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

PROJETO DE ENGENHARIA
DESENVOLVIMENTO DO SOFTWARE
Simulador de Propriedades Mecânicas de Material Submetido ao Processo de Corrosão por Dióxido de Carbono - VERSÃO 1.0
TRABALHO DA DISCIPLINA PROGRAMAÇÃO PRÁTICA

THAYNÁ ANGELO DOS REIS
prof.André Duarte Bueno

MACAÉ - RJ
JANEIRO - 2016

Sumário

Capítulo 1

Introdução

Neste trabalho desenvolve-se um programa multiplataforma com o intuito de auxiliar alunos/pesquisadores a avaliar o comportamento corrosivo do sistema aço carbono/gás carbônico. O programa se baseia em uma leitura de dados, onde os resultados obtidos após o processamento permitem ao usuário obter dados fundamentais como: módulo de elasticidade, tenacidade à fratura e taxa de corrosão, tanto na forma numérica como por meio de gráficos.

1.1 Escopo do problema

A corrosão permanece como um dos maiores obstáculos operacionais para o sucesso da produção de hidrocarbonetos e o seu gerenciamento e controle são necessários para definir o custo efetivo dos projetos, bem como as condições operacionais mais seguras (KERMANI et al, 2003).

As falhas por corrosão, em sua maioria relacionadas à corrosão por dióxido de carbono têm sido reportados como a causa de 25% dos incidentes de segurança, 2,8% da rotatividade dos funcionários, 8,5% aumento nos gastos de capital, 5% de perda de produção de óleo. Logo, sensores que possam detectar a corrosão interna e externa, em tempo real, antes que as falhas ocorram, permitirão maior confiabilidade e segurança nas tubulações de gás e óleo.

A indústria continua a usar, extensivamente, o aço carbono e aços de baixa liga, os quais estão disponíveis em grandes volumes e satisfazem os requisitos mecânicos, estruturais, de fabricação e de custos. Do ponto de vista do custo dos materiais, o uso de aço carbono e aços de baixa liga em tubulações é a opção mais conveniente. Entretanto, um problema sério é a sua vulnerabilidade à corrosão por CO_2.

A corrosão em campos de produção de petróleo manifesta-se de diversas formas, dentre as quais a corrosão por CO_2 e a corrosão por H_2S nos fluidos produzidos e a corrosão por oxigênio na água de injeção dos sistemas, são as formas mais comuns de ataques encontrados na produção de óleo e gás. A maior parte das falhas em campos petrolíferos

resulta de corrosão por CO_2 do carbono e de aços de baixa liga devida a baixa capacidade de resistência desses aços a esse tipo de ataque. Portanto, o entendimento, predição e controle da corrosão por CO_2, são necessários ao projeto, operação e segurança dos campos petrolíferos .

A presença de gás carbônico é um fator crítico nos reservatório do pré-sal. Até agora, os poços testados na região indicam a presença de teores altos de CO_2. Em alguns casos, ultrapassam 30%. Os campos fora do pré-sal, têm em média 3%. Há muito interesse na compreensão do efeito de diferentes fatores no mecanismo da corrosão por CO_2, pois eles determinam a taxa de corrosão resultante.

Entretanto, há um grande número de variáveis envolvidas. A corrosividade é uma função de diversos fatores como a química da água, a velocidade do fluido, o conteúdo de CO_2, a temperatura dentre outros (LOPEZ et al, 2003).

Estudos recentes mostraram que, embora tenha se passado mais de quatro décadas de pesquisa, o entendimento da corrosão por CO_2 continua incompleto. Os modelos quantitativos existentes predizem de maneira irreal a taxa de corrosão do aço carbono e aços de baixa liga frente à corrosão por CO_2, o que resulta em uma super especificação de materiais e de impactos no custo da produção de óleo e gás (KERMANI et al, 2003).

A detecção da degradação dos materiais devido à corrosão pelas técnicas de inspeção contínua necessita do monitoramento em tempo real dos sistemas. Alguns métodos, tais como o monitoramento das propriedades mecânicas, são de relevante importância na maioria das aplicações para as quais se destinam os materiais. As mais importantes propriedades mecânicas são o módulo de elasticidade, a dureza, a ductilidade e a rigidez, decisivas no desempenho de máquinas e dispositivos submetidos a trabalho mecânico.

1.2 Objetivos

Os objetivos deste trabalho são:

- Objetivo geral:
  - Desenvolver um programa que receba dados reais ou arbitrários e calcule as propriedades mecânicas do material a partir de onde serão gerados gráficos que poderão ser utilizados para análise de taxas corrosivas.
- Objetivos específicos:
  - Permitir ao usuário entrar com dados característicos do material.
  - Disponibilizar métodos de entrada de dados a partir de arquivos do tipo texto.
  - Avaliar o comportamento corrosivo do sistema aço carbono/gás carbônico.

– Determinar as propriedades mecânicas mais importantes para a caracterização da superfície de um material: módulo de elasticidade, dureza, força de ligação de matriz de membrana e tenacidade à fratura de escalas de corrosão de CO_2 formados às diferentes temperaturas e na determinação das correspondentes taxas de corrosão.

Capítulo 2

Especificação

Apresenta-se neste capítulo a concepção, a especificação do sistema a ser modelado e desenvolvido.

2.1 Especificação do software - requisitos

Construir um software para fins acadêmicos que será livre, licença GNU/GPL v2.0, multiplataforma e orientado a objeto em C++, de modo a facilitar futuras modificações. Este será em modo texto, pois há pouca interação do usuário com o software. Inicialmente, o usuário definirá os parâmetros de entrada. Estes parâmetros serão inseridos manualmente via teclado ou pela leitura de um arquivo de texto (formato ASCII) . Usando essas informações o software irá calcular o módulo elástico, tenacidade à fratura e a taxa de corrosão.

O software apresentará ao usuário valores fundamentais para avaliação de taxas corrosivas do material em função das suas propriedades mecânicas.

Os resultados serão exibidos na forma de gráficos usando um software externo e impressos na tela. Se desejado pelo usuário, irá salvar os resultados em um disco no formato de texto (ASCII).

2.1.1 Definições da interface

O software tem interface de entrada em modo texto, e saídas em modo texto e gráfica .

2.1.2 Entrada de dados

A entrada dos valores de carga [N], comprimento L [mm] e temperatura [°C] pode ser feita de forma manual onde o usuário entra com os dados um a um ou pode ser feita através da leitura de um arquivo com os valores de carga [N] e comprimento L [mm] para cada temperatura [°C] .

O arquivo de texto a ser lido tem o seguinte formato:

In 2.1: Arquivo Teste

7330 50.851 50
15100 50.902 60
30400 51.003 70
41300 51.816 100
44800 52.832 110
46200 53.848 120
47500 54880 130

2.1.3 Saída de dados

Através dos dados acima será calculado a tensão [N/mm²] e deformação [mm]. Os valores de módulo elástico [GPa], tenacidade à fratura [MPa] e taxa corrosiva [mm/y] são apresentadas na mesma janela onde está sendo executado o programa. Também os gráficos: módulo elástico x temperatura, tenacidade à fratura x temperatura e taxa corrosiva x temperatura.

2.1.4 Nome do produto e componentes

Nome	Simulador para a geração de propriedades mecânicas do material submetido ao processo de corrosão por dióxido de carbono.
Componentes principais	Gerar um modelo sintético para caracterização e análise do componente corrosivo.
Missão	Auxiliar geoquímicos e engenheiros a avaliar as propriedades mecânicas de materias, tal como aço carbono e possibilitar comparação com aqueles que sofreram processo corrosivo .

2.1.5 Requisitos funcionais

RF-01	O usuário deverá ter liberdade para escolher o tipo de entrada de dados .

RF-02	Deve permitir o carregamento de arquivos criados pelo software.

RF-03	Deve permitir a escolha da propriedade mecânica.

RF-04	O usuário terá disponível os resultados em gráficos e em arquivos de texto.

2.1.6 Requisitos não funcionais

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

2.2 Casos de uso do software

Um *caso de uso* descreve um ou mais cenários de uso de software, exemplos de uso, como o sistema interage com usuários externos (atores). Ademais, ele deve representar uma sequência típica de uso do programa (a execucação de determinadas tarefas padrão). Também deve representar as exceções, casos em que o usuário comete algum erro, em que o sistema não consegue realizar as tarefas solicitadas.

A Tabela 2.1 mostra os itens incluídos na descrição do caso de uso. Uma vez apresentados os conceitos relacionados, apresentaremos como modelar os cenários com os diagramas de caso de uso da UML, na próxima subseção. Um caso de uso deve produzir uma descrição clara e não ambígua de como o usuário final e o sistema interagem um com o outro.

Tabela 2.1: Caso de uso

Nome do caso de uso:	Obtenção de valores fundamentais das propriedades mecânicas
Resumo/ descrição:	Geração das curvas propriedades mecânicas em função da temperatura.
Etapas:	1. Criar objeto Simulador.
	2. Criar objeto de Propriedade Mecânica.
	3. Calcular módulo elástico.
	4. Calcular tenacidade á fratura.
	5. Calcular taxa de corrosão.
	6. Gerar gráfico.
	7. Analisar resultado.
Cenários Alternativos:	Inserir dados fora da realidade do sistema simulado.

2.3 Diagrama de caso de uso geral do software

O diagrama de caso de uso geral da Figura 2.1 mostra o usuário calculando uma equação ou analisando resultados.

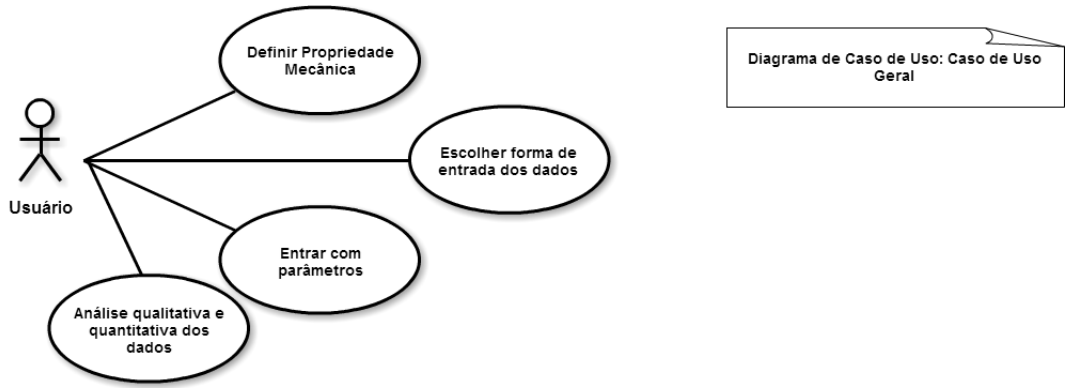


Figura 2.1: Diagrama de caso de uso – Caso de uso geral

2.4 Diagrama de caso de uso específico do software

O caso de uso a seguir, intitulado de caso de uso específico, é descrito na Figura 2.1, na Tabela 2.1 e é detalhado na Figura 2.2. O usuário criará um objeto Propriedade Mecânica, um objeto para cada tipo de propriedade mecânica possível; em seguida, definirá valores de temperatura, assim como parâmetros acessórios, simulará os dados e, por fim, os resultados (eventualmente gerará gráficos com os resultados obtidos utilizando um sistema externo, como o programa *gnuplot, scilab ou octave).*

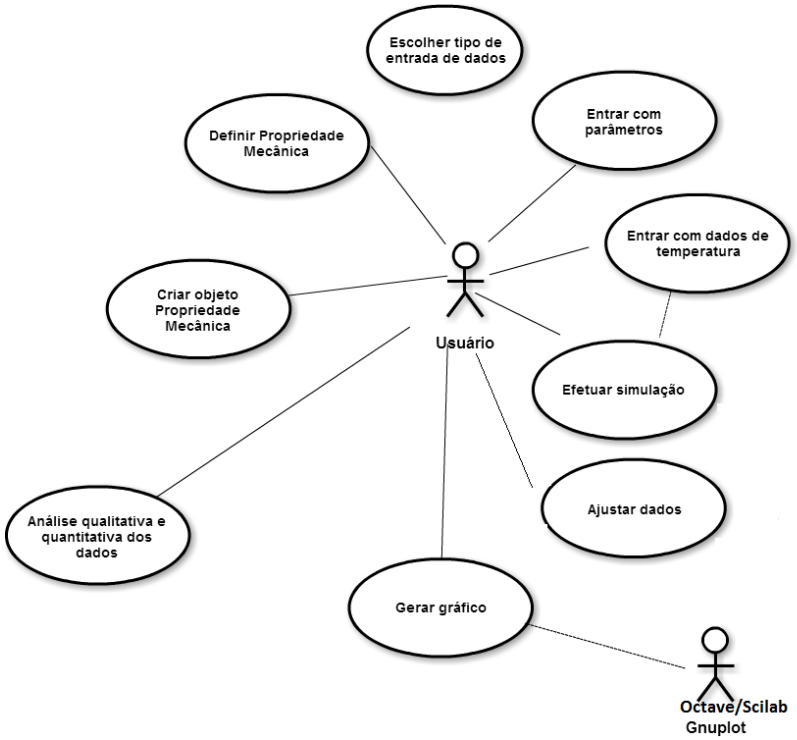


Figura 2.2: Diagrama de caso de uso específico - Detalhamento do Caso de uso geral

Capítulo 3

Elaboração

Neste capítulo iremos apresentar o processo que levou a criação do software, isto inclui atividades como: entrevistas, pesquisas bibliográficas e elaboração de diagramas de pacote.

3.1 Análise de domínio

A análise de domínio é uma parte da elaboração; seu objetivo é entender o domínio, a abrangência do sistema a ser desenvolvido. Envolve itens como estimar o reuso do sistema. Neste ponto, o analista pensa no sistema de uma forma mais genérica, identificando conceitos fundamentais que podem ser reaproveitados em outros sistemas.

- Estudo dos requisitos/especificações do sistema.
- Definição e caracterização do domínio a ser investigado (área dos programas e bibliotecas a serem investigadas).
- Manter contato com especialistas que tenham domínio dos conceitos envolvidos.
- Deve-se fazer entrevistas com os usuários do sistema .
- Identificação de amostras .
- Identificação e definição dos objetos genéricos .

3.2 Formulação teórica

Apresentaremos nesta seção o embasamento teórico para a criação e execução deste projeto de engenharia.

3.2.1 Propriedades mecânicas

A determinação e conhecimento das propriedades mecânicas é importante para a escolha do material para uma determinada aplicação, como exemplo o projeto e fabricação dos oleodutos. As propriedades mecânicas definem o comportamento do material quando sujeito à esforços mecânicos, pois estas estão relacionadas à capacidade do material de resistir ou transmitir estes esforços sem romper e sem se deformar de forma incontrolável.

As propriedades mecânicas são de relevante importância na maioria das aplicações para as quais se destinam os materiais. Algmas propriedades mecânicas importantes são o módulo de elasticidade, a dureza, a ductilidade e a rigidez.

Neste trabalho serão modelados as propriedades mecânicas mais decisivas no desempenho de máquinas e dispositivos submetidos a trabalho mecânico para a caracterização da superfície de um material: módulo de elasticidade e tenacidade à fratura.

3.2.2 Módulo de elasticidade

Há três maneiras de se aplicar uma carga sobre um sólido: tensão, compressão e cisalhamento. Um teste mecânico bastante comum é o teste de tensão, onde o material em estudo, na forma cilíndrica, é deformado até a fratura em um equipamento de tipo "Instron" que aplica uma força de tensão longitudinal de forma que a taxa de enlongação obtida seja constante no tempo. A carga aplicada é medida por uma célula de carga e a elongação por um extensômetro (CALLISTER JR,1997).

Durante a realização do teste, para uma determinada força aplicada F[N], as ligações internas são rompidas, e o material se alonga permanentemente. A tendência ao rompimento é oposta por reações internas, chamadas de tensões (MEYERS; CHAWLA,1999).

Geralmente, usa-se normas técnicas para o procedimento das medidas e confecção do corpo de prova para garantir que os resultados sejam comparáveis. As normas técnicas mais comuns são elaboradas pelas:

- ASTM (*American Society for Testing and Materials*).
- *ABNT (Associação Brasileira de Normas Técnicas).*

Naturalmente, a intensidade do conjunto dessas tensões depende da geometria da amostra: se a área da seção transversal A [mm^2] for dobrada , a resposta interna será duas vezes maior. Desse modo, defini-se tensão normal $\sigma_n$como a resistência por unidade de área (MEYERS; CHAWLA, 1999):

$$\sigma_n = \frac{F}{A} \tag{3.1}$$

Com um aumento dF na carga aplicada, a amostra sofre uma elongação dl em que o aumento normalizado do comprimento será:

$$d\varepsilon_n = \frac{dl}{l} \tag{3.2}$$

Integrando a equação 2, tem-se:

$$d\varepsilon_n =_{l_0} \int^{l_1} \frac{dl}{l} = ln\frac{l_1}{l_0} \tag{3.3}$$

onde l_0[mm] é o comprimento inicial e l_1[mm] é a deformação verdadeira longitudinal.

Para a maioria das aplicações, como o teste de tensão, é utilizada uma definição mais simples, a deformação que decorre da equação 1:

$$\varepsilon = \frac{l_1 - l_0}{l_0} \tag{3.4}$$

Da mesma forma, define-se a tensão [N/mm^2] :

$$\sigma = \frac{F}{A_0} \tag{3.5}$$

onde A_0 [mm^2] é a área original da seção transversal, antes do material se deformar permanentemente. Para pequenas deformações, os valores verdadeiros e de engenharia são aproximadamente os mesmos. Por convenção, tensões e deformações trativas são positivas, e tensões e deformações compressivas são negativas (MEYERS; CHAWLA, 1999). A relação entre tensão e deformação foi demonstrada experimentalmente em 1678 por Robert Hooke (MEYERS; CHAWLA, 1999):

$$E = \frac{\sigma}{\varepsilon} \tag{3.6}$$

A temperatura influencia nas propriedades mecânicas do material, como ilustrado na figura 3.1. O aumento da temperatura ↑provoca:

- ↓Módulo elástico
- ↓Tensão
- ↑Ductibilidade

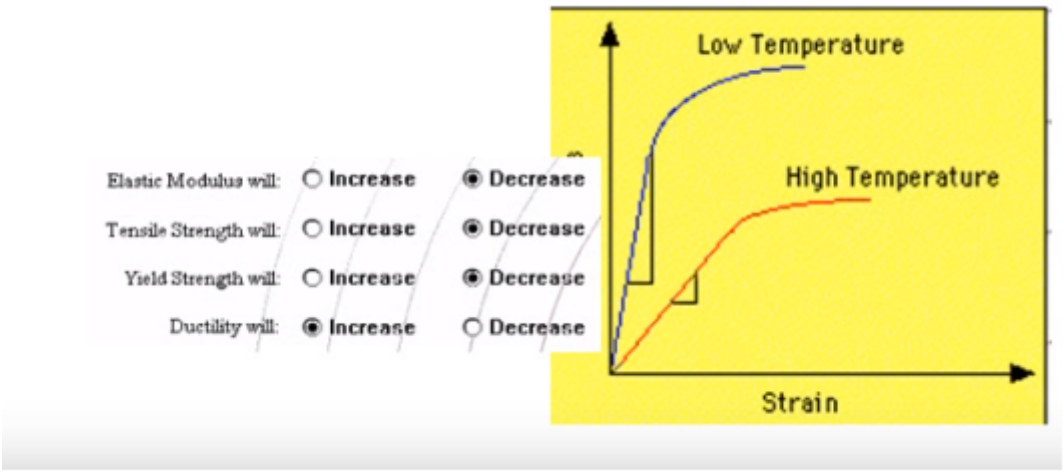


Figura 3.1: Carga de carregamento e descarregamento em função da profundidade

A figura 3.2 e 3.3 mostram o comportamento do módulo elástico e tenacidade à fratura em função da variação da temperaura respectivamente. Válido ressaltar, que é muito comum a formação de uma fina camada chamada de *película protetora*. Essa película pode retardar o processo corrosivo a medida que aumenta a temperatura, por isso que a partir de 100°C, momento que ocorre a formação da película , as propriedades mecânicas, módulo elástico e tenacidade à fratura aumentam em função do aumento da temperatura, como ilustram as figuras 3.2 e 3.3.

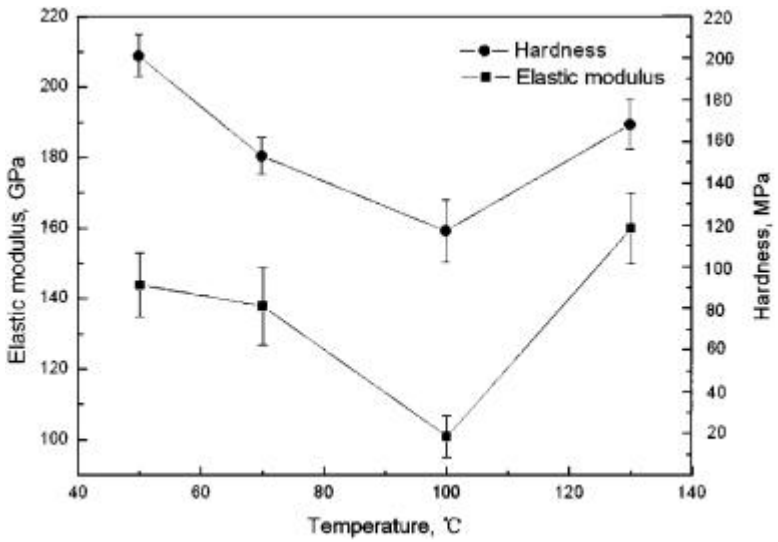


Figura 3.2: Módulo Elástico e dureza de escala corrosiva de CO_2 formados em diferentes temperaturas

3.2.3 Tenacidade à fratura

A resistência à fratura foi medida em K_{IC} [MPa $m^{1/2}$] em seções transversais polidas de escala corrosiva por meio de ensaios de microdureza Vickers, o qual foi calculado conforme a equação:

$$K_{IC} = \frac{0,294d(EP)^{1/2}}{c^{3/2}} \tag{3.7}$$

Onde:
α= 0,04
E = módulo de elasticidade [GPa]
H = dureza [GPa]
P = carga aplicada [N]
$c^{3/2}$= total do comprimento da fenda [m]
d = comprimento diagonal de indentação [m]

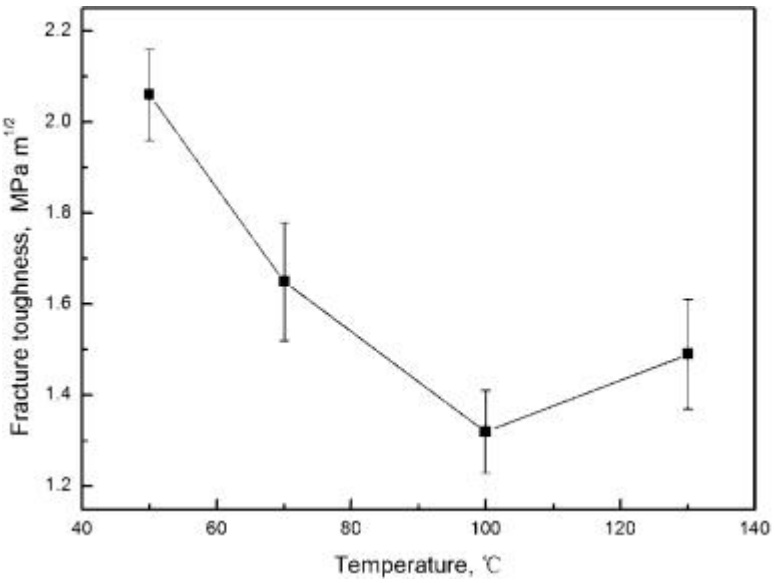


Figura 3.3: Tenacidade à fratura de escala corrosiva de CO_2formados em diferentes temperaturas

3.2.4 Taxa de corrosão

O aço de carbono é utilizado como material em instalações petrolíferas, no entanto ele é suscetível à corrosão em ambientes contendo CO_2, podendo levar a graves danos. A temperatura é um dos fatores que influenciam em ambientes de corrosão de CO_2. Em geral a temperatura desempenha um efeito sobre os valores da taxa de corrosão e da taxa da camada do produto de corrosão. Devido a existência de CO_2 em alguns campos de petróleo, estudos tem sido realizados a respeito do teor de CO_2 e alta concentração de cloreto. Este trabalho visou estudar a corrosão do aço carbono em solução saturada por CO_2 em diferentes temperaturas.

A avaliação da taxa de corrosão é feita através do ensaio de perda de massa que avalia a perda de massa do material após sua inversão no meio estudado. Os corpos de prova precisam ser pesados, antes e após o ensaio. Após cada período de realização do ensaio,

os corpos de prova foram colocados numa solução para remoção dos produtos de corrosão e pesados novamente. A taxa de corrosão foi então calculada.

$$CR = \frac{(W_0 - W_1) * 1000 * 365 * 24}{t * \rho * S} \tag{3.8}$$

Sendo:
W_0 = peso original da espécie [g]
W_1 = peso final da espécie [g]
t = tempo de imersão [h]
ρ= densidade do aço [7,8 x 10 $^{-3}$g mm^{-3}] é usado no presente trabalho
S = superfície exposta da espécie [mm^2]

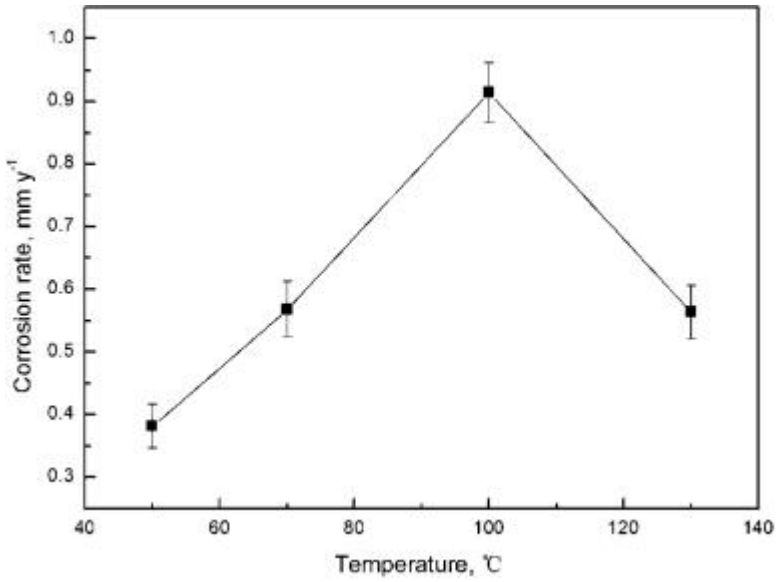


Figura 3.4: Valores da taxa de corrosão do ensaio de perda de massa em diferentes temperaturas.

3.3 Identificação de pacotes

Após estudo dos requisitos/especificações do sistema, algumas entrevistas com profissionais acadêmicos como a professora Georgiana Feitosa, estudos na biblioteca do LENEP e disciplinas do curso foi possível identificar nosso domínio de trabalho:

- Pacote Propriedade Mecânica:
  - Descrição: é o pacote formado pelos conceitos de propriedades mecânicas pertinentes ao assunto, sendo preponderante para designar o tipo de modelo de cálculo a ser aplicado, influenciando diretamente nos resultados obtidos pelos demais componentes envolvidos;

  - Está intimamente ligada com o problema , visto que o mesmo faz uso dos parâmetros para estimar as propriedades mecânicas, de amostras submetidas a temperaturas diferentes. Os valores encontrados são determinantes para que possam ser tomadas decisões sobre o futuro da operação de produção.

- Pacote Biblioteca Numérica:

  - Descrição: é o pacote composto por assuntos ligados a matemática geral. Este pacote é responsável pela concepção final do melhor ajuste para os dados de entrada, referente ao cálculo de propriedades mecânicas mais importanes para a caracterização da superfície de um material formados às diferentes temperaturas e na determinação das correspondentes taxas corrosivas. Este pacote tem uma ligação direta com o pacote Propriedade Mecânica, pois do mesmo ele recebe os parâmetros necessários para executar suas tarefas, aonde encontram-se algumas definições como:tenacidade à fratura e taxa de corrosão respectivamente:

$$K_{IC} = \frac{0,294d(EP)^{1/2}}{c^{3/2}} \tag{3.9}$$

$$CR = \frac{(W_0 - W_1) * 1000 * 365 * 24}{t * \rho * S} \tag{3.10}$$

- Pacote Gráfico

  - Descrição: é o pacote formado pelo *Gnuplot*. Este pacote é relacionados aos demais pois o mesmo recebe os resultados de ajuste propiciados pelos pacotes Biblioteca Numérica, apresentando-os de forma distribuída em eixos coordenados, de taxa de corrosão por tempo decorrido.

- Pacote de Simulação

  - Descrição: seu objetivo é o de repassar ao pacote Biblioteca Numérica os resultados, para que o mesmo faça os devidos ajustes. Esses resultados enviados são fruto do tratamento dos parâmetros de entrada do programa, ou seja, temperaturas, tempos e taxas relacionadas.
  - Responsável por fazer os ajustes necessários, propiciandoa saída dos resultados.

3.4 Diagrama de pacotes - assuntos

Nesta seção, apresentamos o diagrama de pacotes na Figura 3.3. Percebemos a grande interação entre esses componentes, reafirmando o que foi sicutido na Seção 3.3. Neste diagrama, temos os seguintes componentes, representados em UML 2.0:

- Pacote Propriedade Mecânica;

- Pacote Biblioteca Numérica;
- Pacote de Simulação;
- Pacote Gráfico (Gnuplot).

Logo abaixo temos o diagrama de pacotes para elaboração do software, como descrito na análise de domínio.

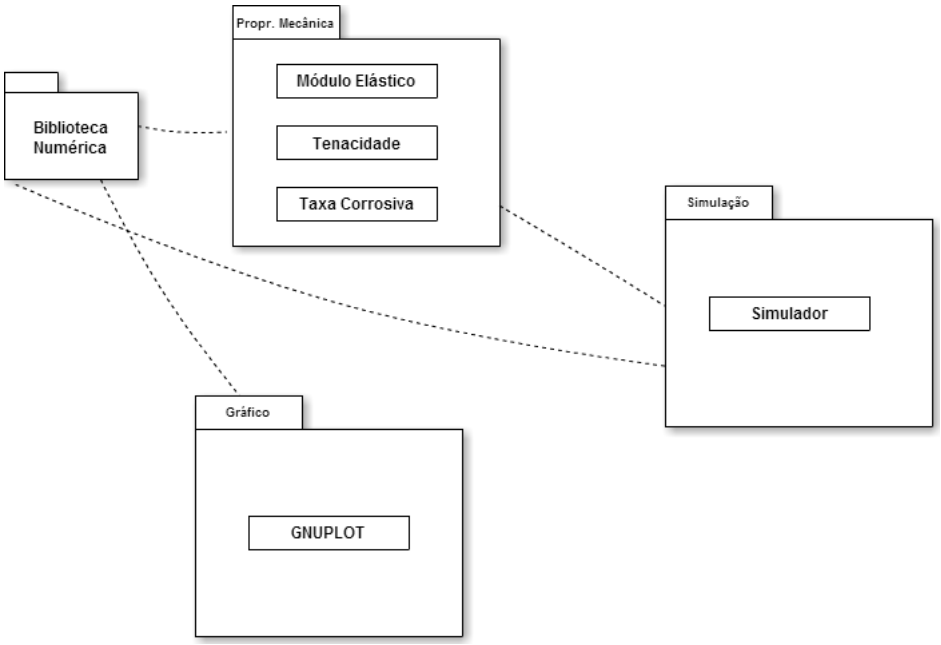


Figura 3.5: Diagrama de pacotes

Capítulo 4

AOO – Análise Orientada a Objeto

A terceira etapa do desenvolvimento de um sistema é a AOO - Análise Orientada a Objeto. A AOO utiliza algumas regras para identificar os objetos de interesse, as relações entre os pacotes, as classes, os atributos, os métodos, as heranças, as associações, 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 análide 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.

4.1.1 Dicionário de classes

A seguir, apresentamos um guia com definições para cada atributo e método de cada classe.

- Classe ModuloElastico: representa classe que define os objetos do tipo módulo elástico.
  - Atributos:
    * Area : área da amostra ;
    * Lo : comprimento inicial da amostra;
    * Deformacao : deformação;
    * Tensao : tensão;
    * MD : módulo elástico;
    * f, fp : objetos que podem tanto ler como escrever em um arquivo
  - Métodos:
    * ModuloElastico(): construtor default para a classe;
    * inicializa (area : float, lo : float): void : método inicializa os atributos área e comprimento inicial;
    * getMD(): vector : retorna o vetor MD;
    * ˜ModuloElastico(): destrói os objetos criados pela classe.
- Classe TenacidadeAFratura: representa classe que define os objetos do tipo tenacidade à fratura.
  - Atributos
    * C: comprimento da fenda;
    * D: comprimento diagonal de indentação
  - Métodos:
    * TenacidadeAFratura(): construtor default para a classe;
    * inicializa(float c, float d): void : método que inicializa os atributos comprimento da fenda e comprimento diagonal de indentação;
    * Calculo(N: vector, T: vector, MD: vector): void: recebe como parâmetros N (carga), T (temperatura) e MD (módulo elástico) e calcula a Tenacidade à fratura.
    * ˜TenacidadeAFratura(): destrói os objetos criados pela classe.
- TaxaDeCorrosao: representa classe que define os objetos do tipo taxa de corrosão.

  - Atributos:
    * wo: peso inicial da amostra;
    * w1: peso final da amostra após ao experimento;
    * T: tempo de imersão;
    * P: densidade do aço;
    * S: superfície exposta da amostra.
  - Métodos:
    * TaxaDeCorrosao(): Construtor default para a classe;
    * inicializa(wo: double,t: double, p: double ,s: double): void: método inicializa os atributos peso inicial da amostra, tempo de imersão, densidade do aço e área de superfície exposta da amostra.
    * Calculo(w1: vector, Temp: double): void: recebe como parâmetros w1 (peso inicial) e Temp (temperatura) e calcula a taxa de corrosão.
    * ˜TaxaDeCorrosao(): destrói os objetos criados pela classe.

- Classe Simulador: responsável pela simulação do ajuste, de acordo com o tipo de propriedade mecânica escolhido pelo usuário
  - main ():void: realiza a simulação.

- Classe Arquivo : responsável pela entrada de dados via arquivo de texto.
  - Atributos:
    * N : carga;
    * L : comprimento;
    * T : temperatura;
  - Métodos:
    * Arquivo () : construtor default
    * abrir (string nomearq int) : void
    * getN() : vector
    * getL() : vector
    * getT() : vector

4.2 Diagrama de seqüência – eventos e mensagens

No diagrama de sequência figura 4.2, enfatiza-se a troca de eventos e mensagens e sua ordem temporal. Contém informações sobre o fluxo de controle do programa. Este

foi montado a partir de um diagrama de caso de uso e estabelece o relacionamento dos usuários e sistemas externos com alguns objetos do sistema.

Para uma melhor visualização, o objeto Propriedade Mecânica representaTenacidadeaFratura, TaxaCorrosiva e ModuloElastico, visto que a troca de eventos e mensagens são similares.

4.2.1 Diagrama de sequência geral

Veja o diagrama de seqüência na Figura 4.2, referente ao programa em questão, onde podemos visualizar o fluxo de controle em meio a execução.

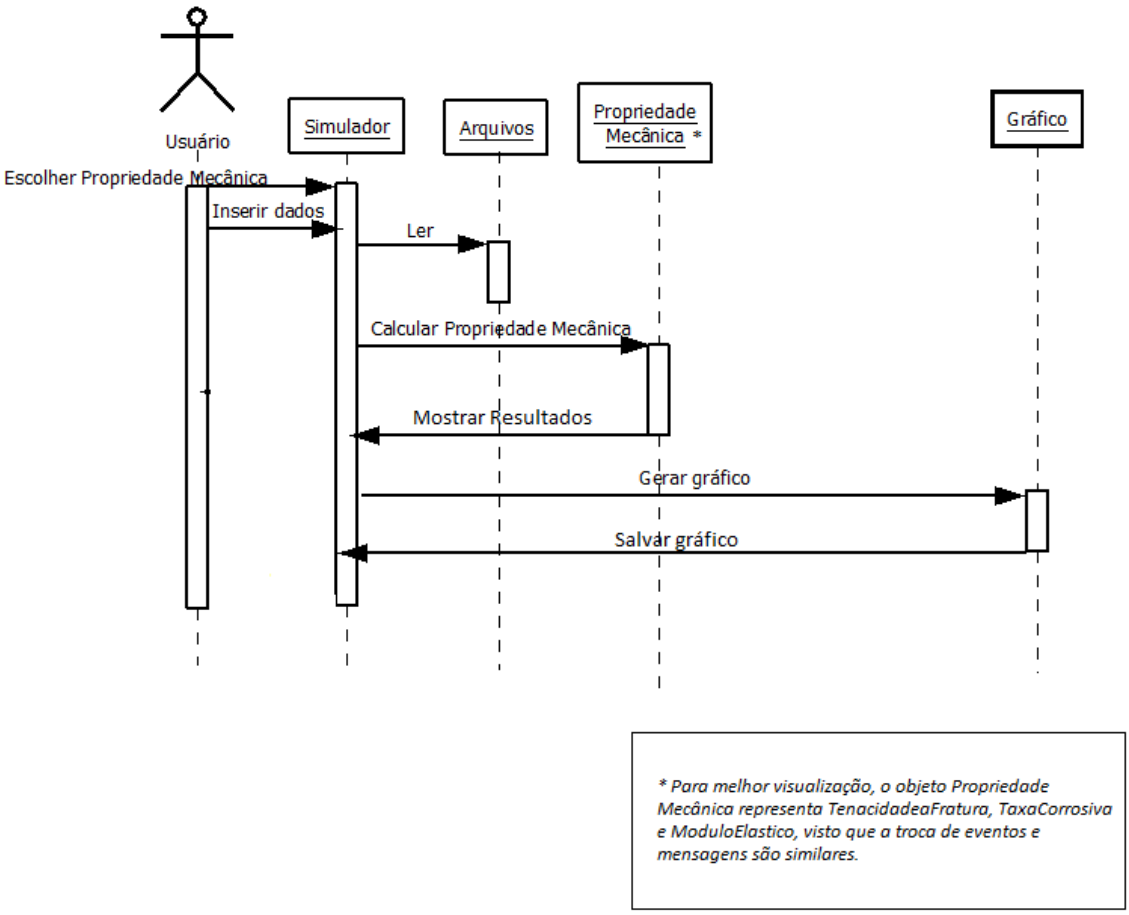


Figura 4.2: Diagrama de seqüência

4.3 Diagrama de comunicação – colaboração

No diagrama de comunicação o foco é a interação e a troca de mensagens e dados entre os objetos.

Veja na Figura 4.3 o diagrama de comunicação. Observe que neste caso, os objetos estão conectados por meio de associações uns com os outros. No exemplo apresentamos o cálculo da Taxa corrosiva.

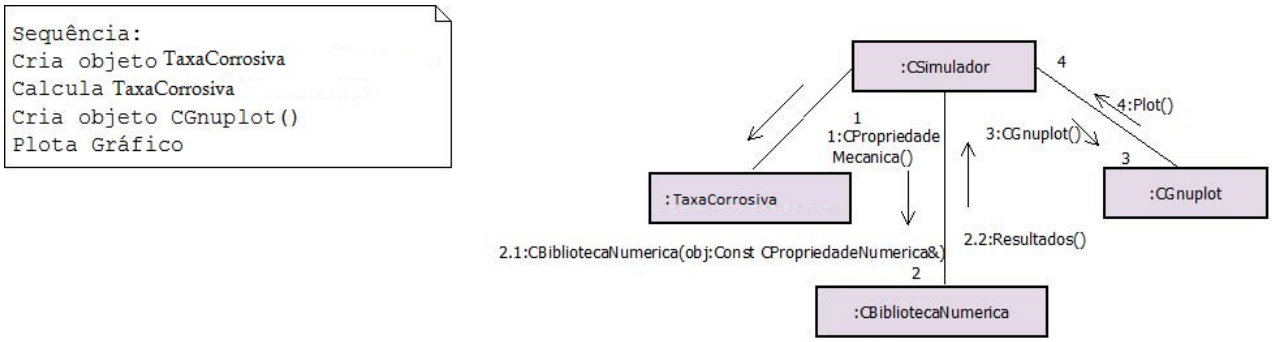


Figura 4.3: Diagrama de comunicação : Cálculo da Taxa Corrosiva

4.4 Diagrama de máquina de estado

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

Veja na Figura 4.4 o diagrama de máquina de estado para o objeto.

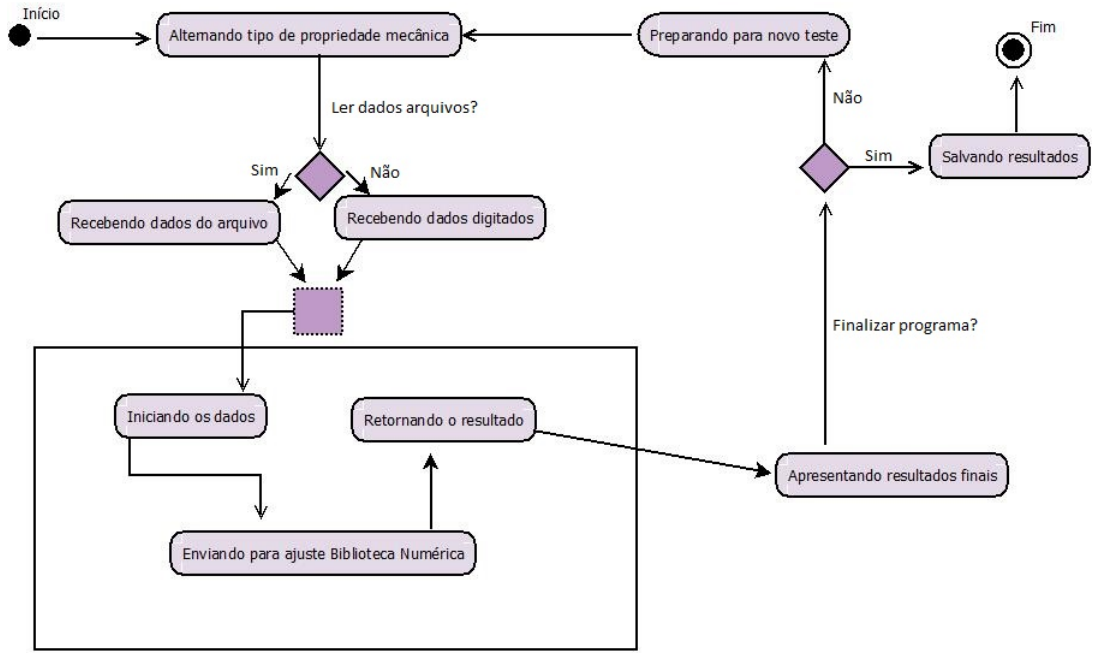


Figura 4.4: Diagrama de máquina de estado:Simulador

4.5 Diagrama de atividades

Segue na figura 4.5 o diagrama de atividade com a descrição do Cálculo da Propriedade Mecânica .4.5

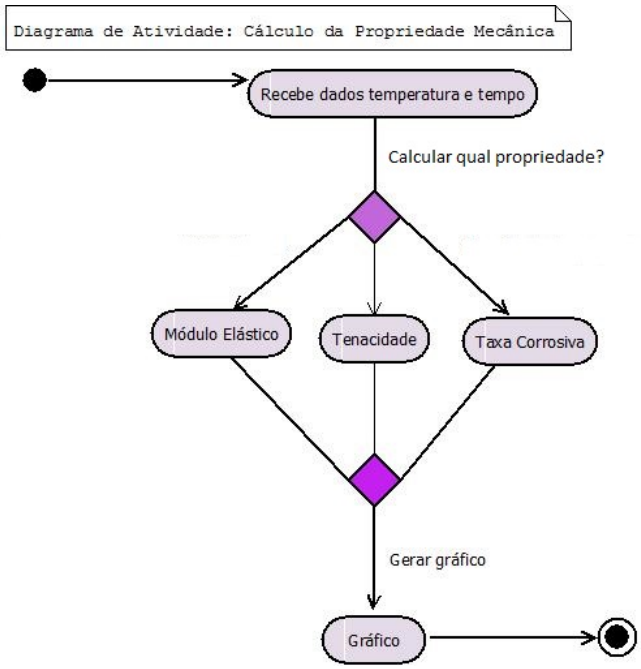


Figura 4.5: Diagrama de atividades para o método Calculo::CSimulador::main

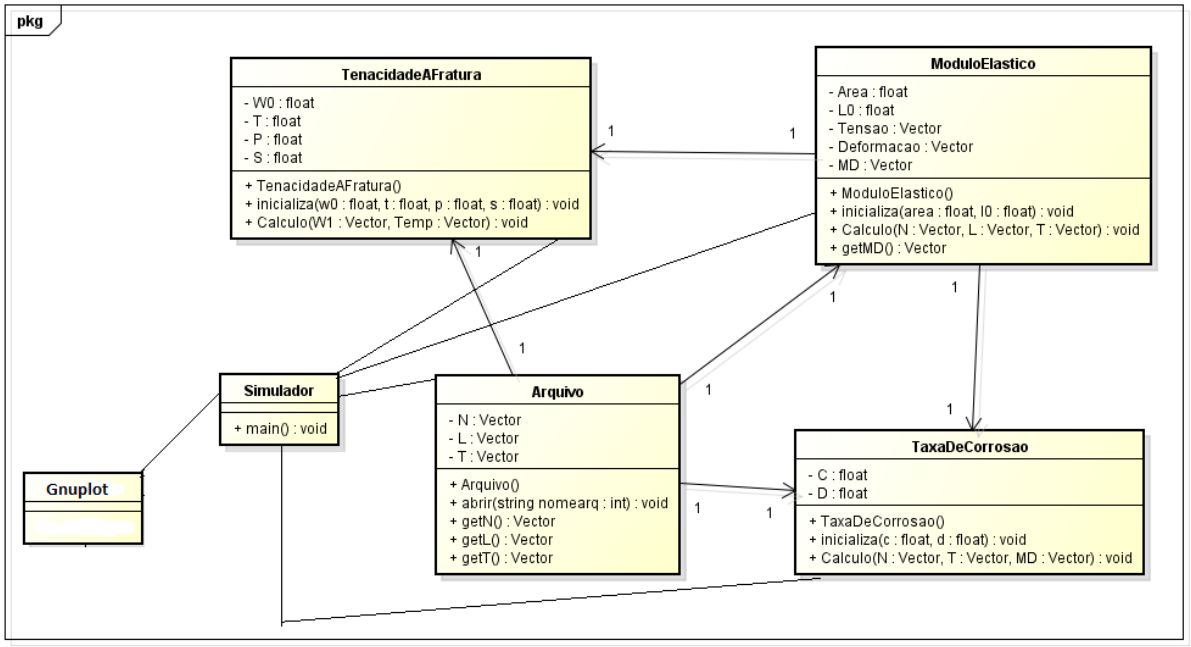


Figura 4.1: Diagrama de classes

Capítulo 5

Projeto

Neste capítulo apresentamos o escopo do projeto idealizado para o software, consistindo tanto do projeto do sistema como do projeto para os objetos envolvidos.

5.1 Projeto do sistema

Depois da análise orientada a objeto, desenvolveu-se o projeto do sistema, o qual reúne etapas como definição dos protocolos, o uso de recursos, a subdivisão do sistema em subsistemas, a alocação dos subsistemas ao hardware e a seleção das estruturas de controle, a seleção das plataformas do sistema, das bibliotecas externas, dos padrões de projeto, além da tomada de decisões conceituais e políticas que formam a infraestrutura do projeto.

Foram definidos padrões de documentação, padrões para o nome das classes, padrões de retorno e de parâmetros em métodos, características da interface do usuário e características de desempenho.

O presente projeto do sistema foi elaborado para apresentar soluções, valendo-se dos seguintes itens como:

1. Protocolos

    - Protocolos de comunicação entre os diversos elementos externos:
        - O sistema tem relação íntima com o hardware do computador utilizado, onde se destaca a necessidade da comunicação com arquivos externos de dados (opcional) ou a inserção manual via teclado pelo usuário;
    - Protocolos de comunicação entre os diversos elementos internos:
        - Os objetos criados devem possuir uma parte privada, acessível somente através dos métodos definidos na sua interface pública;
    - Formato dos arquivos gerados pelo programa:

  - o nome e tipo do arquivo devem ser digitados conforme normas vigentes e salva os preferencialmente nos formatos abaixo, pois estes tem a menor probabilidade de serem corrompidos no salvamento ou em uma leitura posterior pelo programa.
    * *.txt* (ASCII)

2. Recursos

- O presente programa precisará utilizar o HD, o processador, o teclado, a tela, o mouse, a memória e demais componentes internos do computador.
  - Os objetos criados devem possuir uma parte privada, acessível somente através dos métodos definidos na sua interface pública;
  - Nas heranças uma operação ou atributo será redefinida.

3. Controle

- Identificação das condições extremas e de prioridades.
  - Não se aplica.
- Identificação da necessidade de otimização. Por exemplo: prefira sistemas com grande capacidade de memória; prefira vários hds pequenos a um grande.
  - Neste projeto não ha necessidade de uso de processos de otimização. Os cálculos realizados requerem pouco espaço na memória, tanto físico, quanto de processamento.
- Identificação e definição de *loops* de controle e das escalas de tempo.
  - Não se aplica.
- Identificação de concorrências – quais algoritmos podem ser implementados usando processamento paralelo.
  - Neste projeto não ha necessidade de uso de processos de processamento paralelo pois os cálculos realizados requerem pouco esforço de processamento.

4. Plataformas

- Identificação das estruturas arquitetônicas comuns.
- Identificação de subsistemas relacionados à plataforma selecionada. Podem implicar em modificações no diagrama de pacotes e no diagrama de componentes
- Identificação e definição das plataformas a serem suportadas: hardware, sistema operacional e linguagem de programação.

  - O sotware é multiplataforma, funciona no Windows e GNU/Linux.
  - A linguagem selecionada é C++.
- Seleção das bibliotecas externas a serem utilizadas.
  - O software usa a biblioteca externa Gnuplot, que permite o acesso ao gerador de gráfico gnuplot.

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

O projeto orientado a objeto é a etapa posterior ao projeto do sistema. Baseia-se na análise, mas considera as decisões do projeto do sistema. Acrescenta a análise desenvolvida as características da plataforma escolhida (hardware, sistema operacional e linguagem de programação). Passa pelo maior detalhamento do funcionamento do software, acrescentando atributos e métodos que envolvem a solução de problemas específicos não identificados durante a análise.

Envolve a otimização da estrutura de dados e dos algoritmos, a minimização do tempo de execução, de memória e de custos. Existe um desvio de ênfase para os conceitos da plataforma selecionada.

Efeitos do projeto no modelo estrutural

- Adicionar nos diagramas de pacotes as bibliotecas e subsistemas selecionados no projeto do sistema (exemplo: a biblioteca gráfica selecionada).
  - Classes como *Gnuplot* e bibliotecas, header files do C++ como a *iostream.h*, *sstream.h* e *cmath.h* foram utilizadas no programa;
  - Não há restrições associadas à plataforma escolhida.
- Novas classes e associações oriundas das bibliotecas selecionadas e da linguagem escolhida devem ser acrescentadas ao modelo.
  - Não se aplica.

Estabelecer as dependências e restrições associadas à plataforma escolhida. Efeitos do projeto no modelo dinâmico

- Revisar os diagramas de seqüência e de comunicação considerando a plataforma escolhida.

  – Não se aplica.

- Verificar a necessidade de se revisar, ampliar e adicionar novos diagramas de máquinas de estado e de atividades.

  – Não se aplica.

Efeitos do projeto nos atributos

- Atributos novos podem ser adicionados a uma classe, como, por exemplo, atributos específicos de uma determinada linguagem de programação (acesso a disco, ponteiros, constantes e informações correlacionadas).

  – Neste projeto o objeto gnuplot foi adicionado na classe Simulador.

Efeitos do projeto nos métodos

- Em função da plataforma escolhida, verifique as possíveis alterações nos métodos. O projeto do sistema costuma afetar os métodos de acesso aos diversos dispositivos (exemplo: hd, rede).

  – Não se aplica.

- De maneira geral os métodos devem ser divididos em dois tipos: i) tomada de decisões, métodos políticos ou de controle; devem ser claros, legíveis, flexíveis e usam polimorfismo. ii) realização de processamentos, podem ser otimizados e em alguns casos o polimorfismo deve ser evitado.

- Algoritmos complexos podem ser subdivididos. Verifique quais métodos podem ser otimizados. Pense em utilizar algoritmos prontos como os da STL (algoritmos genéricos).

  – Não se aplica.

- Responda a pergunta: os métodos das classes estão dando resposta às responsabilidades da classe?

  – Neste projeto todos os métodos das classes estão correspondendo às responsabilidade da classe.

- Revise os diagramas de classes, de seqüência e de máquina de estado.
  - Não se aplica.

Efeitos do projeto nas heranças

- Reorganização das classes e dos métodos (criar métodos genéricos com parâmetros que nem sempre são necessários e englobam métodos existentes).
  - Não se aplica.
- Utilização de delegação para compartilhar a implementação (quando você cria uma herança irreal para reaproveitar código). Usar com cuidado.
  - Não se aplica.

Nos próximos subtópicos, apresentaremos as dependências dos arquivos e bibliotecas por meio de um diagrama de componentes (veja a seção 5.3), e as relações e dependências entre o sistema e o hardware podem ser ilustradas com o diagrama de implantação (veja a seção 5.4).

5.3 Diagrama de componentes

O diagrama de componentes mostra a forma como os componentes do programa se relacionam, suas dependências. Inclui itens como: componentes, subsistemas, executáveis, nós, associações, dependências, generalizações, restrições e notas.

Exemplo de componentes são bibliotecas estáticas, bibliotecas dinâmicas, dls, componentes Java, executáveis, arquivo de disco, código-fonte.

Algumas observações úteis para o diagrama de componentes:

- De posse do diagrama de componentes, temos a lista de todos os arquivos necessários para compilar e rodar o programa;
- Oberve que um assunto/pacote pode se transformar em uma biblioteca e será incluído no diagrama de componentes;
- A ligação entre componentes pode incluir um estereótipo indicando o tipo de relacionamento ou algum protocolo utilizado.

  Veja na Figura 5.1 o diagrama de componentes. A geração dos objetos depende dos arquivos de classe (extensão .h e .cpp).

  O subsistema Arquivo representa o arquivo que o programa importará os dados a serem manipulados. O programa executável a ser gerado depende das bibliotecas, dos arquivos desta e dos arquivos de entrada.

Veja na Figura 5.2 temos um diagrama de componentes para o programa. Observe que este inclui muitas dependências, ilustrando as relações entre os arquivos. Por exemplo: o subsistema Arquivo, um subsistema externo, inclui os arquivos de código Arquivo.h e Arquivo.cpp

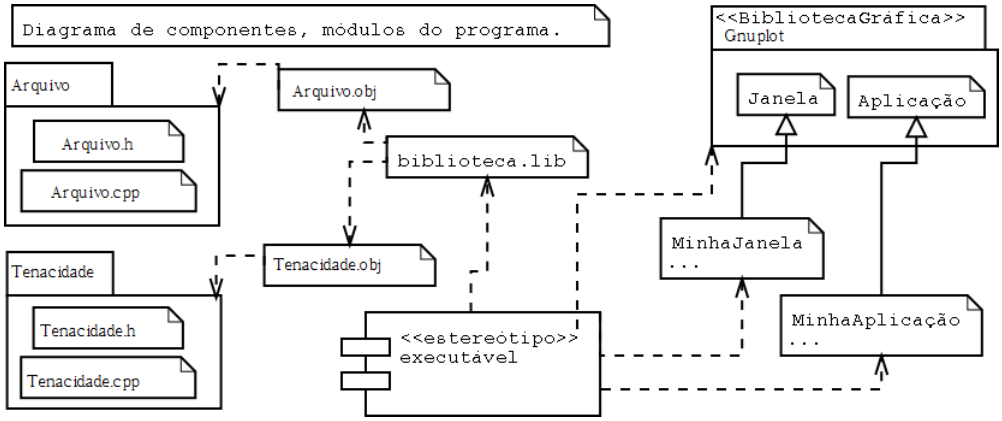


Figura 5.1: Diagrama de componentes

5.4 Diagrama de implantação

O diagrama de implantação é um diagrama de alto nível que inclui relações entre o sistema e o hardware e que se preocupa com os aspectos da arquitetura computacional escolhida. Seu enfoque é o hardware, a configuração dos nós em tempo de execução.

O diagrama de implantação inclui os elementos necessários para que o sistema seja colocado em funcionamento: computador, periféricos, processadores, dispositivos, nós, relacionamentos de dependência, associação, componentes, subsistemas, restrições e notas.

A Figura 5.2 exemplifica o diagrama de implantação do programa de Simulador de Propriedades Mecânicas de Material Submetido ao Processo de Corrosão por Dióxido de Carbono.

Veja na Figura 5.2 diagrama de implantação do programa. Esses arquivos são em formato .txt. O programa importa os dados de arquivo de entrada e na sua execução precisa de um monitor para mostrar os resultados e do teclado para receber parâmetros informados pelo usuário ou cliente.

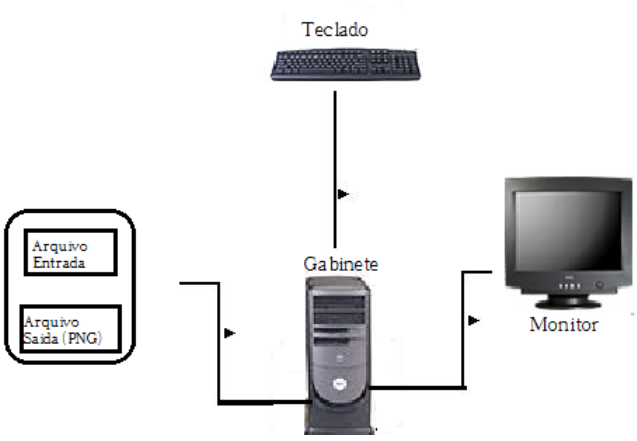


Figura 5.2: Diagrama de implantação

Capítulo 6

Implementação

Neste capítulo apresentaremos o código do simulador de traço sísmico sintético.

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 Arquivo.

Listing 6.1: Arquivo de cabeçalho da classe Arquivo.

```
1#ifndef ARQUIVO_H
2#define ARQUIVO_H
3
4#include <iostream>
5#include <cstdio>
6#include <cstdlib>
7#include <string>
8#include <vector>
9
10class Arquivo
11{
12        std::string linha;
13        double n, l, t;
14        std::vector <double> N, L, T;
15public:
16        Arquivo();
17        virtual ~Arquivo();
18        int abrir(std::string nomearq);
19        std::vector<double> getN();
20        std::vector<double> getL();
21        std::vector<double> getT();
22};
23
24#endif // ARQUIVO_H
```

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

Listing 6.2: Arquivo de implementação da classe Arquivo.

```
25#include "stdafx.h"
26#include "Arquivo.h"
27
28#include <iostream>
29#include <cstdio>
30#include <cstdlib>
31#include <string>
32#include <fstream>
33#include <vector>
34
35using namespace std;
36
37Arquivo::Arquivo()
38{
39        //ctor
40}
41
42Arquivo::~Arquivo()
43{
44        //dtor
45}
46
47int Arquivo::abrir(string nomearq) {
48#ifdef WIN32
49        string nomearqF = "In\\" + nomearq + ".txt";
50#else
51        std::string nomearqF = "In/" + nomearq + ".txt";
52#endif
53        ifstream inFile(nomearqF.c_str());
54        if (!inFile) {
55                cout << "Arquivo nao encontrado, programa encerrado!\n";
56                inFile.clear();
57                return -1;
58        }
59        cout << "Arquivo encontrado com sucesso!\n";
60        string::size_type sz, sz2;
61        while (!inFile.eof()) {
62                getline(inFile, linha);
63                if (!linha.empty()) {
64                        n = stod(linha, &sz);
65                        N.push_back(n);
66                        l = std::stod(linha.substr(sz), &sz2);
67                        L.push_back(l);
68                        t = std::stod(linha.substr(sz).substr(sz2));
69                        T.push_back(t);
```

```
70                  }
71          }
72          inFile.close();
73          inFile.clear();
74          return 0;
75}
76
77
78
79vector<double> Arquivo::getN() {
80
81          return N;
82}
83
84
85vector<double> Arquivo::getL() {
86
87          return L;
88}
89
90
91vector<double> Arquivo::getT() {
92
93          return T;
94}
```

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

Listing 6.3: Arquivo de cabeçalho da classe moduloElastico.

```
95#ifndef MODULOELASTICO_H
96#define MODULOELASTICO_H
97
98#include<iostream>
99#include<cstdio>
100#include<cstdlib>
101#include<string>
102#include<vector>
103#include<fstream>
104#include<cmath>
105
106
107
108class moduloElastico
109{
110         double Area;
111         double L0;
112         std::vector<double> Tensao, Deformacao, MD;
113         std::fstream f, fp;
```

```
114 public:
115         moduloElastico();
116         virtual ~moduloElastico();
117         void inicializa(double area, double l0);
118         void Calculo(std::vector<double> N, std::vector<double> L, std::
                vector<double> T);
119         std::vector<double> getMD();
120
121 };
122
123 #endif // MODULOELASTICO_H
```

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

Listing 6.4: Arquivo de implementação da classe moduloElastico.

```
124 #include "stdafx.h"
125
126 #include "moduloElastico.h"
127 #include<iostream>
128 #include<cstdio>
129 #include<cstdlib>
130 #include <vector>
131 #include<fstream>
132 #include<cmath>
133
134 using namespace std;
135
136
137 moduloElastico::moduloElastico() {
138
139 }
140
141 moduloElastico::~moduloElastico() {
142         //dtor
143 }
144 void moduloElastico::inicializa(double area, double l0) {
145         Area = area;
146         L0 = l0;
147 }
148 void moduloElastico::Calculo(vector<double> N, vector<double> L, vector<
      double> T) {
149         int i = 0;
150         double aux;
151 #ifdef WIN32
152         f.open("Out\\ModuloElastico\\moduloElastico.txt", ios_base::out)
                ;
153         fp.open("Out\\ModuloElastico\\ModuloElastico.plt", ios_base::out
                );
```

```
154#else
155         f.open("./Out/ModuloElastico/moduloElastico.txt", ios_base::out)
               ;
156         fp.open("./Out/ModuloElastico/ModuloElastico.plt", ios_base::out
               );
157#endif
158         cout << "Modulo de distorcao - Temperatura\n";
159         fp << "reset" << endl;
160         fp << "set title\"\"" << endl;
161         fp << "set xlabel \"Temperatura (ºC)\"" << endl;
162         fp << "set ylabel \"Modulo Elastico (GPa)\"" << endl;
163         fp << "f(x) = a*x + b" << endl;
164         fp << "fit f(x) 'moduloElastico.txt' via a,b " << endl;
165         fp << "plot 'moduloElastico.txt', f(x)" << endl;
166         fp << "set term post eps enhanced color" << endl;
167         fp << "set out 'ModuloElastico.eps' " << endl;
168         fp.close();
169         vector<double>::iterator itl = L.begin();
170         vector<double>::iterator itt = T.begin();
171         for (vector<double>::iterator it = N.begin(); it != N.end(); ++
               it, ++itl, ++itt) {
172                 aux = *it;
173                 Tensao.push_back(aux / Area);
174                 if (i == 0) {
175                         Deformacao.push_back(1);
176                 }
177                 else {
178                         Deformacao.push_back((*itl - L0) / L0);
179                 }
180                 MD.push_back((Tensao.at(i)) / (Deformacao.at(i)));
181                 cout << MD.at(i) << " " << *itt << endl;
182                 f << *itt << " " << MD.at(i) << endl;
183                 ++i;
184         }
185         f.close();
186#ifdef WIN32
187         FILE *pipe = _popen("gnuplot\\bin\\gnuplot.exe -persist", "w");
188         if (pipe != NULL) {
189                 fprintf(pipe, "cd 'Out/ModuloElastico'\n");
190                 fprintf(pipe, "load 'ModuloElastico.plt'\n");
191                 fflush(pipe);
192         }
193         else {
194                 cout << "Script não carregado" << endl;
195         }
196         _pclose(pipe);
197#else
198         FILE *pipe = popen("gnuplot -persistent", "w");
```

```
199         if (pipe != NULL) {
200                 fprintf(pipe, "cd␣'Out/ModuloElastico'\n");
201                 fprintf(pipe, "load␣'ModuloElastico.plt'\n");
202                 fflush(pipe);
203         }
204         else {
205                 cout << "Script␣não␣carregado" << endl;
206         }
207         pclose(pipe);
208 #endif
209 }
210
211 std::vector<double> moduloElastico::getMD() {
212         return MD;
213 }
```

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

Listing 6.5: Arquivo de cabeçalho da classe TaxaDeCorrosao.

```
214 #ifndef TAXADECORROSAO_H
215 #define TAXADECORROSAO_H
216
217 #include<iostream>
218 #include<cstdio>
219 #include<cstdlib>
220 #include<string>
221 #include<vector>
222 #include<fstream>
223 #include<cmath>
224
225
226
227
228 class TaxaDeCorrosao
229 {
230         double W0, W1, T, P, S;
231         std::fstream f, fp;
232 public:
233         TaxaDeCorrosao();
234         virtual ~TaxaDeCorrosao();
235         void inicializa(double w0, double t, double p, double s);
236         void Calculo(std::vector<double> W1, std::vector<double> Temp);
237
238 };
239
240 #endif // TAXADECORROSAO_H
```

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

Listing 6.6: Arquivo de implementação da classe TaxaDeCorrosao.

```
241 #include "stdafx.h"
242 #include "TaxaDeCorrosao.h"
243
244 #include<iostream>
245 #include<cstdlib>
246 #include <vector>
247 #include<fstream>
248 #include<cmath>
249
250 using namespace std;
251
252
253 TaxaDeCorrosao::TaxaDeCorrosao()
254 {
255         //ctor
256 }
257
258 TaxaDeCorrosao::~TaxaDeCorrosao()
259 {
260         //dtor
261 }
262 void TaxaDeCorrosao::inicializa(double w0, double t, double p, double s)
      {
263         W0 = w0;
264         T = t;
265         P = p;
266         S = s;
267 }
268 void TaxaDeCorrosao::Calculo(vector<double> W1, vector<double> Temp) {
269 #ifdef WIN32
270         f.open("Out\\TaxaCorrosao\\TaxaDeCorrosao.txt", ios_base::out);
271         fp.open("Out\\TaxaCorrosao\\TaxaDeCorrosao.plt", ios_base::out);
272 #else
273         f.open("./Out/TaxaCorrosao/TaxaDeCorrosao.txt", ios_base::out);
274         fp.open("./Out/TaxaCorrosao/TaxaDeCorrosao.plt", ios_base::out);
275 #endif
276         double sup, inf, Ws;
277         int i = 0;
278         fp << "reset" << endl;
279         fp << "set␣title\"\"" << endl;
280         fp << "set␣xlabel␣\"Temperatura␣(ºC)\"" << endl;
281         fp << "set␣ylabel␣\"Taxa␣de␣Corrosão␣(mm/y)\"" << endl;
282         fp << "f(x)␣=␣a*x␣+␣b" << endl;
283         fp << "fit␣f(x)␣'TaxaDeCorrosao.txt'␣via␣a,b␣" << endl;
284         fp << "plot␣'TaxaDeCorrosao.txt',␣f(x)" << endl;
285         fp << "set␣term␣post␣eps␣enhanced␣color" << endl;
```

```
286         fp << "set␣out␣'TaxaDeCorrosao.eps'␣" << endl;
287         fp.close();
288         vector<double>::iterator itt = Temp.begin();
289         cout << "Taxa␣de␣corrosao␣-␣Temperatura\n";
290         //system("pause");
291         for (vector<double>::iterator it = W1.begin(); it != W1.end();
               ++it, ++itt) {
292                 Ws = W0 - (*it);
293                 sup = Ws * 1000 * 365 * 24;
294                 inf = T*P*S;
295                 cout << (sup / inf) << "␣" << *itt << endl;
296                 f << *itt << "␣" << (sup / inf) << endl;
297                 ++i;
298         }
299         f.close();
300         //system("pause");
301 #ifdef WIN32
302         FILE *pipe = _popen("gnuplot\\bin\\gnuplot.exe␣-persist", "w");
303         if (pipe != NULL) {
304                 fprintf(pipe, "cd␣'Out/TaxaCorrosao'\n");
305                 //system("pause");
306                 fprintf(pipe, "load␣'TaxaDeCorrosao.plt'\n");
307                 //system("pause");
308                 //fflush(pipe);
309                 //system("pause");
310         }
311         else {
312                 cout << "Script␣não␣carregado" << endl;
313         }
314         _pclose(pipe);
315 #else
316         FILE *pipe = popen("gnuplot␣-persistent", "w");
317         if (pipe != NULL) {
318                 fprintf(pipe, "cd␣'Out/TaxaCorrosao'\n");
319                 fprintf(pipe, "load␣'TaxaDeCorrosao.plt'\n");
320                 fflush(pipe);
321         }
322         else {
323                 cout << "Script␣não␣carregado" << endl;
324         }
325         pclose(pipe);
326 #endif
327 }
```

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

Listing 6.7: Arquivo de cabeçalho da classe TenacidadeAFratura.

```
328 #ifndef TENACIDADEAFRATRURA_H
```

```
329#define TENACIDADEAFRATRURA_H
330
331#include <iostream>
332#include <cstdio>
333#include <cstdlib>
334#include <string>
335#include <vector>
336#include <fstream>
337#include <cmath>
338
339
340
341class TenacidadeAFratrura
342{
343        float C, D;
344        std::fstream f, fp;
345public:
346        TenacidadeAFratrura();
347        virtual ~TenacidadeAFratrura();
348        void inicializa(double c, double d);
349        void Calculo(std::vector<double> N, std::vector<double> T, std::
            vector<double> MD);
350};
351
352#endif // TENACIDADEAFRATRURA_H
```

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

Listing 6.8: Arquivo de implementação da classe TenacidadeAFratura.

```
353#include "stdafx.h"
354#include "TenacidadeAFratura.h"
355
356#include<iostream>
357#include<cstdlib>
358#include<cmath>
359#include <vector>
360#include<fstream>
361
362using namespace std;
363
364
365TenacidadeAFratrura::TenacidadeAFratrura()
366{
367        //ctor
368}
369
370TenacidadeAFratrura::~TenacidadeAFratrura()
```

```
371 {
372         //dtor
373 }
374 void TenacidadeAFratrura::inicializa(double c, double d) {
375         C = c;
376         D = d;
377 }
378 void TenacidadeAFratrura::Calculo(vector<double> N, vector<double> T,
        vector<double> MD) {
379         int i = 0;
380         double EP, sup, inf;
381 #ifdef WIN32
382         f.open("Out\\Tenacidade\\tenacidadeAFratura.txt", ios_base::out)
              ;
383         fp.open("Out\\Tenacidade\\TenacidadeAFratura.plt", ios_base::out
              );
384 #else
385         f.open("./Out/Tenacidade/tenacidadeAFratura.txt", ios_base::out)
              ;
386         fp.open("./Out/Tenacidade/TenacidadeAFratura.plt", ios_base::out
              );
387 #endif
388         fp << "reset" << endl;
389         fp << "set␣title\"\"" << endl;
390         fp << "set␣xlabel␣\"Temperatura␣(ºC)\"" << endl;
391         fp << "set␣ylabel␣\"Tenacidade␣a␣fratura␣(MPa␣m^0.5)\"" << endl;
392         fp << "f(x)␣=␣a*x␣+␣b" << endl;
393         fp << "fit␣f(x)␣'tenacidadeAFratura.txt'␣via␣a,b␣" << endl;
394         fp << "plot␣'tenacidadeAFratura.txt',␣f(x)" << endl;
395         fp << "set␣term␣post␣eps␣enhanced␣color" << endl;
396         fp << "set␣out␣'TenacidadeAFratura.eps'␣" << endl;
397         fp.close();
398         vector<double>::iterator itmd = MD.begin();
399         vector<double>::iterator itt = T.begin();
400         cout << "Tenacidade␣a␣fratura␣-␣Temperatura\n";
401         for (vector<double>::iterator it = N.begin(); it != N.end(); ++
              it, ++itmd, ++itt) {
402                 EP = (*it)*(*itmd);
403                 sup = 0.294*D*sqrt(EP);
404                 inf = powf(C, 1.5);
405                 cout << (sup / inf) << "␣" << *itt << endl;
406                 f << *itt << "␣" << (sup / inf) << endl;
407                 ++i;
408         }
409         f.close();
410 #ifdef WIN32
411         FILE *pipe = _popen("gnuplot\\bin\\gnuplot.exe␣-persist", "w");
412         if (pipe != NULL) {
```

```
413                 fprintf(pipe, "cd␣'Out/Tenacidade'\n");
414                 fprintf(pipe, "load␣'TenacidadeAFratura.plt'\n");
415                 fflush(pipe);
416         }
417         else {
418                 cout << "Script␣não␣carregado" << endl;
419         }
420         _pclose(pipe);
421#else
422         FILE *pipe = popen("gnuplot␣-persistent", "w");
423         if (pipe != NULL) {
424                 fprintf(pipe, "cd␣'Out/Tenacidade'\n");
425                 fprintf(pipe, "load␣'TenacidadeAFratura.plt'\n");
426                 fflush(pipe);
427         }
428         else {
429                 cout << "Script␣não␣carregado" << endl;
430         }
431         pclose(pipe);
432#endif
433}
```

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

Listing 6.9: Arquivo de implementação da classe Simulador.

```
434#include "stdafx.h"
435
436#include<iostream>
437#include<cstdio>
438#include<cstdlib>
439#include<string>
440#include <vector>
441
442#include "moduloElastico.h"
443//#include "moduloElastico.cpp"
444#include "TenacidadeAFratura.h"
445//#include "TenacidadeAFratrura.cpp"
446#include "TaxaDeCorrosao.h"
447//#include "TaxaDeCorrosao.cpp"
448#include "Arquivo.h"
449//#include "Arquivo.cpp"
450
451using namespace std;
452
453int main() {
454         int op;
455         double vc, l, temp, area, l0, cp, peso0, tI, d, dens, s, p;
456         std::vector<double> N, L, T, peso1;
```

```
457         string nomearq;
458         moduloElastico mE;
459         TenacidadeAFratrura tF;
460         TaxaDeCorrosao tC;
461         Arquivo arq;
462         cout << "Universidade␣Estadual␣Norte␣Fluminence␣Darcy␣Ribeiro␣\n
               ␣Laboratorio␣de␣Engenharia␣e␣Exploracao␣de␣Petroleo␣\n␣
               Simulador␣de␣Propriedades␣Mecanicas␣de␣Material␣\n␣Submetido␣
               ao␣Processo␣de␣Corrosao␣por␣Dioxido␣de␣Carbono-Versao␣1.0␣\n␣
               Programacao␣Pratica␣Prof.␣Andre␣Duarte␣Bueno␣\n␣copyright␣
               Thayna␣Angelo␣dos␣Reis␣\n" << "Licenca␣GNU/GPL␣v2.0\n"<<endl;
463         cout << "Entre␣com␣o␣numero␣respectivo␣a␣operacao␣desejada:\n␣1-
               ␣Usar␣dados␣de␣arquivo\n␣2-␣Entrar␣com␣dados␣manualmente\n␣3-
               ␣Encerrar␣programa\n";
464         cin >> op;
465         switch (op) {
466         case(1) :
467#ifdef WIN32
468                 system("cls");
469#endif
470                 cout << "Entre␣com␣o␣nome␣do␣arquivo␣sem␣extensao␣.txt,␣
                       que␣esta␣localizado␣na␣pasta␣In␣e␣a␣seguir␣tecle␣
                       enter:\n";
471                 cin >> nomearq;
472                 if (arq.abrir(nomearq) == -1) {
473#ifdef WIN32
474                         system("pause");
475#endif
476                         return 0;
477                 }
478                 N = arq.getN();
479                 L = arq.getL();
480                 T = arq.getT();
481                 cout << "Informe␣a␣area␣original␣[mm^2]␣da␣secao␣
                       transversal␣do␣material:\n";
482                 cin >> area;
483                 cin.ignore();
484                 //setbuf(stdin, NULL);
485                 cout << "Informe␣o␣comprimento␣inicial␣[mm]␣do␣material
                       :\n";
486                 cin >> l0;
487                 cin.ignore();
488                 //setbuf(stdin, NULL);
489                 cout << "Informe␣o␣comprimento␣da␣fenda␣[m]:\n";
490                 cin >> cp;
491                 cin.ignore();
492                 //setbuf(stdin, NULL);
493                 cout << "Informe␣o␣comprimento␣diagonal␣de␣indentacao␣[m
```

```
                    ]:\n";
494                 cin >> d;
495                 cin.ignore();
496                 //setbuf(stdin, NULL);
497                 cout << "Informe o peso original da especie [g]:\n";
498                 cin >> peso0;
499                 cin.ignore();
500                 //setbuf(stdin, NULL);
501                 cout << "Informe o peso final da especie [g] em relacao
                        a seguinte temperatura:\n";
502                 for (vector<double>::iterator it = T.begin(); it != T.
                        end(); ++it) {
503                       cout <<"T[Â°C]= "<< *it << " Peso[g]= ";
504                       cin >> p;
505                       peso1.push_back(p);
506                 }
507                 cin.ignore();
508                 //setbuf(stdin, NULL);
509                 cout << "Informe o tempo de imensao [h]:\n";
510                 cin >> tI;
511                 cin.ignore();
512                 //setbuf(stdin, NULL);
513                 cout << "Informe a densidade do material [g/mm^3]:\n";
514                 cin >> dens;
515                 cin.ignore();
516                 //setbuf(stdin, NULL);
517                 cout << "Informe a area da superficie exposta do
                        material [mm^2]:\n";
518                 cin >> s;
519                 cin.ignore();
520                 //setbuf(stdin, NULL);
521                 mE.inicializa(area, l0);
522                 tF.inicializa(cp, d);
523                 tC.inicializa(peso0, tI, dens, s);
524                 cout << "Fazendo calculos...\n\n";
525                 mE.Calculo(N, L, T);
526                 tF.Calculo(N, T, mE.getMD());
527                 tC.Calculo(peso1, T);
528                 break;
529
530         case(2) :
531#ifdef WIN32
532                 system("cls");
533#endif
534                 cout << "Entre com o valor de carga [N], comprimento [mm
                        ] e temperatura[Celsius] respectivamente:" << endl;
535                 cin >> vc >> l >> temp;
536                 setbuf(stdin, NULL);
```

```
537                 do {
538                         N.push_back(vc);
539                         L.push_back(l);
540                         T.push_back(temp);
541                         system("cls");
542                         cout << "Entre␣com␣o␣valor␣de␣carga␣[N],␣
                               comprimento␣[mm]␣e␣temperatura[Celsius]
                               respectivamente,␣ou␣todos␣os␣valores␣iguais␣a
                               ␣0␣para␣encerrar:" << endl;
543                         cin >> vc >> l >> temp;
544                         setbuf(stdin, NULL);
545                 } while (vc != 0 && l != 0 && temp != 0);
546                 cout << "Informe␣a␣area␣original␣[mm^2]␣da␣secao␣
                        transversal␣do␣material:\n";
547                 cin >> area;
548                 setbuf(stdin, NULL);
549                 cout << "Informe␣o␣comprimento␣inicial␣[mm]␣do␣material
                        :\n";
550                 cin >> l0;
551                 setbuf(stdin, NULL);
552                 cout << "Informe␣o␣comprimento␣da␣fenda␣[m]:\n";
553                 cin >> cp;
554                 setbuf(stdin, NULL);
555                 cout << "Informe␣o␣comprimento␣diagonal␣de␣indentacao[m
                        ]:\n";
556                 cin >> d;
557                 setbuf(stdin, NULL);
558                 cout << "Informe␣o␣peso␣original␣da␣especie␣[g]:\n";
559                 cin >> peso0;
560                 setbuf(stdin, NULL);
561                 cout << "Informe␣o␣peso␣final␣da␣especie␣[g]␣em␣relacao␣
                        a␣seguinte␣temperatura:\n";
562                 for (vector<double>::iterator it = T.begin(); it != T.
                        end(); ++it) {
563                         cout <<"T[Â°C]=␣"<< *it << "␣Peso[g]=␣";
564                         cin >> p;
565                         peso1.push_back(p);
566                 }
567                 setbuf(stdin, NULL);
568                 cout << "Informe␣o␣tempo␣de␣imensao␣[h]:\n";
569                 cin >> tI;
570                 setbuf(stdin, NULL);
571                 cout << "Informe␣a␣densidade␣do␣material␣[g/mm^3]:\n";
572                 cin >> dens;
573                 setbuf(stdin, NULL);
574                 cout << "Informe␣a␣area␣da␣superficie␣exposta␣do␣
                        material␣[mm^2]:\n";
575                 cin >> s;
```

```
576                 setbuf(stdin, NULL);
577                 mE.inicializa(area, l0);
578                 tF.inicializa(cp, d);
579                 tC.inicializa(peso0, tI, dens, s);
580                 cout << "Fazendo calculos...\n\n";
581                 mE.Calculo(N, L, T);
582                 tF.Calculo(N, T, mE.getMD());
583                 tC.Calculo(peso1, T);
584                 break;
585
586         case(3) :
587                 exit(0);
588         default:
589                 cout << "Entrada invalida! Programa encerrado.";
590         }
591#ifdef WIN32
592         system("pause");
593#endif
594         return 0;
595}
```

Capítulo 7

Teste

Neste capítulo apresentaremos uma sequência de testes realizados com o "Simulador de Propriedades Mecânicas de Material Submetido ao Processo de Corrosão por Dióxido de Carbono - Versão 1.0" subdividas em: teste de execução e teste de erro do sistema. Ambos os testes visam submeter o programa a extremas diferentes formas de utilização, ou seja, da execução correta do mesmo até possíveis erros do utilizador.

7.1 Teste 1: Compatibilidade, Compilação, Execução e Desempenho do software

Este teste tem por objetivo a verificação do bom funcionamento das funções do software, bem como da valia dos resultados esperados após o processamento. Para isso, este teste levou em conta o comportamento do programa em dois sistemas operacionais (ambientes Linux e Windows), sua inicialização, o desempenho e a funcionalidade da interface com o usuário, pondo em prova também a qualidade dos resultados da funções incluídas no projeto de classes. Começaremos mostrando o comportamento do programa pela etapa de compatibilidade com sistemas operacionais distintos.

7.1.1 Compatibilidade e Compilação

Para esta etapa, foram escolhidos dois sistemas operativos, um deles da família Microsoft e uma distribuição GNU/Linux.

- Compatibilidade:
  - Foi verificado que o programa pode ser executado em ambientes Linux e Windows.
- Compilação:
  - Foram utilizados os seguintes parâmetros para a execução:

* GNU/Linux : O programa utiliza funções C++11, portanto precisa verificar se o compilador tem suporte completo para C++11. Em g++, por exemplo, você tem que habilitá-lo, verifique qual a versão que está usando com g++ -v, se é uma versão antiga (como 4.2 por exemplo) a funcionalidade não estará disponível.
* Windows: Software Development Kit - contém cabeçalhos, bibliotecas e uma seleção de ferramentas que você pode usar durante a criação de aplicativos que são executados em sistemas operacionais Windows.

7.1.2 Execução e Desempenho

Após a compilação do programa, demos início a fase de execução do mesmo, um teste do sistema. Como resultado deste teste, espera-se que o programa apresente as respostas esperadas. Dentre as duas opções de método de entrada oferecidas na tela inicial. (ver Figuras 7.1 e 7.2) optei pela entrada de dados oriundos de um arquivo externo chamado *Teste.txt*, que contém como parâmetros de entrada os valores de carga [N], comprimento [mm] e temperatura [ºC], baseados no experimento retirado no artigo *Mechanical properties of CO_2 corrosion scale formed at different temperatures and their relationship to corrosion rate,* cuja autoria são de *S.D.Zhu, G.S.Zhou, J.Miao, R.Cai e J.F.Wei.*

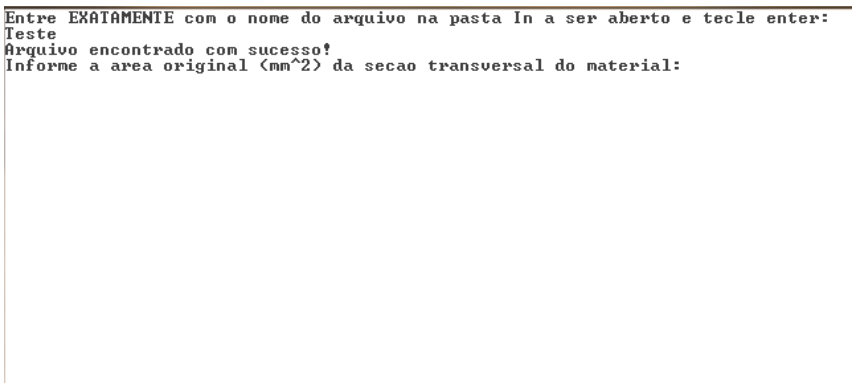


Figura 7.1: Arquivo *Teste.txt* para leitura do programa. Tela mostrando o arquivo de entrada de dados

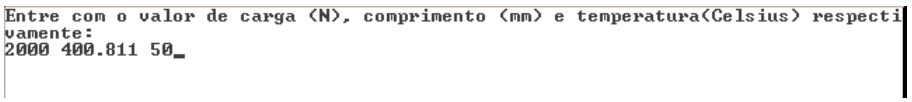


Figura 7.2: O usuário fornece valores de carga, comprimento e temperatura respectivamente fornecido manualmente pelo usuário

Já na Figura 7.3, verificamos a entrada de dados como parâmetros acessórios requeridos pelo programa para execução dos cálculos.

```
Teste
Arquivo encontrado com sucesso!
Informe a area original (mm^2) da secao transversal do material:
40
Informe o comprimento inicial (mm) do material:
400
Informe o comprimento da fenda (m):
0.00025
Informe o comprimento diagonal de indentacao(m):
0.003
Informe o peso original da especie (g):
100
Informe o peso final da especie (g) em relacao a seguinte temperatura:
40 95
50 90
60 85
70 80
100 75
110 70
120 65
130 60
Informe o tempo de imensao (h):
168
Informe a densidade do material (g/mm^3):
7.86
```

Figura 7.3: Parâmetros acessórios

7.2 Resultados

Logo abaixo visualizamos os resultados da simulação. Ao analisarmos os resultados, verificamos que o programa está realizando suas funções de forma satisfatória.

7.2.1 Propriedade mecânica Módulo Elástico

```
Informe o peso final da especie (g) em relacao a seguinte temperatura:
40 95
50 90
60 85
70 80
100 75
110 70
120 65
130 60
Informe o tempo de imensao (h):
168
Informe a densidade do material (g/mm^3):
7.86
Informe a area da superficie exposta do material (mm^2):
20
Fazendo calculos...

Modulo de distorcao - Temperatura
0.025 40
-209.939 50
-432.543 60
-871.068 70
-1186.15 100
-1290.44 110
-1334.67 120
8.7188 130
```

Figura 7.4: Saída na janela de execução do programa

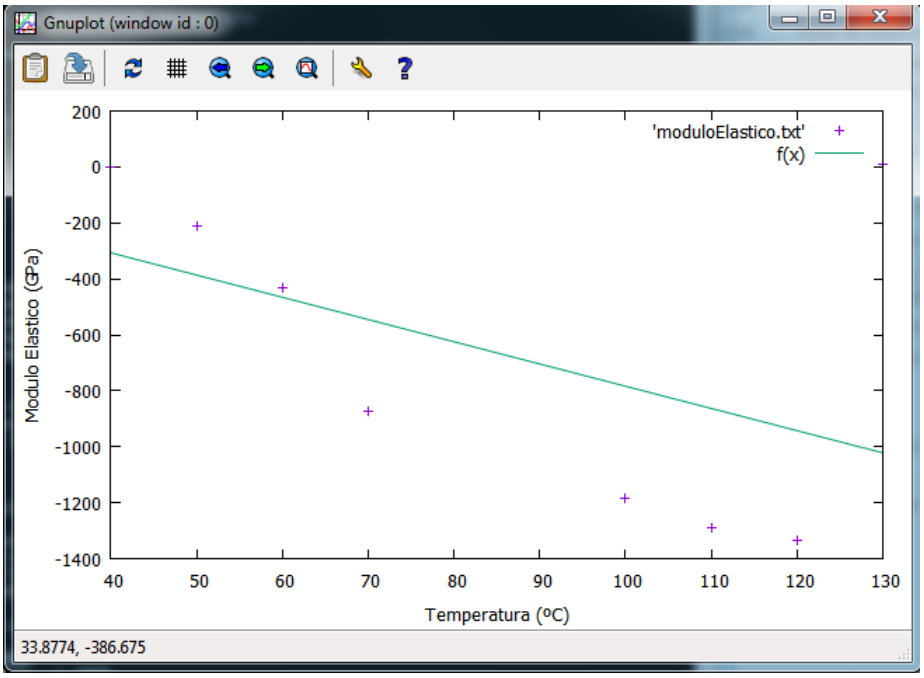


Figura 7.5: Saída do gráfico em arquivo de imagem

7.2.2 Propriedade mecânica Tenacidade à fratura

```
Tenacidade a fratura - Temperatura
35.28 40
-nan(ind) 50
-nan(ind) 60
-nan(ind) 70
-nan(ind) 100
-nan(ind) 110
-nan(ind) 120
143593 130
iter      chisq       delta/lim  lambda   a             b
   0 2.0581345477e+10  0.00e+00 6.80e+01  1.000000e+00  1.000000e+00
   1 2.5326926136e+09 -7.13e+05 6.80e+00  8.074516e+02  2.374284e+00
   2 1.7474195734e+09 -4.49e+04 6.80e-01  1.014047e+03 -5.937201e+02
   3 4.6120056274e+08 -2.79e+05 6.80e-02  1.296839e+03 -3.131265e+04
   4 5.0410986144e+04 -9.15e+08 6.80e-03  1.591968e+03 -6.342883e+04
   5 5.6259719230e-04 -8.96e+12 6.80e-04  1.595085e+03 -6.376812e+04
   6 6.2788188930e-16 -8.96e+16 6.80e-05  1.595086e+03 -6.376815e+04
   7 9.1828761000e-22 -6.84e+10 6.80e-06  1.595086e+03 -6.376815e+04
   8 3.7340257632e-23 -2.36e+06 6.80e-07  1.595086e+03 -6.376815e+04
   9 1.3579009860e-24 -2.65e+06 6.80e-08  1.595086e+03 -6.376815e+04
  10 1.3579009860e-24  0.00e+00 6.80e-09  1.595086e+03 -6.376815e+04
iter      chisq       delta/lim  lambda   a             b
```

Figura 7.6: Saída na janela de execução do programa

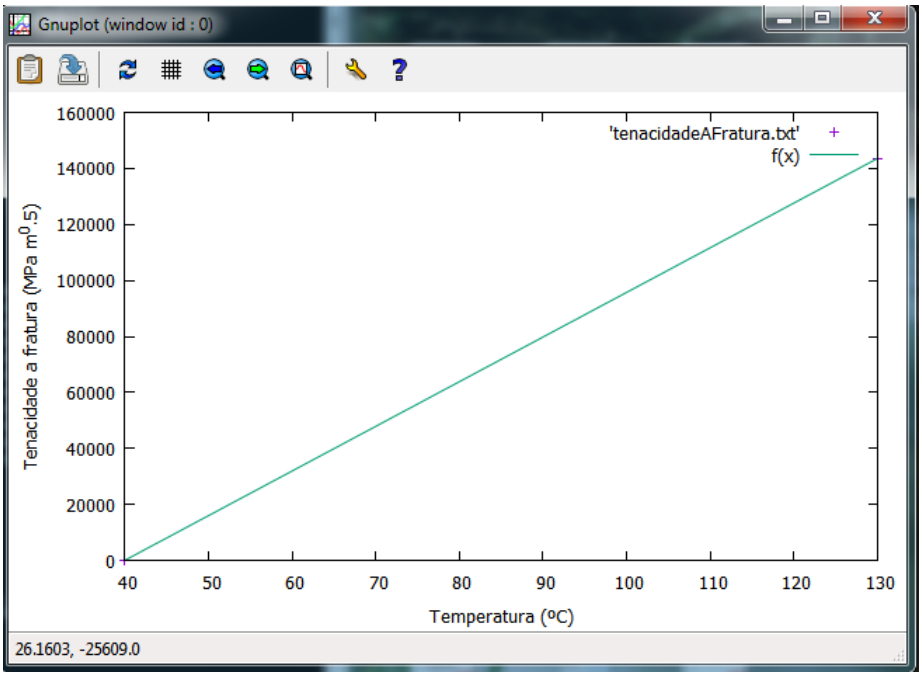


Figura 7.7: Saída do gráfico em arquivo de imagem

7.2.3 Propriedade mecânica Taxa Corrosiva

```
Taxa de corrosao - Temperatura
1658.49 40
3316.98 50
4975.46 60
6633.95 70
3292.44 100
9950.93 110
11609.4 120
13267.9 130
iter      chisq       delta/lim  lambda   a             b
   0 5.4899251690e+08  0.00e+00 6.42e+01  1.000000e+00  1.000000e+00
   1 1.0810528528e+07 -4.98e+06 6.42e+00  8.613349e+01  1.272596e+00
   2 8.6508288953e+06 -2.50e+04 6.42e-01  9.205886e+01 -5.769925e+01
   3 3.1969620990e+06 -1.71e+05 6.42e-02  1.099021e+02 -1.789192e+03
   4 2.6834868554e+06 -1.91e+04 6.42e-03  1.172770e+02 -2.504980e+03
   5 2.6834780802e+06 -3.27e-01 6.42e-04  1.173076e+02 -2.507951e+03
iter      chisq       delta/lim  lambda   a             b
```

Figura 7.8: Saída na janela de execução do programa

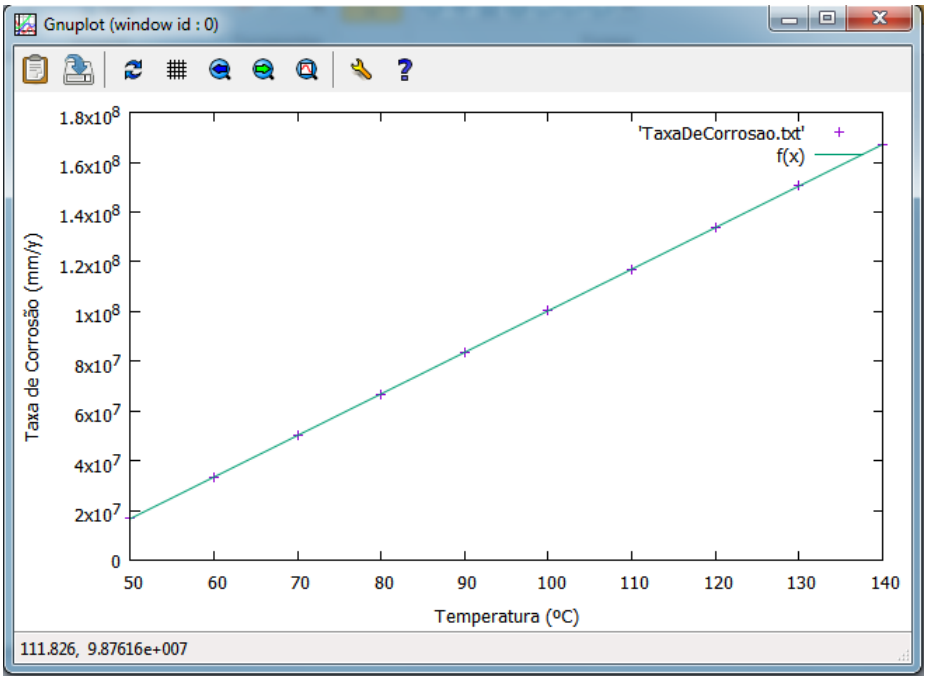


Figura 7.9: Sáida do gráfico em arquivo de imagem

7.3 Teste 2 : Erro

Para este segmento, foi conferido o desempenho do programa com relação a possíveis equívocos tomados pelo usuário durante a execução do programa. Este teste tem a finalidade de verificar se estes erros estão sendo suportados pelo código concebido, não permitindo que eles atrapalhem a obtenção dos resultados.

- Falha de digitação:
  - Caso o usuário informe um arquivo inexistente, o programa acusará o erro ilustrado na Figura 7.10.

```
Entre EXATAMENTE com o nome do arquivo na pasta In a ser aberto e tecle enter:
Lenep
Arquivo nao encontrado, programa encerrado!
Pressione qualquer tecla para continuar. . .
```

Figura 7.10: Teste de erro de arquivo

- Caso o usuário utilize vírgula no lugar do ponto, poderá alterar a execução do programa. Veja ao utilizar 0,00025, o programa pula a pergunta *Informe o comprimento diagonal de indentacao (m)*. Figura 7.11

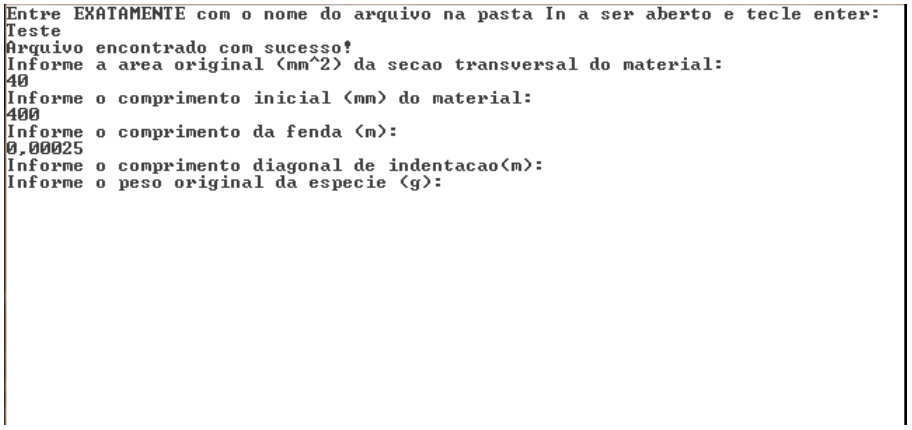


Figura 7.11: Erro de Digitação - Ponto por Vírgula

- Entrada Inexistente
  - Caso o usuário opte por algum tipo de entrada que o programa não possui suporte, o mesmo não permitirá o prosseguimento da simulação. Figura7.12.

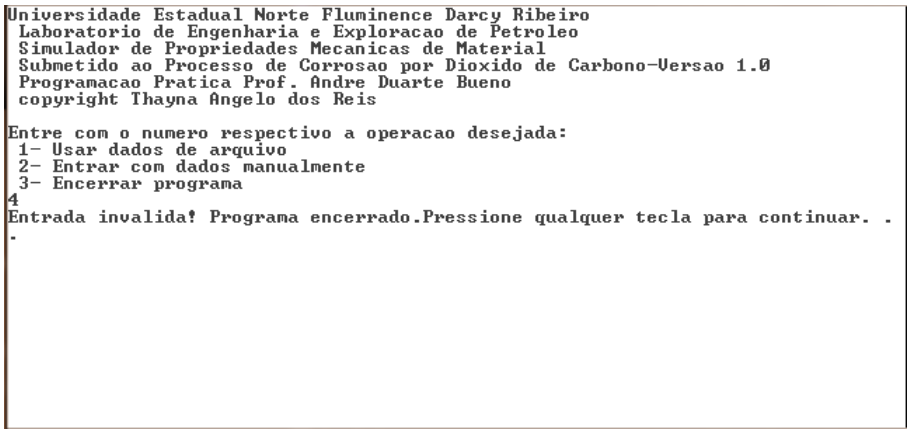


Figura 7.12: Entrada Inválida

Capítulo 8

Documentação

A presente documentação refere-se ao uso do Simulador de Propriedades Mecânicas de Material Submetido ao Processo de Corrosão por Dióxido de Carbono - Versão 1.0 ao ser executado em uma distribuição GNU/Linux ou Windows. Sendo assim, as etapas a seguir se diferem de outras plataformas somente no momento de compilação e execução do programa, onde os procedimentos de utilização deste em diante - ou seja, via Console - é totalmente comum para qualquer sistema operacional.

8.1 Escolha do método de entrada de dados

Como ponto de partida, deve-se inicialmente escolher o método de entrada (1-Usar dados de arquivo, 2 - Entrar com dados manualmente e 3 - Encerrar o programa) digitando o número correspondente ao tipo desejado, como no exemplo da Figura 8.1.

Nesta etapa, tem se disponível duas formas de entradas de dados: do Arquivo (onde o mesmo deve estar no mesmo diretório do executável do programa) ou do teclado (onde cada parâmetro é inserido manualmente).

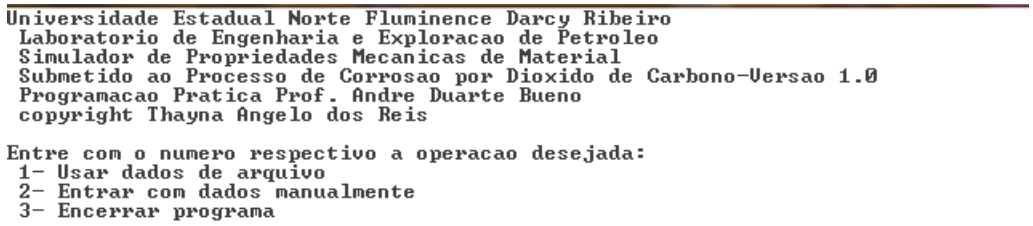
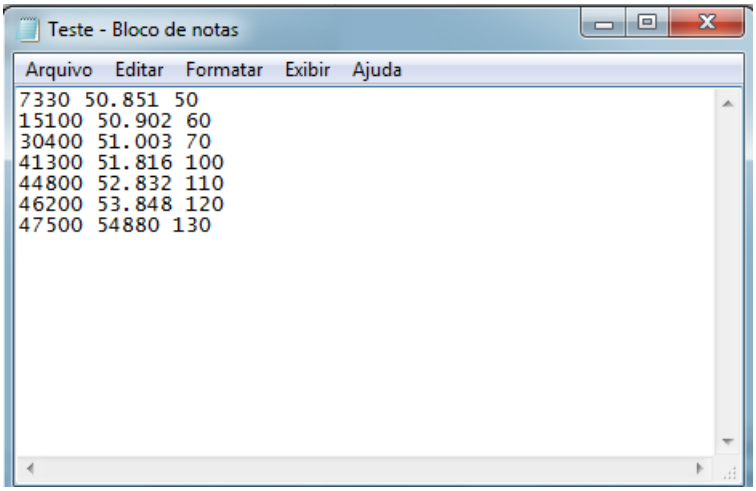


Figura 8.1: Escolha do método de entrada de dados.

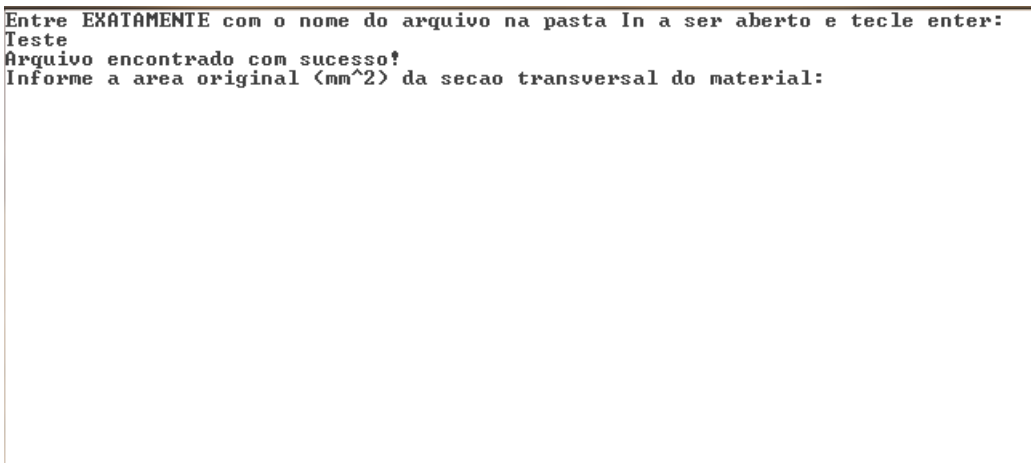
8.1.1 Entrada de dados via arquivo (Opção 1)

Caso escolha a opção de entrada de dados via arquivo de texto, uma nova linha irá surgir para que se possa digitar o nome do arquivo desejado. Nesse arquivo contém dados

das cargas [N], comprimentos da amostras [mm] e temperaturas [°C].Veja na Figura 8.2.



(a) Arquivo txt



(b) Janela de Comando

Figura 8.2: Entrada de dados via arquivo texto Teste.

8.1.2 Entrada de dados manual (Opção 2)

Para a entrada de dados por teclado, deverá ser escrito os valores de carga [N], comprimento [mm] e temperatura [°C], como na Figura 8.3, onde estes valores estão exemplificados.

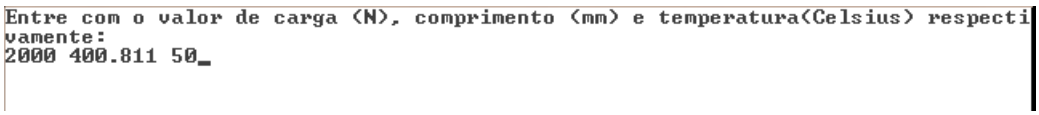


Figura 8.3: Entrada de dados manual (teclado)

8.2 Entada de parâmetros acessórios

Logo após a entrada de dados, via arquivo, o programa pede para o usuário informe:

* área original [mm²] da seção transversal do material;
*comprimento da fenda [m];
*comprimento diagonal de indentação [m];
*peso original da espécie [g];
*peso final da espécie [g] em relação a cada temperatura.
Veja na figura 8.4 um exemplo.

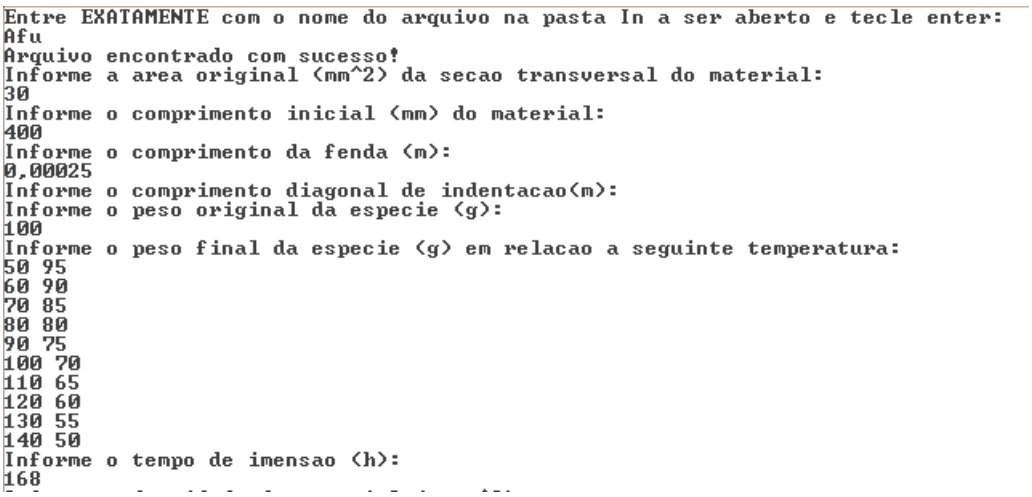


Figura 8.4: Entrada de parâmetros acessórios

8.3 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.

8.3.1 Dependências

Para compilar o software é necessário atender as seguintes dependências:

- No sistema operacional GNU/Linux::
  - Instalar o compilador g++ da GNU disponível em http://gcc.gnu.org.
  - Para instalar no GNU/Linux use o comando `yum install gcc`.
- No sistema operacional Windows:
  - Instalar um compilador apropriado.
  - Recomenda-se o Dev C++ disponível em http://dev-c.softonic.com.br/.
- O software gnuplot, deve estar instalado.
  - Gnuplot está disponível no endereço http://www.gnuplot.info/

– É possível que haja necessidade de setar o caminho para execução do gnuplot.

- O programa depende da existência de um arquivo de dados (formato.txt) para preencher os vetores relacionados aos parâmetros da amostra (carga, comprimento e temperatura).

8.3.2 Documentação usando doxygen

A documentação do código do software foi feita usando o padrão JAVADOC. Depois de documentar o código, o software *doxygen* foi usado para gerar a documentação do desenvolvedor no formato *html*. O software *doxygen* lê os arquivos com os códigos (*.h e *.cpp) e gera uma documentação muito útil e de fácil navegação no formato *html*. Apresenta-se a seguir a imagem de hieraquia de classe mostrada na tela de saída gerada pelo software *doxygen*:

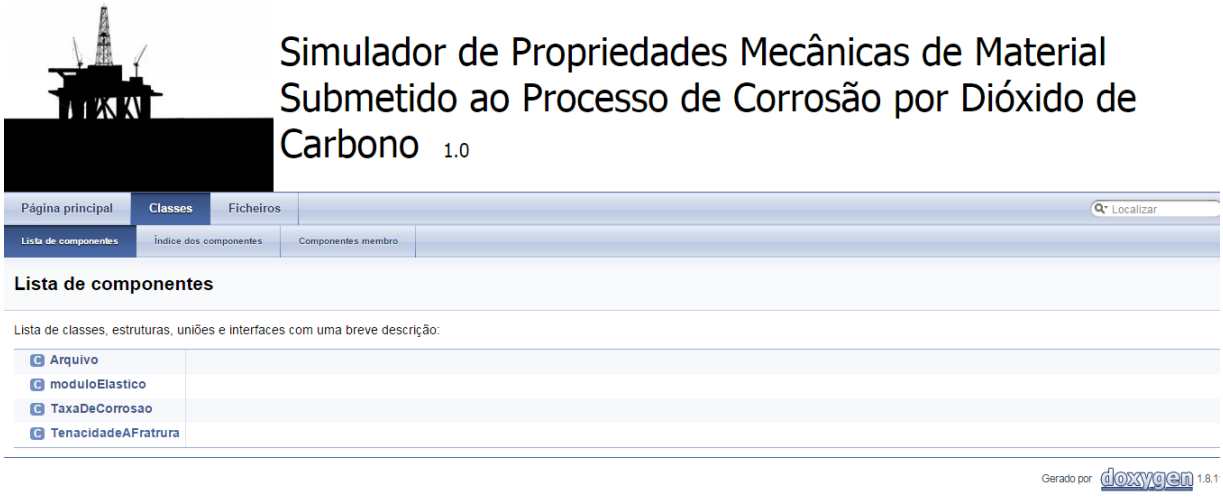


Figura 8.5: Hierarquia de classe

Índice Remissivo