\nO valor do fator de atrito e: "<<f<<std::
                    endl;
683                         return f;
```

```
684         }
685
686         if(re>=3000 && re<4000)
687         {
688                 std::cout<<"\nNao␣e␣possivel␣calcular␣o␣fator␣de␣atrito␣
                    (f),␣pois␣esta␣na␣regiao␣de␣transicao.␣␣Aproximando␣
                    para␣Reynolds␣>3000␣temos:\n";
689                     f= pow((1/((0.86 * (log (re) + log (sqrt(f))))-
                        0.8)),(1/((0.86 * (log (re) + log (sqrt(f))))
                        - 0.8)));
690                 std::cout<<"\nO␣valor␣do␣fator␣de␣atrito␣e:␣␣"<<f<<std::
                    endl;
691                     return f;
692         }
693
694   if( re>=4000)
695   {
696   std::cout<<"\nDado␣o␣numero␣de␣Reynolds:␣"<< re << "e␣a␣rugosidade␣
        relativa:␣"<< tubo->Rdh() <<"␣mm,␣qual␣e␣o␣tipo␣de␣escoamento?\
        nDigite:\n␣1-Para␣escoamento␣turbulento␣hidraulicamente␣liso.\n␣2-
        Para␣escoamento␣turbulento,␣com␣transicao␣entre␣hidraulicamente␣
        liso␣e␣rugoso.\n␣3-Para␣escoamento␣completamente␣turbulento␣e␣
        hidraulicamente␣rugoso.\nA␣escolha␣foi:␣";
697   int resposta;
698   std::cin>>resposta; std::cin.get();
699         // Alternativa 1
700         if(resposta==1){
701                 f= pow((1/((0.86 * ((log(re)*2.302585093) + (log (sqrt(f
                    ))*2.302585093)))- 0.8)), 2);
702         }
703     // Alternativa 2
704         else if(resposta==2){
705                 f = pow( (1.0/ ( -0.86* (log ( (tubo->Rdh()/3.7) + (2.51
                     / (re * sqrt(f)) ) ) *2.302585093))),2);
706         }
707         // Alternativa 3
708         else if(resposta==3){
709                         f= pow(( 1.0/ ( -0.86 * (log(tubo->Rdh() / 3.7)
                            *2.302585093) ) ), 2 );
710         }}
711
712   std::cout<<"\nO␣valor␣do␣fator␣de␣atrito␣e:␣␣"<<f<<std::endl;
713   return f;
714 }
715 //Calculo Perda de Carga Distribuida
716 //Mostra na tela Perda de Carga e Vazao
717 double CSimuladorPerdaCargaDistribuida::CalcHfQ( )
718 {
```

```
719         hf= (f * tubo->L() * pow(fluido->VelocidadeFluido(), 2)) / (tubo
                ->DiametroHidraulico() * 2 * g);
720         std::cout<<"\nA perda de carga no decorrer da tubulacao e de:  "
                << hf <<" m.\nA vazao do fluxo do fluido e de: "<<vazao<<" m3
                /s"<<std::endl;
721         return hf;
722 }
723
724
725 void CSimuladorPerdaCargaDistribuida::Plotar()
726 {
727         std::vector<double> fpvct;
728         std::vector<double> hfpvct;
729         double x;
730         for (float fi = 0.008; fi <0.1; fi+=0.0001){
731                 fpvct.push_back(fi);
732                 x = (fi * tubo->L() * pow(fluido->VelocidadeFluido(), 2)
                        ) / (tubo->DiametroHidraulico() * 2 * g);
733                 hfpvct.push_back(x);
734         }
735
736         Gnuplot g2d ("points"); // Construtor
737         g2d.Legend("inside").Legend("left").Legend("bottom").Legend("box
                ");
738     g2d.Title("Perda de carga Distribuida x Fator de Atrito"); // Titulo
            do grafico
739     g2d.XLabel("Fator de Atrito");     // Rotulo eixo x
740     g2d.YLabel("Perda de carga Distribuida");     // Rotulo eixo y
741     g2d.PlotVector(fpvct,hfpvct);// Plota vetores
742     cout << "Pressione Enter para sair\n";
743         cin.get();
744 }
```

Apresenta-se na listagem 6.9 o arquivo de implementação da classe main.

Listing 6.9: Arquivo de implementação da classe main.

```
745 #include <cstdlib>
746 #include <string>
747 #include <map>
748 #include <iostream>
749 #include <fstream>
750 #include <map>
751 #include <cmath>
752 #include "CConversaoUnidades.h"
753 #include "CTubo.h"
754 #include "CFluido.h"
755 #include "CSimuladorPerdaCargaDistribuida.h"
756 #include "CGnuplot.h"
757
```

```
758 using namespace std;
759
760
761 int main() {
762         //CCalculoEscoamento calculografico;
763         CSimuladorPerdaCargaDistribuida simulador;
764         char resposta = 's';
765         do {
766                 simulador.Simular();
767                 cout << "Deseja continuar? (s/n)\n";
768                 cin >> resposta;
769                 cout << "\n";
770         } while (resposta == 's' or resposta == 'S');
771         //calculografico.Plotar();
772         return 0;
773 }
```

Capítulo 7

Teste

Todo projeto de engenharia passa por uma etapa de testes. Neste capítulo apresentamos testes do software desenvolvido. Estes testes devem dar resposta aos diagramas de caso de uso inicialmente apresentados (diagramas de caso geral e específicos).

7.1 Teste 1: Teste entrada de dados, conversão de unidades e criação de tabela pelo usuário

Neste teste são testados a entrada de dados iniciais do usuário, a conversão de unidades de entrada de dados e a criação de tabela para um novo banco de dados. Estes valores são comparados com os encontrados na literatura e calculados numéricamente.

Etapas:

- Rodar programa.
- Entrar com o comprimento do Tubo.
- Entrar com o Diâmetro Hidráulico do Tubo.
- Escolher alternativa para criar um novo banco de dados.
- Escolher o nome do arquivo do novo banco de dados.
- Entrar com dados da nova tabela para compor o banco de dados.
- Verificar se o banco de dados foi criado.

```
[aluno@musgravito Listagens1]$ ./testeUM
     DiametroHidraulico          m          cm          0.01
                      L          m          cm          0.01
      VelocidadeFluido        m/s        cm/s          0.01
         MassaEspecifica      kg/m3      g/cm3         0.001
     ViscosidadeDinamica     N*s/m2   dina*s/cm2           0.1
  ViscosidadeCinematica       m2/s        cm2/s        0.0001
                                                     0.0001
              Material  Rugosidade    Acorebitado          4.05
           ferrofundido        0.26       concreto          1.35
                madeira      0.36ferrogalvanizado          0.15
           acocomercial       0.046        plastico             0
                  vidro         0.0             0.0             0
Entre com o comprimento do tubo e em seguida sua unidade [Ex: 35 m].
 1000 m
Transformando para SI -----> 1000 m
Entre com o valor do  Diametro Hidraulico e em seguida sua unidade [Ex: 2  m].
 45 cm
Transformando para SI -----> 0.45 m

Se deseja criar uma nova tabela digite 1.
Caso deseje utilizar o banco de dados digite 2.
1

Entre com o nome para nomear o novo banco de dados.
Obs: Sem espacos ou caracteres especiais.
MaterialxRugosidade

Quantos dados de cada elemento deseja preencher a tabela?
2

Entre os dados da Tabela  Material x Rugosidade no seguinte formato:
Material enter Rugosidade enter.
acocomercial
0.000046
ferrofundido
0.00023

A tabela criada e:

acocomercial                4.6e-05
ferrofundido                0.00023

Digite o nome do material correspondente (com o uso de letras maiusculas ou acentos):
```

Figura 7.1: Tela do programa mostrando teste 1

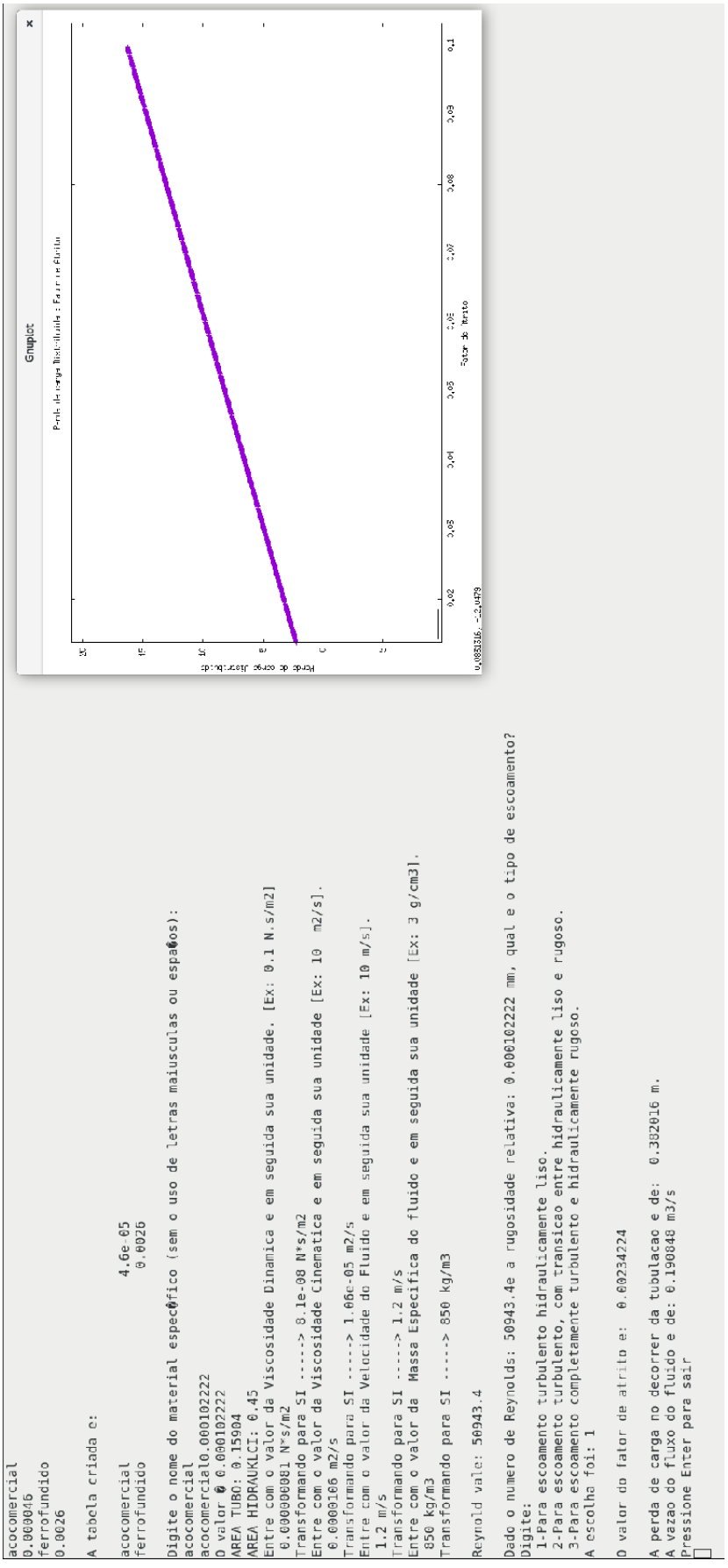

Figura 7.2: Tela do programa mostrando teste 1

7.2 Teste 2: Descrição

Neste teste verifica-se o funcionamento do map e a realização dos cálculos. Para sua realização, utilizamos o banco de dados criado ou o novo banco de dados criado pelo usuário. Após disso são realizados os cálculos e mostrado em tela. Como pode ser visualisado na Figura

- Entrar com o valor de viscosidade dinâmica.
- Entrar com o valor de massa específica do fluido. Entrar com o valor da velocidade do fluido.
- Entrar com o valor da viscosidade cinemática.
- De acordo com os cálculos é formado o número de Reynold que se relaciona com o tipo de escoamento e fator de atrito.
- É mostrado em tela o resultado dos cálculos de perda de carga no decorrer da tubulação e a vazão do fluxo do fluido.

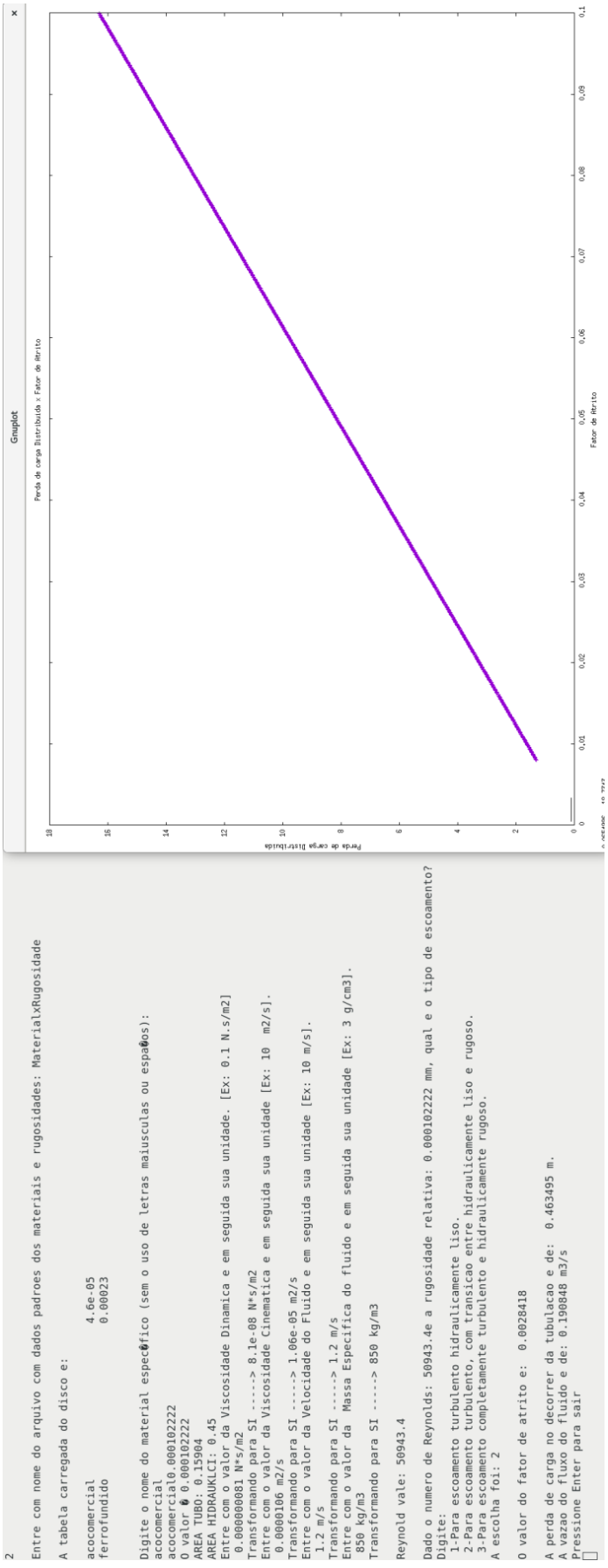

Figura 7.3: Tela do programa mostrando teste 2

Capítulo 8

Documentação

Todo projeto de engenharia precisa ser bem documentado. Neste sentido, apresenta-se neste capítulo a documentação de uso do software "Cálculo da perda de carga distribuída de um fluido no decorrer do escoamento". Esta documentação tem o formato de uma apostila que explica passo a passo como usar o software.

8.1 Documentação do usuario

Descreve-se aqui o manual do usuário, um guia que explica, passo a passo a forma de instalação e uso do software desenvolvido.

8.1.1 Como rodar o software

Para usar o software pela interface de tecto compile o arquivo CGnuplot.cpp, CConversaoUnidades.cpp, CFluido.cpp, CTubo.cpp, CSimuladorPerdaCargaDistribuida.cpp, main.cpp e rode o programa.

8.2 Documentação para desenvolvedor

Apresenta-se nesta seção a documentação para o desenvolvedor, isto é, informações para usuários que queiram modificar, aperfeiçoar ou ampliar este software.

8.2.1 Dependências

Para compilar o software é necessário atender as seguintes dependências:

- Instalar o compilador g++ da GNU disponível em http://gcc.gnu.org.
- É necessário também que se tenha instalado algum editor de texto para a visualização dos arquivos de disco gerados pelo software.
- Gnuplot e biblioteca Gnuplot.

Capítulo 9

Bibliográfia

Brutetti, F. Mecânica dos Fluidos. Editora Pearson Prentice Hall, 2008.

S.R.José, e M.S.Valdo, Apostila de mecânica dos fluidos para o curso de engenharia de exploração e produção de petróleo, 2016.

Bueno, A.D, Apostila Disciplina Programação Orientada a Objeto com C++ e Programação Prática, 2015.

Índice Remissivo