

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

PROJETO ENGENHARIA  
DESENVOLVIMENTO DO SOFTWARE  
XXX  
TRABALHO DA DISCIPLINA PROGRAMAÇÃO PRÁTICA

Versão 1:  
AUTORES

Versão 2:  
AUTORES

Prof. André Duarte Bueno

MACAÉ - RJ

Janeiro - 2015

# Sumário

<b>1</b>	<b>Introdução</b>	<b>1</b>
1.1	Escopo do problema . . . . .	1
1.2	Objetivos . . . . .	1
<b>2</b>	<b>Especificação</b>	<b>3</b>
2.1	O que é a especificação? . . . . .	3
2.2	Nome do sistema/produto . . . . .	4
2.3	Especificação . . . . .	4
2.3.1	Requisitos funcionais . . . . .	4
2.3.2	Requisitos não funcionais . . . . .	5
2.4	O que são os casos de uso do sistema – cenários? . . . . .	5
2.5	Casos de uso . . . . .	6
2.5.1	O que são os diagramas de casos de uso? . . . . .	6
2.5.2	Diagrama de caso de uso geral . . . . .	6
2.5.3	Diagrama de caso de uso específico . . . . .	6
<b>3</b>	<b>Elaboração</b>	<b>9</b>
3.1	O que é a análise de domínio . . . . .	9
3.2	Análise de domínio . . . . .	10
3.3	Formulação teórica . . . . .	11
3.4	O que é identificação de pacotes – assuntos . . . . .	11
3.5	Identificação de pacotes – assuntos . . . . .	11
3.6	O que é o diagrama de pacotes – assuntos . . . . .	12
3.7	Diagrama de pacotes – assuntos . . . . .	12
<b>4</b>	<b>AOO – Análise Orientada a Objeto</b>	<b>13</b>
4.1	Diagramas de classes . . . . .	13
4.1.1	Dicionário de classes . . . . .	13
4.2	Diagrama de seqüência – eventos e mensagens . . . . .	13
4.2.1	Diagrama de seqüência geral . . . . .	14
4.2.2	Diagrama de seqüência específico . . . . .	14
4.3	Diagrama de comunicação – colaboração . . . . .	15

4.4	Diagrama de máquina de estado . . . . .	15
4.5	Diagrama de atividades . . . . .	16
<b>5</b>	<b>Projeto</b>	<b>19</b>
5.1	Projeto do sistema . . . . .	19
5.2	Projeto orientado a objeto – POO . . . . .	21
5.3	Diagrama de componentes . . . . .	25
5.4	Diagrama de implantação . . . . .	26
<b>6</b>	<b>Implementação</b>	<b>28</b>
6.1	Código fonte . . . . .	28
<b>7</b>	<b>Teste</b>	<b>32</b>
7.1	Teste 1: Descrição . . . . .	32
7.2	Teste 2: Descrição . . . . .	33
<b>8</b>	<b>Documentação</b>	<b>35</b>
8.1	Documentação do usuário . . . . .	35
8.1.1	Como instalar o software . . . . .	35
8.1.2	Como rodar o software . . . . .	35
8.2	Documentação para desenvolvedor . . . . .	35
8.2.1	Dependências . . . . .	36
8.2.2	Como gerar a documentação usando doxygen . . . . .	36
<b>9</b>	<b>Título do Apêndice</b>	<b>40</b>
9.1	Sub-Título do Apêndice . . . . .	40
<b>10</b>	<b>Título do Apêndice</b>	<b>41</b>
10.1	Roteiro Para Uso do Sistema de Citações Com Banco de Dados .bib . . . .	42
10.1.1	Citações no meio do texto . . . . .	43
10.1.2	Incluir nas referências bibliográficas (fim do documento), mas não citar . . . . .	43
10.2	Informações adicionais . . . . .	43

# Capítulo 1

## Introdução

No presente projeto de engenharia desenvolve-se o software XXXX, um software aplicado a engenharia de petróleo e que utiliza o paradigma da orientação a objetos.

- O primeiro parágrafo da introdução pode ser um super resumo do trabalho. A ideia é fazer um resumo do resumo.

### 1.1 Escopo do problema

Segundo o CREA/CONFEA um dos quesitos fundamentais que diferenciam a atuação de um tecnólogo da atuação de um engenheiro é a capacidade de desenvolver um projeto em engenharia; Neste trabalho, desenvolve-se um projeto em engenharia de software aplicado a solução de um problema específico de engenharia de petróleo. Trabalhando-se com todo ciclo de um projeto, isto é, especificação, elaboração, análise, projeto, desenvolvimento, teste e documentação.

- ESTE MODELO TEM TEXTOS EXPLICATIVOS QUE DEVEM SER ELIMINADOS DA VERSÃO FINAL. O OBJETIVO DE COLOCAR AS EXPLICAÇÕES É FACILITAR O ENTENDIMENTO DO QUE DEVE ENTRAR EM CADA SEÇÃO.
- Definir o escopo do projeto de engenharia, a ideia geral do software, acentuar sua importância, usos e aplicações em engenharia [de petróleo].
- Delimitar o assunto. Situar-lo no tempo e no espaço. Situar-lo em relação a outros softwares.
- Neste capítulo podem entrar figuras que ilustram a ideia geral, o detalhamento será feito depois, na elaboração.

### 1.2 Objetivos

Os objetivos deste projeto de engenharia são:

- Objetivo geral:
  - Descreva aqui o objetivo geral do projeto de engenharia, incluindo vínculos com engenharia de petróleo e com modelagem matemática computacional (ideia de lógica, algoritmos,...).
  - Desenvolver um projeto de engenharia de software para ...[.....descrever de forma clara, direta, objetiva, o objetivo geral do software].
- 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 (desenvolver algoritmos e diagramas exemplificando os fluxos de processamento).
  - Calcular XXX[.....descrever de forma clara, direta, objetiva, cada objetivo específico, cada parte do software].
  - Calcular XXX[.....descrever de forma clara, direta, objetiva, cada objetivo específico, cada parte do software].
  - Simular (realizar simulações para teste do software desenvolvido).
  - Implementar manual simplificado de uso do software.

# Capítulo 2

## Especificação

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

### 2.1 O que é a especificação?

**Nota:** os textos que explicam o que é especificação; diagrama caso uso; etc....; foram colocados apenas para que você se lembre do assunto; na versão final do seu trabalho estes texto explicativos feitos pelo professor devem ser retirados/apagados.

A primeira etapa do desenvolvimento de um sistema é a concepção, a definição de requisitos a serem satisfeitos e a especificação do sistema (descrição do objetivo e o que se espera do sistema a ser desenvolvido, o contexto da aplicação).

Na concepção ocorre a primeira reunião da equipe de desenvolvimento com os clientes, quando é feita a especificação do software. O cliente por meio das especificações e das entrevistas, passa para o analista idéias gerais de uso do sistema. Neste caso, o cliente é o professor da disciplina e os alunos os analistas/desenvolvedores.

O resultado da etapa de especificação é um conjunto formal de documentos e requisitos organizados pelo analista do sistema com apoio dos usuários.

As *especificações* definem as características gerais do sistema, aquilo que ele deve realizar e não a forma como irá fazê-lo. Definem as necessidades a serem satisfeitas. Envolvem a seleção do tipo de interface, a forma de interação com o usuário; se vai ter uma ou múltiplas janelas; se o software vai imprimir seus resultados, o formato dos arquivos de disco; se vai existir um help e seu formato. Podem ser especificadas características de desempenho. O cliente define o que deve obrigatoriamente ser satisfeito e o que é opcional – isto é, tudo o que o sistema/software deve ser.

**Dica:** a comunicação com o cliente pode ser melhorada com o uso de metáforas, como, por exemplo, a metáfora da "lixeira".

**NOTA IMPORTANTE:** Como requisitos mínimos o software deve:

Solicitar dados ao usuário via terminal (teclado/tela);

Mostrar resultados na tela;

Ler dados de entrada e gerar dados de saída em arquivos de disco;

Plotar gráficos usando biblioteca externa CGnuplot (tem livro do gnuplot na biblioteca );

## 2.2 Nome do sistema/produto

Nome	
Componentes principais	
Missão	

## 2.3 Especificação

Apresenta-se a seguir a especificação do software.

....coloque aqui a especificação...

...eventualmente coloque imagem ilustrativa geral, mostrando o que se pretende (primeira ideia) do software; normalmente esta figura já será mais detalhada que aquela apresentada no escopo...

### 2.3.1 Requisitos funcionais

Apresenta-se a seguir os requisitos funcionais.

<b>RF-01</b>	O sistema deve conter uma base de dados xxx. O usuário deve ser capaz de xxx.
<b>RF-02</b>	O usuário deverá ter liberdade para xxxx.
<b>RF-03</b>	Deve permitir o carregamento de arquivos criados pelo software.
<b>RF-04</b>	Deve permitir a escolha da equação xxx.
<b>RF-05</b>	O usuário deve ter tal liberdade para escolher xxx.
<b>RF-06</b>	O usuário poderá plotar seus resultados em um gráfico. O gráfico poderá ser salvo como imagem ou ter seus dados exportados como texto.

### 2.3.2 Requisitos não funcionais

<b>RNF-01</b>	Os cálculos devem ser feitos utilizando-se xxxx.
<b>RNF-02</b>	O programa deverá ser multi-plataforma, podendo ser executado em <i>Windows</i> , <i>GNU/Linux</i> ou <i>Mac</i> .

## 2.4 O que são os casos de uso do sistema – cenários?

Segundo [Blaha and Rumbaugh, 2006], "um *cenário* pode ser um registro histórico de execução de um sistema real ou um experimento de execução teórica de um sistema proposto".

Um *caso de uso* descreve um ou mais cenários de uso do software, exemplos de uso, como o sistema interage com usuários externos (atores). Ademais, ele deve representar uma seqüência típica de uso do software (a execução 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.

Segundo [?], o caso de uso deve "produzir uma descrição clara e não ambígua de como o usuário final e o sistema interagem um com o outro". [?] diz ainda que "uma vez definidos o ator e seu caso de uso, uma hierarquia de comandos é identificada".

Devem ser montados diversos diagramas de caso de uso. Por exemplo: para cada cenário, crie diagramas de seqüência listando os objetos e os diversos eventos que partem de um objeto para outro.

A Tabela 2.1 mostra os itens a serem incluídos na descrição do caso de uso, podendo-se ainda incluír lista de atores e as pré-condições para execução. O item etapas é opcional, pois elas podem ser incluídas diretamente no diagrama de caso de uso.

Uma vez apresentados os conceitos relacionados aos casos de uso, cenários, apresentaremos como modelar os cenários com os diagramas de caso de uso da UML.

Tabela 2.1: Exemplo de caso de uso

Nome do caso de uso:	Cálculo da integral de uma função.
Resumo/descrição:	Determinação da integral de uma função em um dado intervalo.
Etapas:	<ol style="list-style-type: none"> <li>1. Criar objeto função.</li> <li>2. Criar objeto de integração.</li> <li>3. Definir intervalo de integração e número de pontos.</li> <li>4. Calcular área da função.</li> <li>5. Gerar gráficos.</li> <li>6. Analisar resultados.</li> </ol>
Cenários alternativos:	Um cenário alternativo envolve uma entrada errada do usuário (por exemplo, a função é logarítmica, e o intervalo vai de -1 a 10). O software apresentará um bug quando for determinar o logaritmo de -1.



## 2.5 Casos de uso

....tabela descritiva aqui...

### 2.5.1 O que são os diagramas de casos de uso?

O diagrama de casos de uso é uma representação visual dos casos de uso. É o diagrama mais simples da UML, sendo utilizado para demonstrar os cenários de uso do sistema pelos usuários, os quais ao verem esses diagramas terão uma visão geral do sistema.

O diagrama de caso de uso pode ser utilizado durante e após a etapa de especificação, adicionando às especificações textuais alguns cenários de uso do software a ser desenvolvido.

### 2.5.2 Diagrama de caso de uso geral

....coloque aqui caso de uso geral, descrição + figura(s)...

exemplo...

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

### 2.5.3 Diagrama de caso de uso específico

....coloque aqui casos de uso específicos, descrição + figura(s)...

exemplo...O caso de uso Calcular área função descrito na Figura 2.1 e na Tabela 2.1 é detalhado na Figura 2.2. O usuário criará um objeto função matemática, um objeto para sua integração; em seguida, definirá o intervalo de integração, calculará a área da função criada e, por fim, analisará os resultados (eventualmente gerará gráficos com os resultados obtidos utilizando um sistema externo, como o software *gnuplot*). Este diagrama de caso de uso ilustra as etapas a serem executadas pelo usuário ou sistema, a interação do usuário com o sistema.

**Nota:**

Não perca de vista a visão do todo; do projeto de engenharia como um todo. Cada capítulo, cada seção, cada parágrafo deve se encaixar. Este é um diferencial fundamental do engenheiro em relação ao técnico, a capacidade de desenvolver projetos, de ver o todo e suas diferentes partes, de modelar processos/sistemas/produtos de engenharia.



Figura 2.2: Diagrama de caso de uso específico – Título

# Capítulo 3

## Elaboração

Depois da definição dos objetivos, da especificação do software e da montagem dos primeiros diagramas de caso de uso, a equipe de desenvolvimento do projeto de engenharia passa por um processo de elaboração que envolve o estudo de conceitos relacionados ao sistema a ser desenvolvido, a análise de domínio e a identificação de pacotes.

Na elaboração fazemos uma análise dos requisitos, ajustando os requisitos iniciais de forma a desenvolver um sistema útil, que atenda às necessidades do usuário e, na medida do possível, permita seu reuso e futura extensão.

Eliminam-se os requisitos "impossíveis" e ajusta-se a idéia do sistema de forma que este seja flexível, considerando-se aspectos como custos e prazos.

### 3.1 O que é a 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 softwares e bibliotecas a serem investigadas). Uma boa maneira de identificar os assuntos relacionados ao sistema que está sendo desenvolvido é dar uma olhada nos livros, manuais e artigos da área.
- Manter contato com especialistas que tenham domínio dos conceitos envolvidos. Segundo [?], estas pessoas "devem ser abertas, ter bastante conhecimento prático e estar disponíveis para perguntas".
- Deve-se fazer entrevistas com os usuários do sistema (para detalhamento dos diagramas de caso de uso).

- Identificação de amostras (instalar e testar softwares e bibliotecas similares).
- Na elaboração deve-se analisar a possibilidade de se reaproveitar projetos antigos (sistemas existentes etc). Procurar as classes em bibliotecas existentes antes de implementá-las [?]. Se a equipe de desenvolvimento já tem experiência com alguns sistemas, isto deve ser considerado. A idéia é reaproveitar o conhecimento legado.
- Identificação e definição dos objetos genéricos (a serem reutilizados).

Talvez o aspecto mais importante da elaboração seja o contato com o cliente. Quando o cliente/usuário tem uma noção básica de como o sistema é desenvolvido, ele passa a fazer solicitações que serão mais facilmente implementadas.

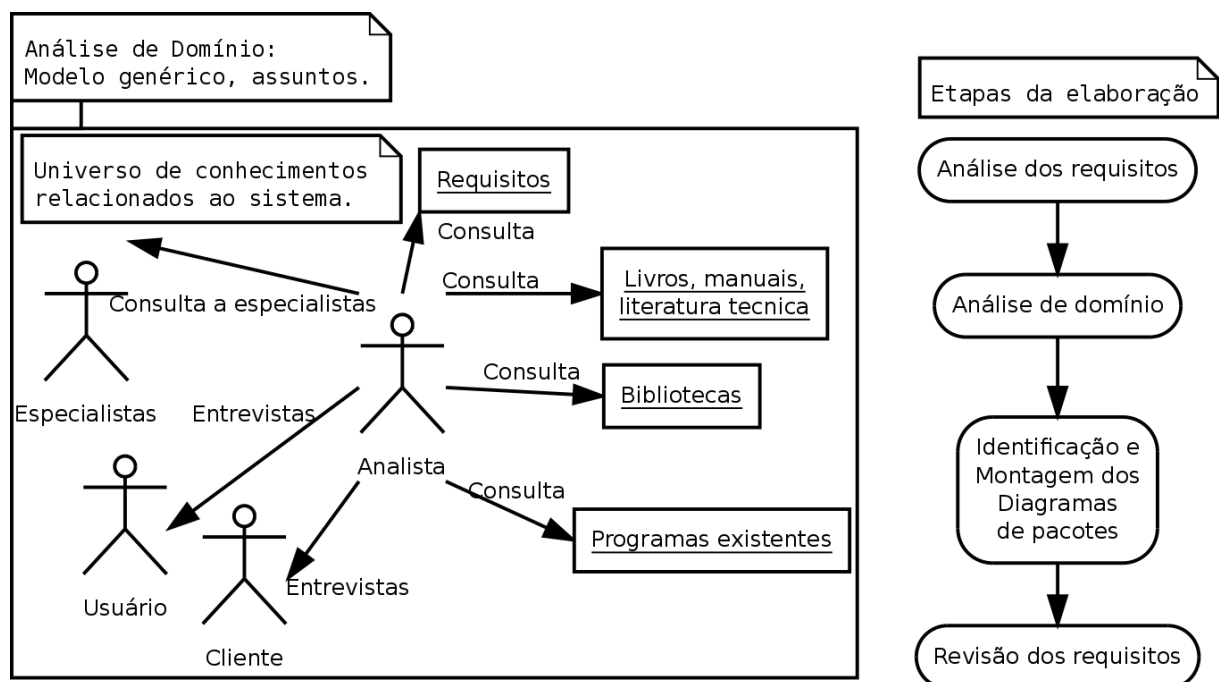


Figura 3.1: Elaboração e análise de domínio (adaptado de [?])

## 3.2 Análise de domínio

Após estudo dos requisitos/especificações do sistema, algumas entrevistas, estudos na biblioteca e disciplinas do curso foi possível identificar nosso domínio de trabalho:

- Definição e caracterização do domínio a ser investigado
- Descrição das áreas/disciplinas relacionadas.
- Descrição de questões associadas a espaço/tempo (espaço físico, local, instalação).
- Fotografias de sistemas equivalentes/parecidos.
- ...

### 3.3 Formulação teórica

...coloque aqui texto explicando a formulação física-matemática, a descrição das equações que serão utilizadas...

...aqui entram figuras, tabelas explicativas, equações a serem utilizadas no software...

...logo depois, com os dados levantados na análise de domínio, e esta formulação teórica, iremos identificar os pacotes, os assuntos com os quais iremos trabalhar...

... no final da formulação teórica, indicar livros e artigos que tem mais informação sobre o assunto...

...se vai usar modelos numéricos, colocar seção 3.3.x falando deles; no final citar/referenciar fontes...

### 3.4 O que é identificação de pacotes – assuntos

Um assunto é aquilo que é tratado ou abordado em uma discussão, em um estudo e é utilizado para orientar o leitor em um modelo amplo e complexo.

- Se um grupo de classes troca muitas informações entre si, provavelmente as classes fazem parte de um mesmo assunto. Assim, classes que se relacionam por meio de conceitos em comum fazem parte de uma mesma área, são reunidas e formam um assunto (pacote).
- Em um pacote, os nomes de classes e de associações devem ser semelhantes.
- Preste atenção a semelhanças na forma. Procedimentos semelhantes indicam polimorfismo e são candidatos a superclasses. Usualmente a classe mais genérica de uma hierarquia de classes identifica um assunto.

### 3.5 Identificação de pacotes – assuntos

...aqui...

- Nome Pacote: Descrição. O que é, para que serve, como se relaciona com os demais pacotes.
- Nome Pacote: Descrição. O que é, para que serve, como se relaciona com os demais pacotes.
- Nome Pacote: Descrição. O que é, para que serve, como se relaciona com os demais pacotes.

### 3.6 O que é o diagrama de pacotes – assuntos

Um diagrama de pacotes é útil para mostrar as dependências entre as diversas partes do sistema; pode incluir: sistemas, subsistemas, hierarquias de classes, classes, interfaces, componentes, nós, colaborações e casos de uso.

O exemplo da Figura 3.2 apresenta um diagrama de pacotes para um sistema de métodos numéricos. Observe que o diagrama de pacotes inclui um pacote bibliotecas, o qual é dividido em bibliotecas matemáticas, de estatística e de simulação. A seguir o componente biblioteca de simulação será detalhado.

### 3.7 Diagrama de pacotes – assuntos

...aqui...

coloque aqui texto falando do diagrama de pacotes, reference a figura. Veja Figura 3.2.

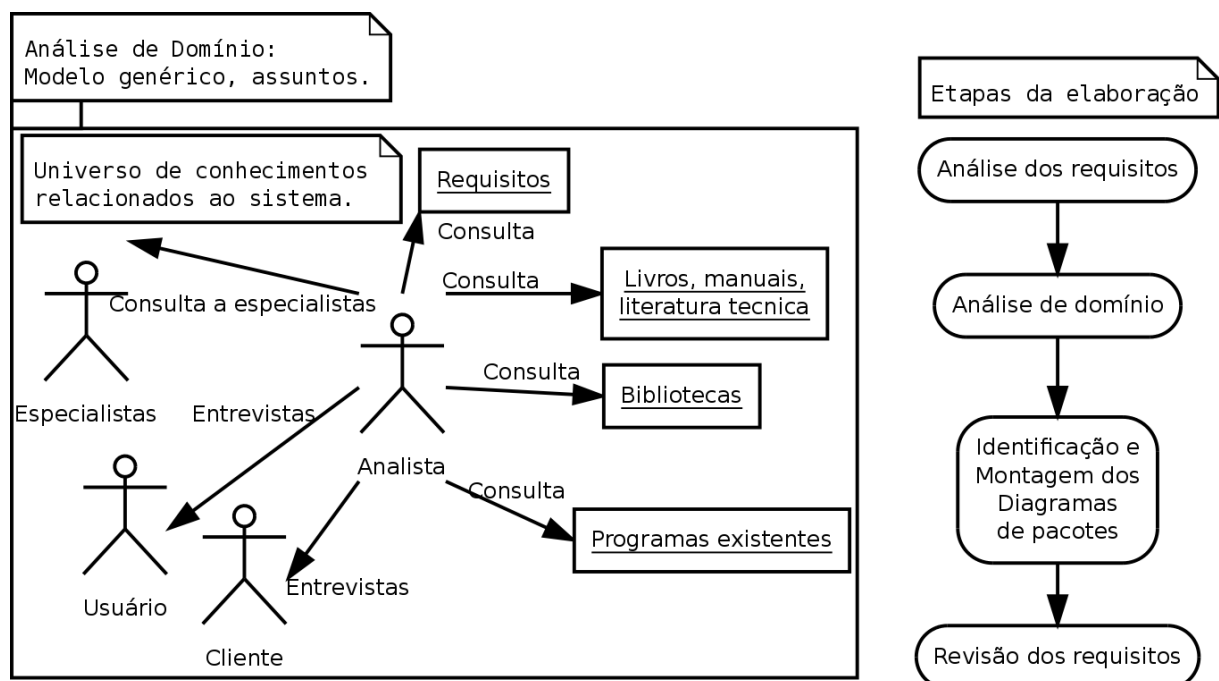


Figura 3.2: Diagrama de Pacotes

#### Nota:

Não perca de vista a visão do todo; do projeto de engenharia como um todo. Cada capítulo, cada seção, cada parágrafo deve se encaixar. Este é um diferencial fundamental do engenheiro em relação ao técnico, a capacidade de desenvolver projetos, de ver o todo e suas diferentes partes, de modelar processos/sistemas/produtos de engenharia.

# 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 os pacotes, 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.

Nota:

deve ocupar toda a página impressa! se necessário rotacionar 90 graus; SE NECESSÁRIO DIVIDIR EM 2 PÁGINAS; o importante é que toda figura/tabela deve ser bem legível (fonte mínima = 10).

#### 4.1.1 Dicionário de classes

- Classe CNomeClasse: representa.....
- Classe CNomeClasse: representa.....
- Classe CNomeClasse: representa.....

### 4.2 Diagrama de seqüência – 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 geral

Veja o diagrama de sequência na Figura 4.2.

- [Aqui a ênfase é o entendimento da sequência com que as mensagens são trocadas, a ordem temporal.]

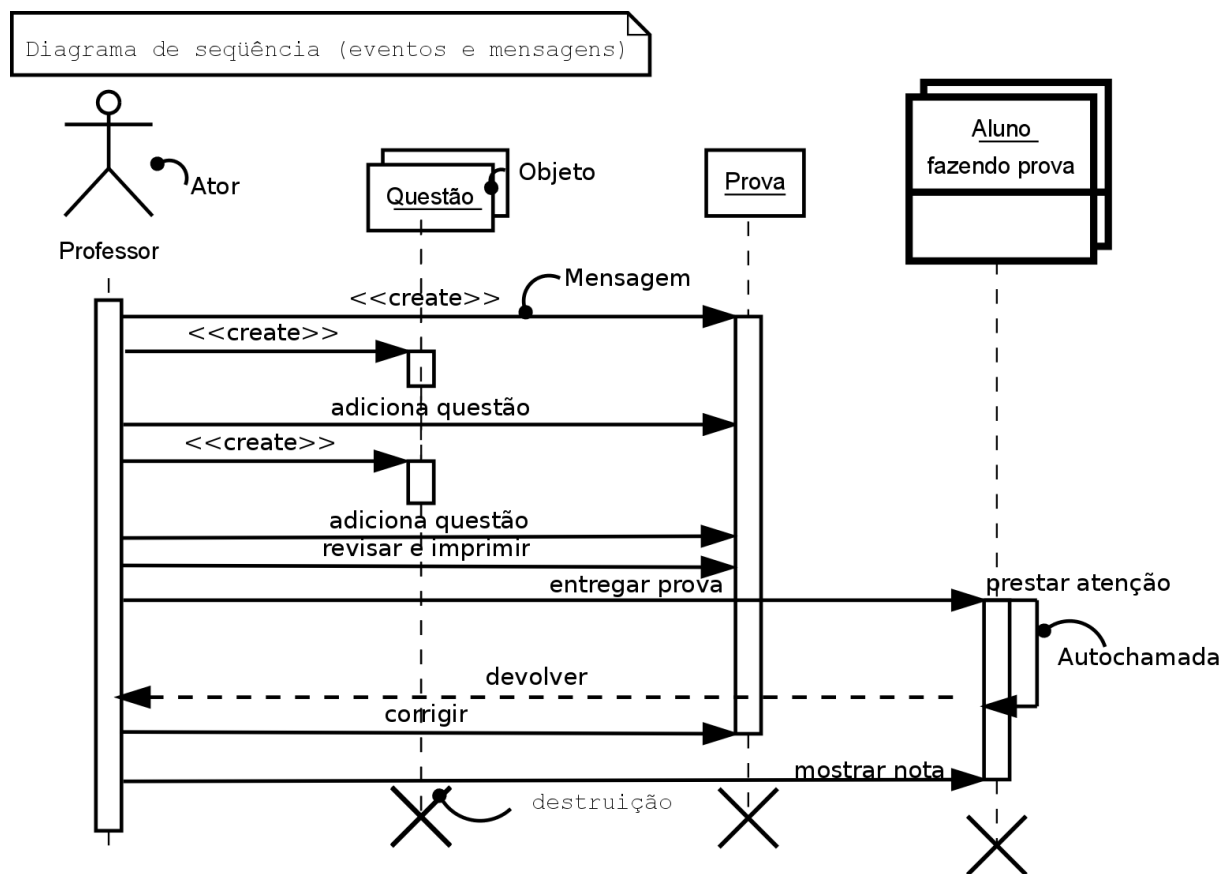


Figura 4.2: Diagrama de sequência

### 4.2.2 Diagrama de sequência específico

...

- [deve mostrar uma sequência específica; NÃO É PARA REPETIR O GERAL COM 1-2 coisas diferentes!]

é um novo diagrama; detalhando algo! ]

## 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.

- [Pode ser a repetição de um diagrama de sequência; mas note que o formato do gráfico é diferente, aqui a ênfase é o entendimento das mensagens que chegam e saem de cada objeto.]

Veja na Figura 4.3 o diagrama de comunicação mostrando a sequência de blablabla. Observe que ....

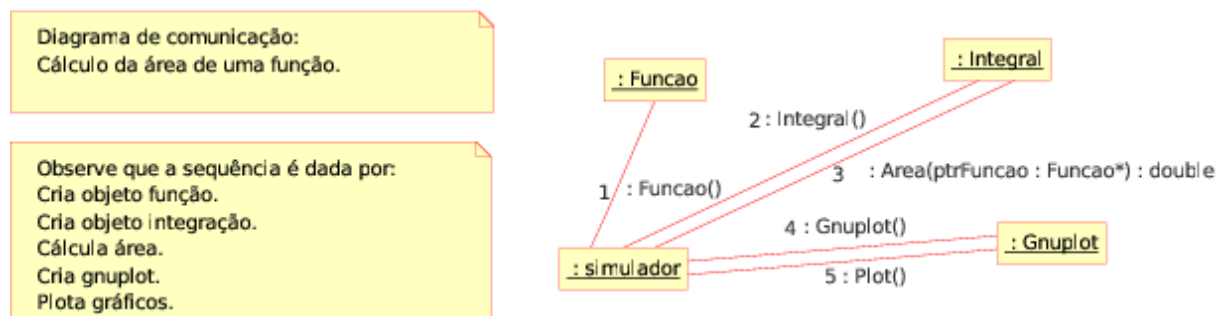


Figura 4.3: Diagrama de comunicação

## 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 ao 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 XXX. Observe que....

- Lembre-se, são os estados de um objeto específico e não uma sequência de cálculo; as sequência já foram mostrados nos diagramas de sequência e comunicação!!
- Vou repetir; Não faça o diagrama de máquina de estado como sendo uma repetição dos diagramas de sequência!
- Este diagrama trata dos estados de um objeto único/específico!

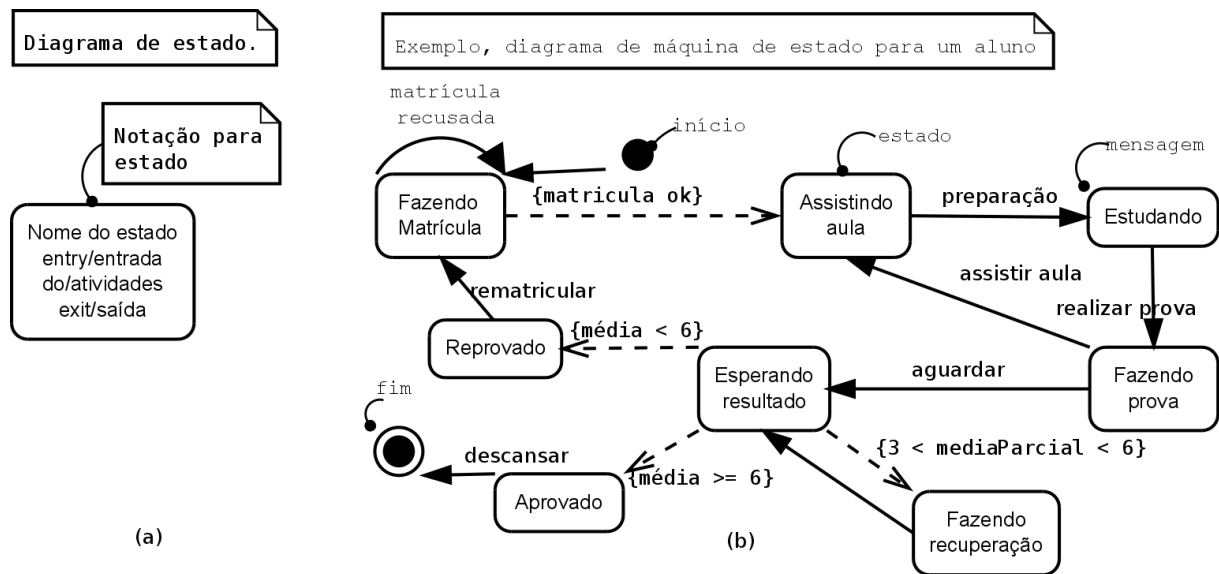


Figura 4.4: Diagrama de máquina de estado

## 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....

...descrever em detalhes uma atividade específica..não pode ser a sequência de uso geral, trata-se de um caso específico, detalhado do diagrama de máquina de estado.

- Lembrar que o diagrama de sequência é a representação de um método de cálculo específico.
- Não é para fazer o diagrama de atividades do método de gerenciamento!!!
- Coloque aqui um diagrama de atividades que mostra contas/cálculos!

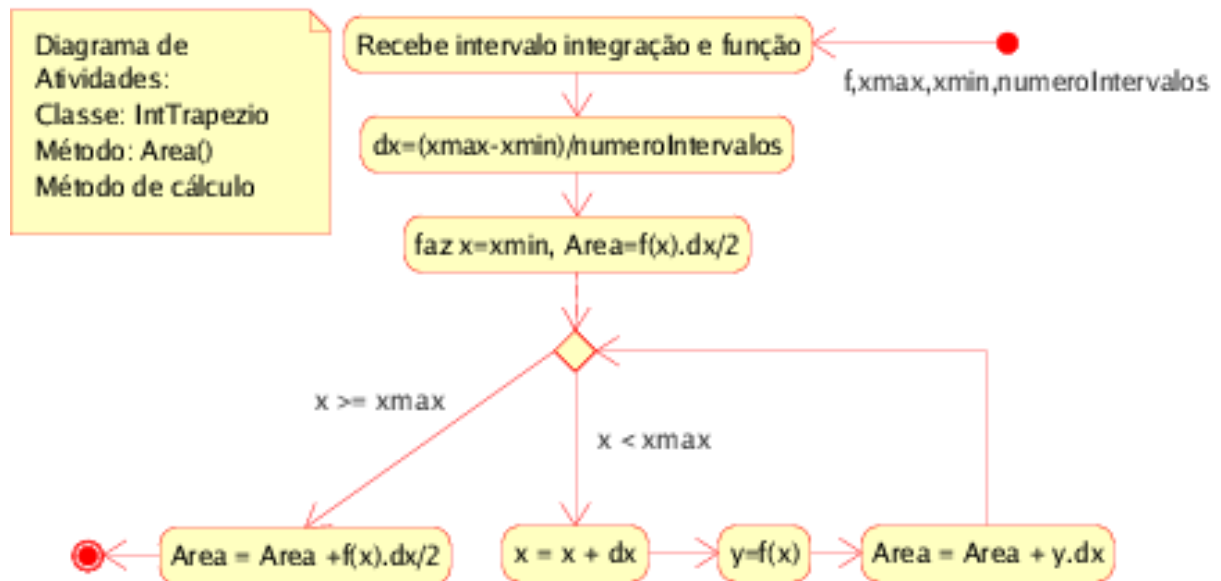


Figura 4.5: Diagrama de atividades

**Nota:**

Não perca de vista a visão do todo; do projeto de engenharia como um todo. Cada capítulo, cada seção, cada parágrafo deve se encaixar. Este é um diferencial fundamental do engenheiro em relação ao técnico, a capacidade de desenvolver projetos, de ver o todo e suas diferentes partes, de modelar processos/sistemas/produtos de engenharia.

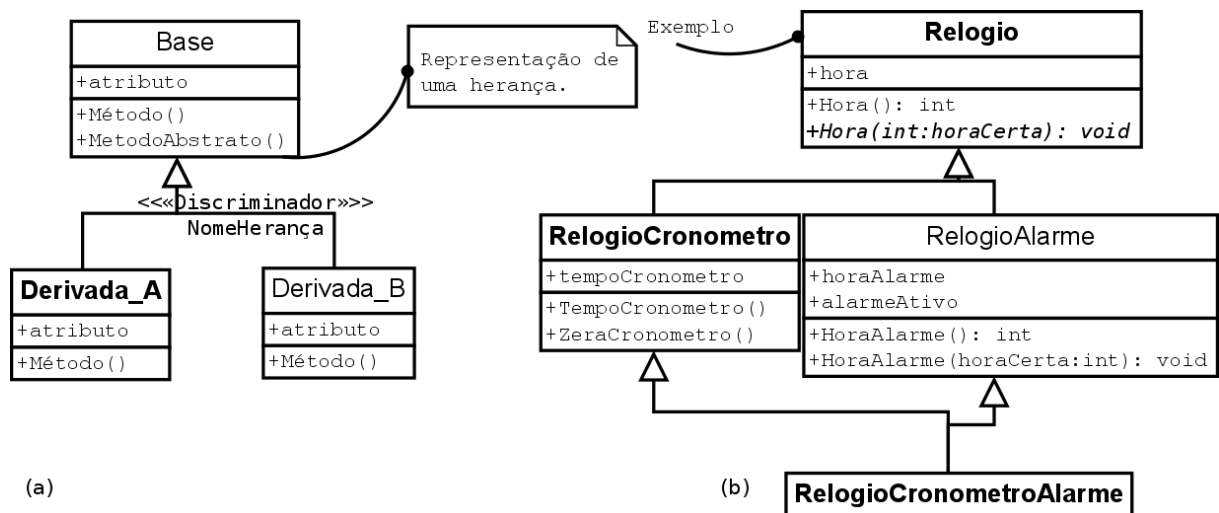


Figura 4.1: Diagrama de classes

# 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

Depois da análise orientada a objeto desenvolve-se o projeto do sistema, qual envolve etapas como a definição dos protocolos, da interface API, 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.

Deve-se definir 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.

Segundo [Rumbaugh et al., 1994, Blaha and Rumbaugh, 2006], o projeto do sistema é a estratégia de alto nível para resolver o problema e elaborar uma solução. Você deve se preocupar com itens como:

#### 1. Protocolos

- Definição dos protocolos de comunicação entre os diversos elementos externos (como dispositivos). Por exemplo: se o sistema envolve o uso dos nós de um cluster, devem ser considerados aspectos como o protocolo de comunicação entre os nós do cluster.
  - Neste projeto blablabla
- Definição dos protocolos de comunicação entre os diversos elementos internos (como objetos).

- Neste projeto blablabla
- Definição da interface API de suas bibliotecas e sistemas.
  - Neste projeto blablabla
- Definição do formato dos arquivos gerados pelo software. Por exemplo: prefira formatos abertos, como arquivos txt e xml.
  - Neste projeto blablabla

## 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 blablabla
- Identificação da necessidade do uso de banco de dados. Implicam em modificações nos diagramas de atividades e de componentes.
  - Neste projeto blablabla
- Identificação da necessidade de sistemas de armazenamento de massa. Por exemplo: um *storage* em um sistema de cluster ou sistemas de backup.
  - Neste projeto blablabla

## 3. Controle

- Identificação e seleção da implementação de controle, sequencial ou concorrente, baseado em procedimentos ou eventos. Implicam modificações no diagrama de execução.
  - Neste projeto blablabla
- Identificação das condições extremas e de prioridades.
  - Neste projeto blablabla
- Identificação da necessidade de otimização. Por exemplo: prefira sistemas com grande capacidade de memória; prefira vários hds pequenos a um grande.
  - Neste projeto blablabla
- Identificação e definição de *loops* de controle e das escalas de tempo.
  - Neste projeto blablabla
- Identificação de concorrências – quais algoritmos podem ser implementados usando processamento paralelo.
  - Neste projeto blablabla

#### 4. Plataformas

- Identificação das estruturas arquitetônicas comuns.
  - Neste projeto blablabla
- Identificação de subsistemas relacionados à plataforma selecionada. Podem implicar em modificações no diagrama de pacotes e no diagrama de componentes.
  - Neste projeto blablabla
- Identificação e definição das plataformas a serem suportadas: hardware, sistema operacional e linguagem de software.
  - Neste projeto blablabla
- Seleção das bibliotecas externas a serem utilizadas.
  - Neste projeto blablabla
- Seleção da biblioteca utilizada para montar a interface gráfica do software – GDI.
  - Neste projeto blablabla
- Seleção do ambiente de desenvolvimento para montar a interface de desenvolvimento – IDE.
  - Neste projeto blablabla

#### 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.
  - Neste projeto blablabla

### 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 e as características da plataforma escolhida (hardware, sistema operacional e linguagem de softwareção). 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.



Exemplo: na análise você define que existe um método para salvar um arquivo em disco, define um atributo nomeDoArquivo, mas não se preocupa com detalhes específicos da linguagem. Já no projeto, você inclui as bibliotecas necessárias para acesso ao disco, cria um objeto específico para acessar o disco, podendo, portanto, acrescentar novas classes àquelas desenvolvidas na análise.

### **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).
  - Neste projeto blablabla
- Novas classes e associações oriundas das bibliotecas selecionadas e da linguagem escolhida devem ser acrescentadas ao modelo.
  - Neste projeto blablabla
- Estabelecer as dependências e restrições associadas à plataforma escolhida.
  - Neste projeto blablabla

### **Efeitos do projeto no modelo dinâmico**

- Revisar os diagramas de seqüência e de comunicação considerando a plataforma escolhida.
  - Neste projeto blablabla
- Verificar a necessidade de se revisar, ampliar e adicionar novos diagramas de máquinas de estado e de atividades.
  - Neste projeto blablabla

### **Efeitos do projeto nos atributos**

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

**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).
  - Neste projeto blablabla
- 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.
  - Neste projeto blablabla
- Algoritmos complexos podem ser subdivididos. Verifique quais métodos podem ser otimizados. Pense em utilizar algoritmos prontos como os da STL (algoritmos genéricos).
  - Neste projeto blablabla
- Responda a pergunta: os métodos da classes estão dando resposta às responsabilidades da classe?
  - Neste projeto blablabla
- Revise os diagramas de classes, de seqüência e de máquina de estado.
  - Neste projeto blablabla

**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).
  - Neste projeto blablabla
- Abstração do comportamento comum (duas classes podem ter uma superclasse em comum).
  - Neste projeto blablabla
- Utilização de delegação para compartilhar a implementação (quando você cria uma herança irreal para reaproveitar código). Usar com cuidado.
  - Neste projeto blablabla

- Revise as heranças no diagrama de classes.
  - Neste projeto blablabla

### **Efeitos do projeto nas associações**

- Deve-se definir na fase de projeto como as associações serão implementadas, se obedecerão um determinado padrão ou não.
  - Neste projeto blablabla
- Se existe uma relação de "muitos", pode-se implementar a associação com a utilização de um dicionário, que é um mapa das associações entre objetos. Assim, o objeto A acessa o dicionário fornecendo uma chave (um nome para o objeto que deseja acessar) e o dicionário retorna um valor (um ponteiro) para o objeto correto.
  - Neste projeto blablabla
- Evite percorrer várias associações para acessar dados de classes distantes. Pense em adicionar associações diretas.
  - Neste projeto blablabla

### **Efeitos do projeto nas otimizações**

- Faça uma análise de aspectos relativos à otimização do sistema. Lembrando que a otimização deve ser desenvolvida por analistas/desenvolvedores experientes.
  - Neste projeto blablabla
- Identifique pontos a serem otimizados em que podem ser utilizados processos concorrentes.
  - Neste projeto blablabla
- Pense em incluir bibliotecas otimizadas.
- Se o acesso a determinados objetos (atributos/métodos) requer um caminho longo (exemplo: A->B->C->D.atributo), pense em incluir associações extras (exemplo: A-D.atributo).
  - Neste projeto blablabla
- Atributos auxiliares podem ser incluídos.
  - Neste projeto blablabla

- A ordem de execução pode ser alterada.
  - Neste projeto blablabla
- Revise as associações nos diagramas de classes.
  - Neste projeto blablabla

Depois de revisados os diagramas da análise você pode montar dois diagramas relacionados à infraestrutura do sistema. As dependências dos arquivos e bibliotecas podem ser descritos pelo diagrama de componentes, e as relações e dependências entre o sistema e o hardware podem ser ilustradas com o diagrama de implantação.

### 5.3 Diagrama de componentes

O diagrama de componentes mostra a forma como os componentes do software se relacionam, suas dependências. Inclui itens como: componentes, subsistemas, executáveis, nós, associações, dependências, generalizações, restrições e notas. Exemplos de componentes são bibliotecas estáticas, bibliotecas dinâmicas, dlls, componentes Java, executáveis, arquivos de disco, código-fonte.

Veja na Figura 5.1 um exemplo de diagrama de componentes. Observe que este inclui muitas dependências, ilustrando as relações entre os arquivos. Por exemplo: o subsistema biblioteca inclui os arquivos das classes A e B, e a geração dos objetos A.obj e B.obj depende dos arquivos A.h, A.cpp, B.h e B.cpp. A geração da biblioteca depende dos arquivos A.obj e B.obj. O subsistema biblioteca Qt, um subsistema externo, inclui os arquivos de código da biblioteca Qt e a biblioteca em si. O subsistema banco de dados representa o banco de dados utilizado pelo sistema e tem uma interface de acesso que é utilizada pelo software para acesso aos dados armazenados no banco de dados. O software executável a ser gerado depende da biblioteca gerada, dos arquivos da biblioteca Qt, do módulo de arquivos MinhaJanela e do banco de dados.

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 software.
- Observe 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.

Neste projeto blablabla

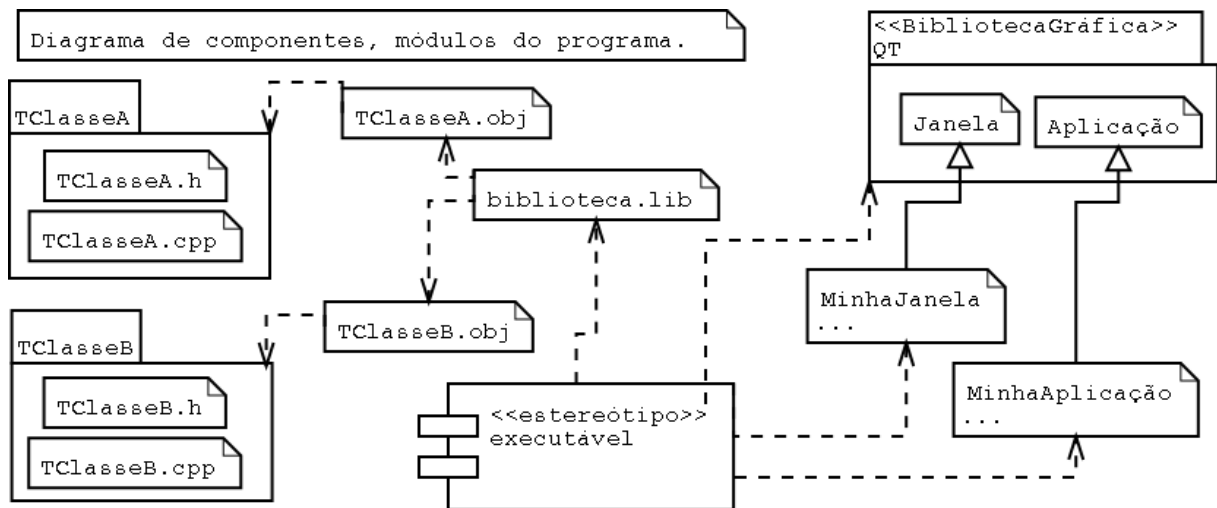


Figura 5.1: Diagrama de componentes

## 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 deve incluir 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.

Veja na Figura 5.2 um exemplo de diagrama de implantação de um cluster. Observe a presença de um servidor conectado a um switch. Os nós do cluster (ou clientes) também estão conectados ao switch. Os resultados das simulações são armazenados em um servidor de arquivos (*storage*).

Pode-se utilizar uma anotação de localização para identificar onde determinado componente está residente, por exemplo {localização: sala 3}.

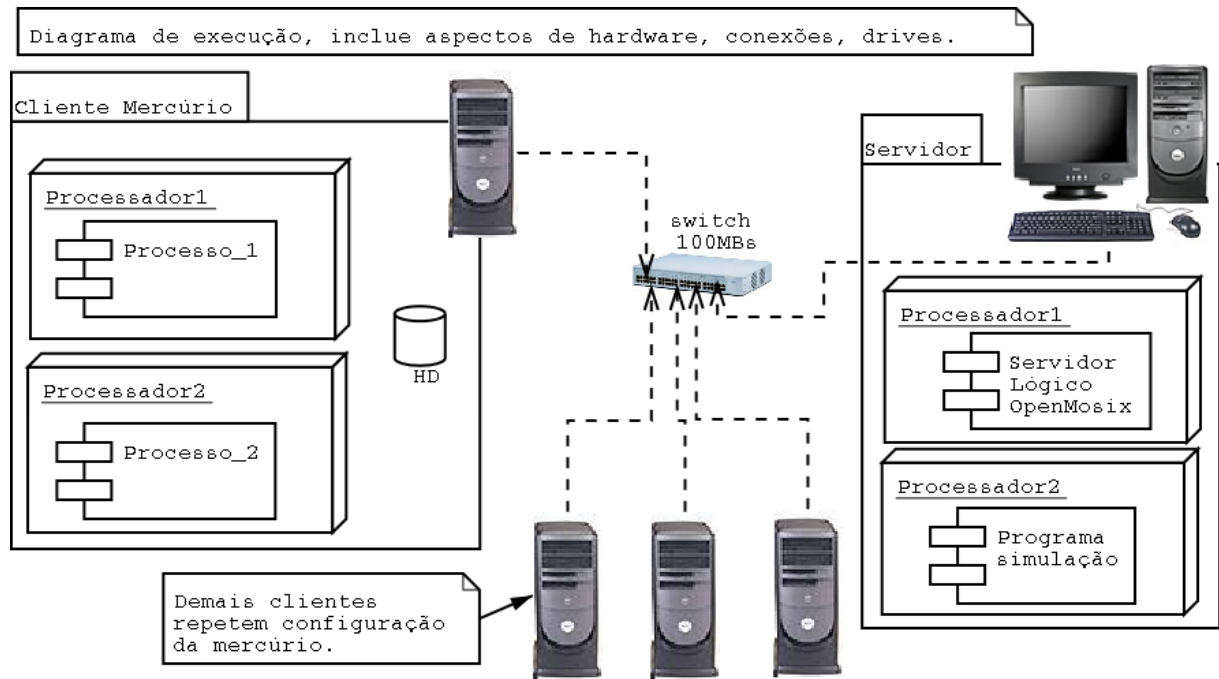


Figura 5.2: Diagrama de implantação

**Nota:**

Não perca de vista a visão do todo; do projeto de engenharia como um todo. Cada capítulo, cada seção, cada parágrafo deve se encaixar. Este é um diferencial fundamental do engenheiro em relação ao técnico, a capacidade de desenvolver projetos, de ver o todo e suas diferentes partes, de modelar processos/sistemas/produtos de engenharia.

# Capítulo 6

## Implementação

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

**Nota:** os códigos devem ser documentados usando padrão **javadoc**. Posteriormente usar o programa **doxygen** para gerar a documentação no formato html.

- Veja informações gerais aqui <http://www.doxygen.org/>.
- Veja exemplo aqui <http://www.stack.nl/~dimitri/doxygen/manual/docblocks.html>.

**Nota:** ao longo deste capítulo usamos inclusão direta de arquivos externos usando o pacote *listings* do L<sup>A</sup>T<sub>E</sub>X. Maiores detalhes de como a saída pode ser gerada estão disponíveis nos links abaixo.

- [http://en.wikibooks.org/wiki/LaTeX/Source\\_Code\\_Listings](http://en.wikibooks.org/wiki/LaTeX/Source_Code_Listings).
- <http://mirrors.ctan.org/macros/latex/contrib/listings/listings.pdf>.

### 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 `CAplicacao`.

Listing 6.1: Arquivo de cabeçalho da classe `CAplicacao`.

---

```
1 // Este programa exemplifica a estrutura de um programa típico em C
  ++.
2 // Note que no arquivo .h documentamos a interface; a forma de uso;
3 // No arquivo .cpp detalhes dos códigos; lógica numérica-
  computacional.
4 #include <string>
5 #include <vector>
```

```
6
7/** Breve descrição da classe termina com ponto.
8 * ...descrição detalhada da classe...
9 * ...pode ter várias linhas...
10**/
11class CAplicacao
12{
13public:
14    /// Descrição breve do construtor.
15    /** Descrição detalhada do construtor.
16     * ....blablabla....
17     */
18    CAplicacao()        {};
19
20    /// Descrição breve do construtor.
21    /** Descrição detalhada do construtor.
22     * ....blablabla....
23     */
24    ~CAplicacao()       {};
25
26    /// Apenas exibe mensagem na tela.
27    void Run();
28
29    /// Seta valor de x
30    void X( int _x)      { x = _x; }
31
32    /// Retorna valor de x
33    int X()              { return x; }
34
35private:
36    /// Descrição breve do método M1.
37    /**
38     * Descrição detalhada....
39     * Posso incluir informações sobre parâmetros e retorno.
40     * @param a um inteiro que representa ....
41     * @param s uma string que representa ....
42     * @return retorna ...
43     */
44    int M1(int a, std::string s);
45
46    /// Descrição breve do atributo...
47    /** Descrição detalhada do atributo... */
```



---

```

48     std::vector<int> vy;
49
50     /// Descrição breve do atributo...
51     int x;
52
53     int z; ///< Descrição breve (use apenas se for bem curta!).
54
55     /// Enum representa (descrição breve).
56     /** Descrição detalhada. */
57     enum Enum {
58         EVal1, ///< Breve descrição EVal1.
59         EVal2, ///< Breve descrição EVal2.
60         EVal3  ///< Breve descrição EVal3.
61     } ;
62
63     /// Descrição breve.
64     /** Descrição detalhada. */
65     Enum    variavelDoTipoEnumeracao;
66
67 };

```

---

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

Listing 6.2: Arquivo de implementação da classe `CAplicacao`.

---

```

1// Este programa exemplifica a estrutura de um programa típico em
  C++
2#include <iostream>
3
4// Inclui a declaração da classe
5#include "CAplicacao.h"
6
7/** Note que no arquivo .cpp não é necessário colocar novamente a
  documentação
8 * que foi colocada no arquivo .h.
9 * A documentação no arquivo .cpp costuma usar o padrão básico de C
  ++ que é //
10 * e costuma estar mais diretamente relacionada a implementação em
  sí,
11 * ou seja, aos detalhes numéricos e computacionais;
12 * detalhes e explicação das contas e da lógica computacional.
13 * */
14void CAplicacao::Run()
15{

```

```
16 // std::cout escreve na tela o texto "Bem-vindo ao C++!"
17 std::cout << "Bem-vindo_ao_C++!" << std::endl;
18 }
```

Apresenta-se na listagem 6.3 o programa que usa a classe `CAplicacao`.

Listing 6.3: Arquivo de implementação da função `main()`.

```
1
2 /** Este programa exemplifica a estrutura/layout de um programa
   típico em C++ */
3
4 // Inclui o arquivo "CAplicacao.h" que tem a declaração da classe
   CAplicacao
5 #include "CAplicacao.h"
6
7 /// A função main(), retorna um inteiro, se chama main() e nao tem
   nenhum parametro
8 int main ()
9 {
10  CAplicacao ap; // Cria objeto do tipo CAplicacao com nome ap
11
12  ap.Run ();      // Executa o método Run() do objeto ap
13
14  return 0;       // A função main() deve retornar um inteiro
15                  // o zero indica que o programa terminou bem.
16 }
1 Bem vindo ao C++!
```

#### Nota:

Não perca de vista a visão do todo; do projeto de engenharia como um todo. Cada capítulo, cada seção, cada parágrafo deve se encaixar. Este é um diferencial fundamental do engenheiro em relação ao técnico, a capacidade de desenvolver projetos, de ver o todo e suas diferentes partes, de modelar processos/sistemas/produtos de engenharia.

# Capítulo 7

## Teste

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

### 7.1 Teste 1: Descrição

No início apresente texto explicativo do teste:

- O que esta sendo testado?
- Como o teste vai ser realizado?
- Como o programa será validado?

A seguir apresente texto explicando a sequência do teste e imagens do programa (captura de tela).

coloque aqui texto falando do diagrama de pacotes, referencie a figura. Veja Figura 7.1.

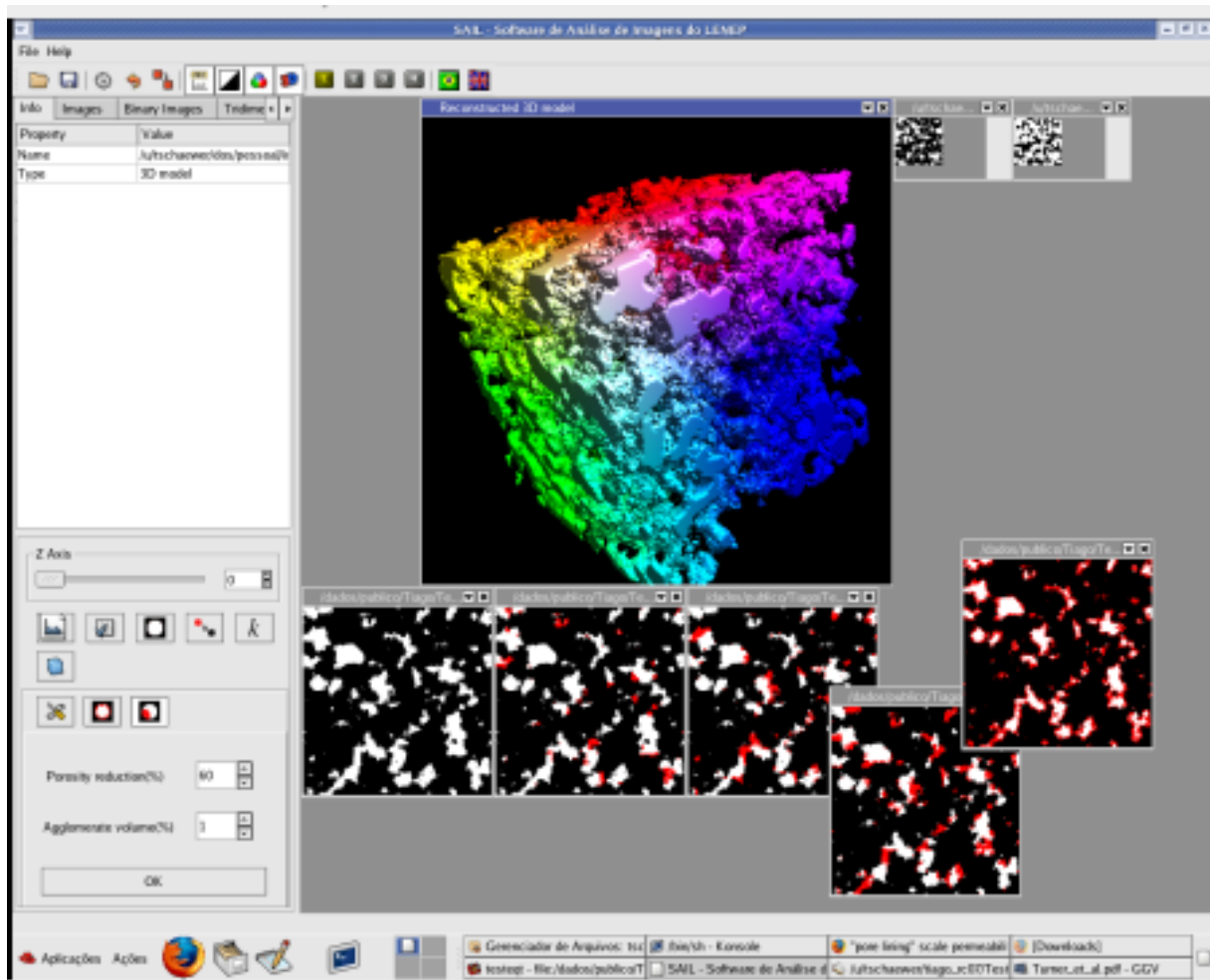


Figura 7.1: Tela do programa mostrando xxx

## 7.2 Teste 2: Descrição

No início apresente texto explicativo do teste:

- O que esta sendo testado?
- Como o teste vai ser realizado?
- Como o programa será validado?

A seguir apresente texto explicando a sequência do teste e imagens do programa (captura de tela).

Coloque aqui texto falando do diagrama de pacotes, referencie a figura. Veja Figura 7.2.

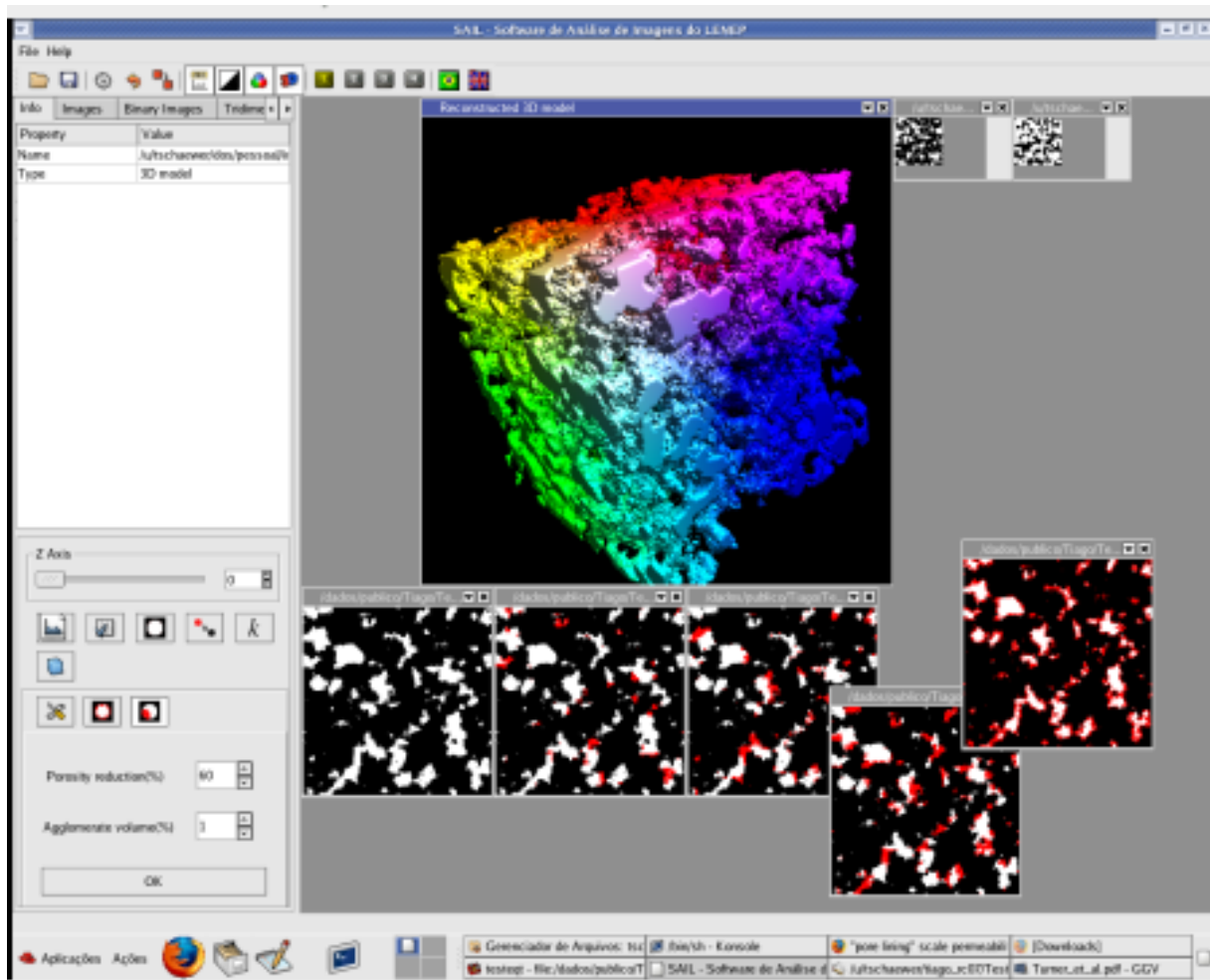


Figura 7.2: Tela do programa mostrando xxx

### Nota:

Não perca de vista a visão do todo; do projeto de engenharia como um todo. Cada capítulo, cada seção, cada parágrafo deve se encaixar. Este é um diferencial fundamental do engenheiro em relação ao técnico, a capacidade de desenvolver projetos, de ver o todo e suas diferentes partes, de modelar processos/sistemas/produtos de engenharia.

# Capítulo 8

## Documentação

Todo projeto de engenharia precisa ser bem documentado. Neste sentido, apresenta-se neste capítulo a documentação de uso do "software XXXX". Esta documentação tem o formato de uma apostila que explica passo a passo como usar o software.

### 8.1 Documentação do usuário

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 instalar o software

Para instalar o software execute o seguinte passo a passo:

- blablabla
- ..
- .

#### 8.1.2 Como rodar o software

Para rodar o software ....blablabla

Veja no Capítulo 7 - Teste, exemplos de uso do software.

### 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>. Para instalar no GNU/Linux use o comando `yum install gcc`.
- Biblioteca CGnuplot; os arquivos para acesso a biblioteca CGnuplot devem estar no diretório com os códigos do software;
- O software `gnuplot`, disponível no endereço <http://www.gnuplot.info/>, deve estar instalado. É possível que haja necessidade de setar o caminho para execução do `gnuplot`.
- .
- .

### 8.2.2 Como gerar a documentação usando doxygen

A documentação do código do software deve ser feita usando o padrão JAVADOC, conforme apresentada no Capítulo - Documentação, do livro texto da disciplina. Depois de documentar o código, use o software `doxygen` 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.

- Veja informações sobre uso do formato JAVADOC em:
  - <http://www.stack.nl/~dimitri/doxygen/manual/docblocks.html>
- Veja informações sobre o software `doxygen` em
  - <http://www.stack.nl/~dimitri/doxygen/>

Passos para gerar a documentação usando o `doxygen`.

- Documente o código usando o formato JAVADOC. Um bom exemplo de código documentado é apresentado nos arquivos da biblioteca CGnuplot, abra os arquivos `CGnuplot.h` e `CGnuplot.cpp` no editor de texto e veja como o código foi documentado.
- Abra um terminal.
- Vá para o diretório onde está o código.

```
cd /caminho/para/seu/codigo
```

- Peça para o `doxygen` gerar o arquivo de definições (arquivo que diz para o `doxygen` como deve ser a documentação).

```
doxygen -g
```

- Peça para o `doxygen` gerar a documentação.

```
doxygen
```

- Verifique a documentação gerada abrindo o arquivo `html/index.html`.

```
firefox html/index.html
```

ou

```
chrome html/index.html
```

Apresenta-se a seguir algumas imagens com as telas das saídas geradas pelo software `doxygen`.

**Nota:**

Não perca de vista a visão do todo; do projeto de engenharia como um todo. Cada capítulo, cada seção, cada parágrafo deve se encaixar. Este é um diferencial fundamental do engenheiro em relação ao técnico, a capacidade de desenvolver projetos, de ver o todo e suas diferentes partes, de modelar processos/sistemas/produtos de engenharia.





# Referências Bibliográficas

- [Blaha and Rumbaugh, 2006] Blaha, M. and Rumbaugh, J. (2006). *Modelagem e Projetos Baseados em Objetos com UML 2*. Campus, Rio de Janeiro. 5, 19
- [e Patrick W. Daly, 1995] e Patrick W. Daly, H. K. (1995). *A Guide to Latex 2e*. Addison-Wesley, New York, 2 edition. 42
- [Grossens et al., 1993] Grossens, M., Mittelbach, F., and Samarin, A. (1993). *Latex Companion*. Addison-Wesley, New York. 42
- [Karger, 2004] Karger, A. (2004). *O Tutorial de Lyx*. LyX Team - <http://www.lyx.org>. 42
- [Knuth, 1986] Knuth, D. E. (1986). *The TeXbook*. Addison-Wesley. 42
- [Lamport, 1985] Lamport, L. (1985). *Latex - A Document Preparation System*. Addison-Wesley. 42
- [LyX-Team, 2004a] LyX-Team, editor (2004a). *Extended LyX Features*. LyX Team - <http://www.lyx.org>. 42
- [LyX-Team, 2004b] LyX-Team, editor (2004b). *The LyX User's Guide*. LyX Team - <http://www.lyx.org>. 42
- [Rumbaugh et al., 1994] Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., and Lorensen, W. (1994). *Modelagem e Projetos Baseados em Objetos*. Edit. Campus, Rio de Janeiro. 19
- [Steding-Jessen, 2000] Steding-Jessen, K. (2000). *Latex demo: Exemplo com Latex 2e*. 42

# Capítulo 9

## Título do Apêndice

Descreve-se neste apêndice ...

- Os anexos ou apêndices contém material auxiliar. Por exemplo, tabelas, gráficos, resultados de experimentos, algoritmos, códigos e simulações.
- Um apêndice pode incluir assuntos mais gerais (geral demais para estar no núcleo do trabalho) ou mais específicos (detalhado demais para estar no núcleo do trabalho).
- Pode conter um artigo de auxílio fundamental ao trabalho.
- Pode conter artigos publicados.
- [tudo aquilo que for importante para a tese mas não essencial, deve ser colocado em apêndices]
- [como exemplo, revisão de metodologias, técnicas, modelos matemáticos, itens desenvolvidos por terceiros]
- [algoritmos e programas devem ser colocados no apêndice]
- [imagens detalhadas de programas desenvolvidos devem ser colocados no apêndice]

### 9.1 Sub-Título do Apêndice

.....conteúdo..

# Capítulo 10

## Título do Apêndice

Descreve-se neste apêndice ...

[tudo aquilo que for importante para a tese mas não essencial, deve ser colocado em apêndices]

[como exemplo, revisão de metodologias, técnicas, modelos matemáticos, itens desenvolvidos por terceiros]

[algoritmos e programas devem ser colocados no apêndice]

[imagens de programas desenvolvidos/utilizados devem ser colocados no apêndice]

## 10.1 Roteiro Para Uso do Sistema de Citações Com Banco de Dados .bib

O sistema de referências usando bibtex é extremamente simples e muito prático. O mesmo é composto de uma base de dados (um arquivo .bib que contém a lista de referências a ser utilizada). Por exemplo, o arquivo andre.bib, inclui referências bibliográficas no formato bib (de uma olhada agora no arquivo andre.bib usando um editor de texto como o emacs). A seguir, você deve incluir no arquivo do lyx, o nome de sua base de dados. Finalmente, você precisa incluir as referências cruzadas.

Veja a seguir um roteiro:

1. Você deve fazer uma cópia do arquivo andre.bib com seu nome, e a seguir usar um editor qualquer (mas preferencialmente o emacs) para incluir suas referências bibliográficas. Ou seja, inclua no arquivo seuNome.bib todas as citações e referências bibliográficas a serem incluídas em sua tese (tudo que você leu, e que pode ser incluído na citação da tese e de outros artigos. É sua base de dados de citações).
  - (a) Você pode incluir itens no arquivo .bib que não irão fazer parte da tese, mas poderão ser citadas em artigos futuros.
2. Para fazer uma citação é necessário incluir no arquivo do lyx um "Insert-> Lists & Toc->Bibtex reference". Vai aparecer um diálogo pedindo para você incluir o nome do arquivo com a base de dados de citações (digite seuNome.bib).
3. Finalmente, faça referências cruzadas usando o item de menu "Insert Cross-Reference".
4. Aqui um exemplo, vou citar material sobre LyX e Latex. Veja maiores informações sobre latex em [Grossens et al., 1993, Knuth, 1986, Steding-Jessen, 2000, e Patrick W. Daly, 1995, Lamport, 1985, LyX-Team, 2004a, Karger, 2004, LyX-Team, 2004b].

### 10.1.1 Citações no meio do texto

### 10.1.2 Incluir nas referências bibliográficas (fim do documento), mas não citar

asldkjasldkajsdllkajsdllaksjd  
asldkjasldkajsdllkajsdllaksjd

## 10.2 Informações adicionais

- Manuais do LyX (precisa ler!)
- <http://chem-e.org/comando-cite-e-citeonline-no-abntex/>
- <http://win.ua.ac.be/~nschloe/content/bibtex-how-cite-website>.
- <http://chem-e.org/comando-apud-e-apudonline-no-abntex/>.
- [http://en.wikibooks.org/wiki/LaTeX/Bibliography\\_Management](http://en.wikibooks.org/wiki/LaTeX/Bibliography_Management)

...

# Índice Remissivo

- Análise de domínio, 9
- Análise orientada a objeto, 13
- AOO, 13
- Associações, 24
- Assuntos, 11
- assuntos, 12
- atributos, 22
- Casos de uso, 5, 6
- casos de uso, 6
- Cenário, 5
- Citações, 42
- Citações no meio do texto, 43
- colaboração, 15
- comunicação, 15
- Concepção, 3
- Controle, 20
- Diagrama de caso de uso, 6
- Diagrama de colaboração, 15
- Diagrama de componentes, 25
- Diagrama de execução, 26
- Diagrama de máquina de estado, 15
- Diagrama de pacotes, 12
- Diagrama de pacotes (assuntos), 12
- Diagrama de sequência, 13
- Efeitos do projeto nas associações, 24
- Efeitos do projeto nas heranças, 23
- Efeitos do projeto nos métodos, 23
- Elaboração, 9
- especificação, 3, 4
- Especificações, 3
- estado, 15
- Eventos, 13
- Heranças, 23
- heranças, 23
- Identificação de pacotes, 11
- Implementação, 28
- métodos, 23
- módulos, 12
- Mensagens, 13
- modelo, 22
- otimizações, 24
- Plataformas, 21
- POO, 21
- Projeto do sistema, 19
- Projeto orientado a objeto, 21
- Protocolos, 19
- Recursos, 20