Guilherme Pasqualette – 1610596

Resumo da matéria de Programação Modular

1 – ***INTRODUÇÃO***

- *Vantagens da programação modular:*
  - Ela vence barreiras de complexidade de um software;
  - Facilita o trabalho em grupo, após divisão das tarefas no grupo;
  - Possibilita o reuso;
  - Facilita a criação de um acervo, já que diminui a quantidade de novos programas;
  - Desenvolvimento incremental;
  - Aprimoramento individual de módulos;
  - Facilita a administração de baselines, que são versões "bases" do programa que funcionam de acordo com o pedido, as quais os programadores podem utiliza-las novamente caso tenha algum problema com uma versão posterior.

2 – ***PRINCÍPIOS DE MODULARIDADE***

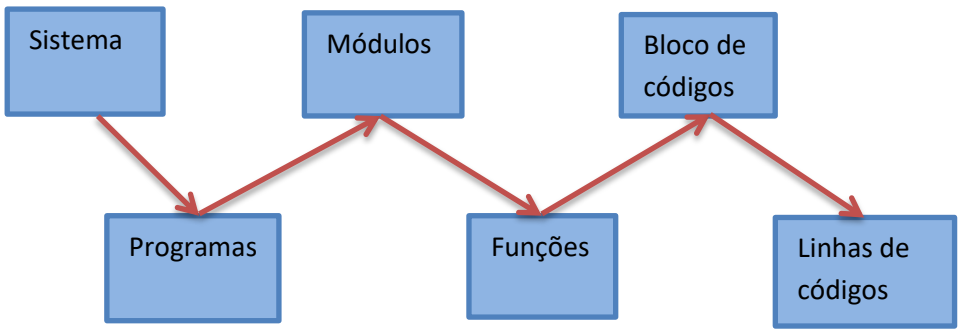
I) Módulo:

- A definição física de módulo consiste nele ser uma unidade de compilação independente (.c);
- E a definição lógica é baseada em um único conceito o qual o módulo trata.

II) Abstração de Sistema:

- Abstração é o procedimento de considerar somente o que é preciso ou não em diversas situações e, o que descartar com segurança nas mesmas situações.

i) **Níveis de abstração**:



Alguns conceitos importantes:

1) Artefato: Item com identidade própria, criado dentro de um processo de desenvolvimento podendo possuir baselines;

2) Construto (build): Artefato que pode ser executado, estando incompleto ou não.

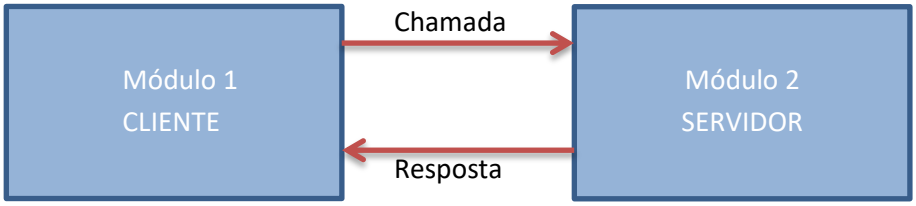
III) Interface:

- É o mecanismo de troca de dados, estados, eventos entre elementos de um mesmo nível de abstração;

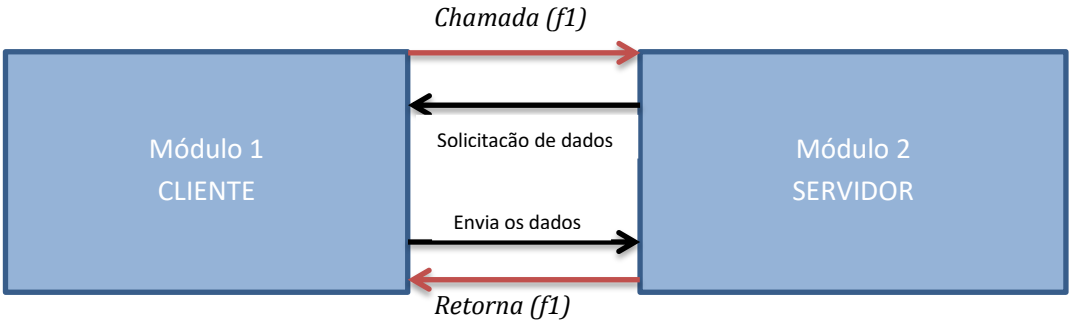
i) <u>Exemplo:</u>

- Arquivo entre sistemas
- Funções de acesso entre módulos
- Passagem de parâmetro entre funções
- Variável global entre blocos

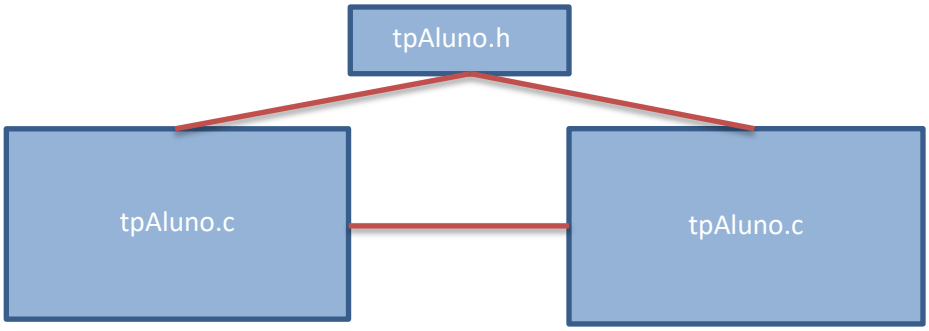
ii) <u>Relacionamento cliente-servidor:</u>



Caso Especial: Callback -> É quando o servidor, após o cliente enviar dados para ele, retorna uma resposta para o usuário, avisando sobre uma possível falta de certos dados essenciais para o funcionamento do programa.



iii) <u>Interface fornecida por terceiros:</u>



Importante! Evitar duplicidade de código!!!

iv) <u>Interface Em Detalhe:</u>

- Todo resultado trazido pelo compilador não depende propriamente dos blocos de códigos dentro dos módulos, mas também depende da sintaxe (regras) e da semântica (significado) inserido neles.

Exemplo:

- Sintaxe: Jamais poderá pedir para uma função do tipo Arvore pedir para retornar um valor Int. Isso resultará em um erro de compilação.

- Semântica: Quando possui duas variáveis (v1 & v2) do tipo Float, em dois módulos diferentes (m1 & m2), uma pra área e outra tempo, respectivamente, não se pode utilizar uma delas no módulo onde não pertencem, pois o programa funcionará e, provavelmente, dará um resultado equivocado para o cliente.

v) <u>Análise de Interface:</u>

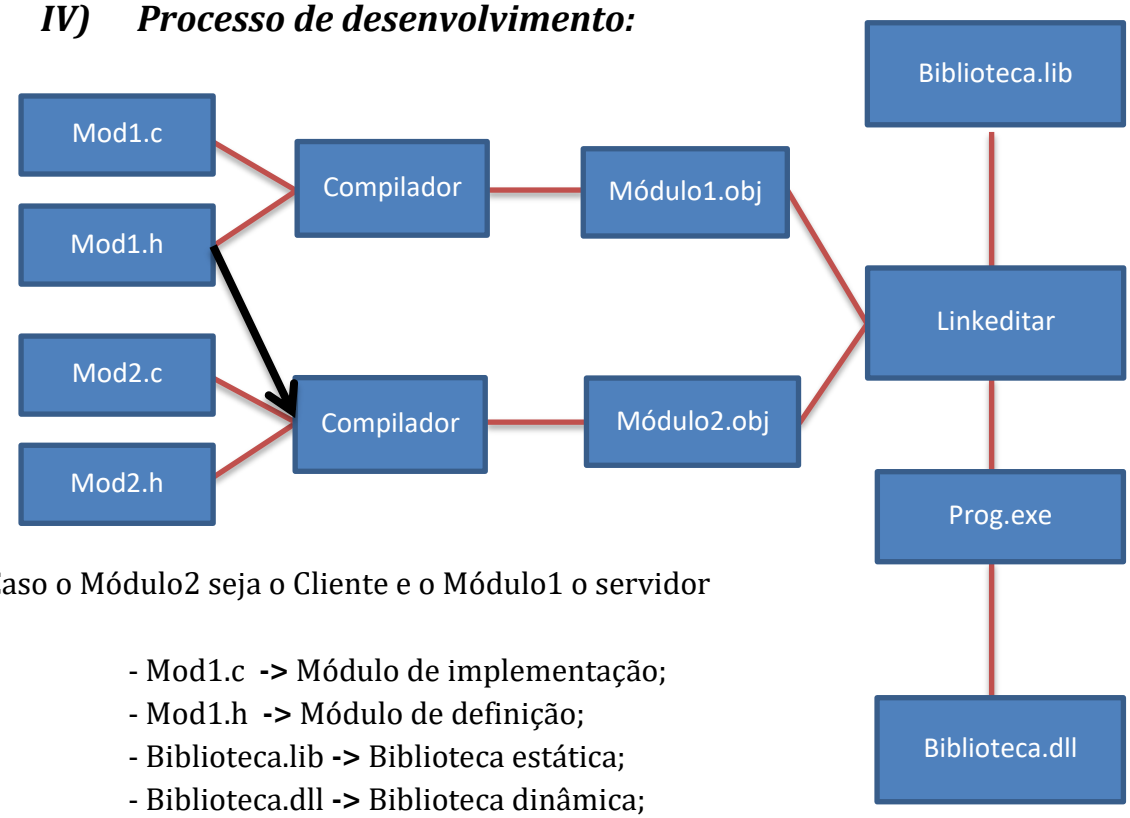
No caso de um protótipo de função de acesso: tpAluno *obterMatricula (int Mat):

- A interface esperada pelo cliente **->** Ponteiro para os dados válidos do aluno correto ou NULL;

- A interface esperada pelo servidor **->** Valor inteiro apto para representar a matrícula do aluno;

- A interface esperada pelo servidor e cliente **->** tpAluno.

IV) Processo de desenvolvimento:



Caso o Módulo2 seja o Cliente e o Módulo1 o servidor

- Mod1.c **->** Módulo de implementação;
- Mod1.h **->** Módulo de definição;
- Biblioteca.lib **->** Biblioteca estática;
- Biblioteca.dll **->** Biblioteca dinâmica;

V) Bibliotecas estáticas e dinâmicas:

i) <u>Estática:</u>

-Vantagem **->** Ela já é acoplada, em termo de linkedição, à aplicação executável;

-Desvantagem **->** Existe uma cópia dessa biblioteca para cada executável alocado na memória que a use;

ii) <u>Dinâmica:</u>

- Vantagem **->** Só é carregada à uma só instancia;

- Desvantagem **->** A DLL precisa estar na máquina, preferencialmente do cliente, para o executável funcionar;

VI) Módulo de definição(.h):

- Interface do módulo;

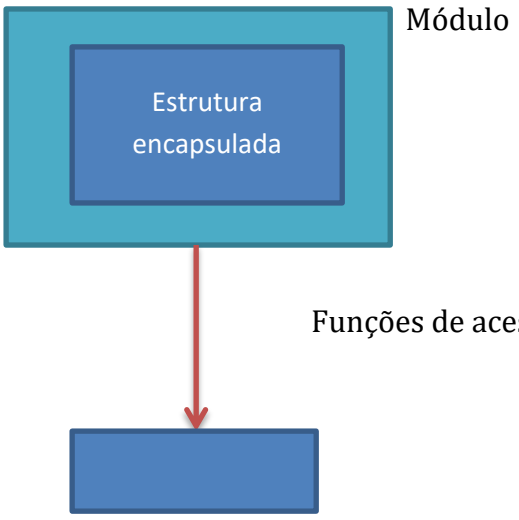
-Os protótipos das funções de acesso estão contidos nele, é uma interface fornecida por terceiros (tpAluno);

- Sua documentação é voltada para o implementador do módulo Cliente;

VII) Módulo de implementação:

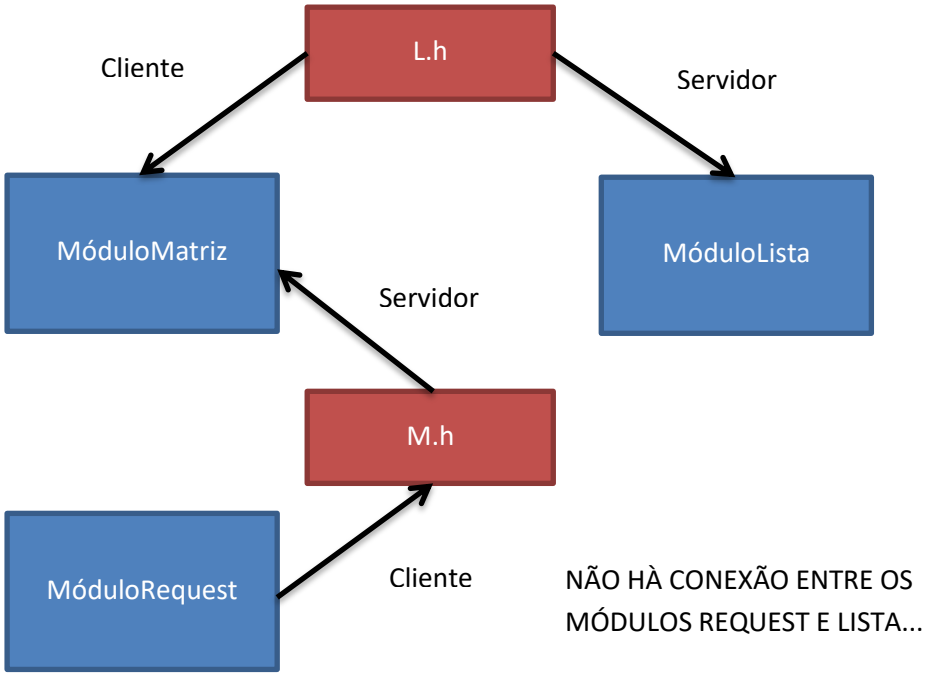
- Contém o código das funções de acesso;
- Códigos e protótipos de funções próprias do módulos (static);
- Contém variáveis internas ao módulo;
- Documentação voltada para o implementador do módulo servidor;

VIII) Tipo Abstratos de Dados:



Uma estrutura encapsulada em um módulo é apenas conhecida pelos módulos clientes a partir de funções de acesso disponibilizadas na interface.

Exemplo:
- Para um módulo que manipula uma matriz utiliza listas para criar suas colunas e linhas, o esquema estaria de acordo com o pedido:



Nesse caso, a solução seria utilizar ponteiros para a "cabeça" da estrutura. Logo, as funções de acesso tem que receber tal ponteiro a fim de indicar qual estrutura de dados será modificada, com isso, interface deverá disponibilizar um typedef para corresponder com o ponteiro.

IX) Encapsulamento:

- Definição: Proteção de elementos que compõem módulos.

i) <u>Objetivos</u>

- Facilitaria a manutenção por manter os erros confinados;
- Impediria a utilização ou alteração indevida da estrutura do módulo;

ii) <u>Outros tipos de encapsulamento:</u>

- De documentação
  - **->** Interna do módulo de implementação;
  - **->** Externa do módulo de definição;
  - **->** De uso do manual do usuário (README);
- De código – Para blocos de códigos visíveis somente
  - **->** Dentro do módulo;
  - **->** Dentro do outro bloco de código;
  - **->** Código de uma função;

- De variáveis

-> Private: Encapsulado no objeto;

-> Static: Encapsulado no módulo (Ou na classe no caso de Orientação a Objetos);

-> Local: Em um bloco de código;

X) Acoplamento:

- Propriedade relacionada com a interface entre os módulos.

i) <u>Conector:</u> Item de interface, a partir de funções de acesso e variáveis globais.

ii) <u>Critérios de Qualidade:</u>

- Quantidade de conectores **->** Necessidade X Suficiência

----- Há excesso ou falta deles? -----

- Tamanho do conector **->** Quantidade de parâmetros de uma função
- Complexidade do conector **->** Tem que haver explicação na documentação e utilização de mnemônios.

XI) Coesão:

- Propriedade relacionada com o grau de interligação dos elementos que compõem um módulo.

i) <u>Níveis de coesão:</u>

- Incidental **->** Pior coesão, pois não há relação praticamente entre os elementos;
- Lógico **->** Elementos logicamente relacionados;
- Temporal **->** Itens que funcionam em um mesmo período de tempo;
- Procedural **->** Itens em sequencia;
- Funcional **->** Abstração de dados, torna-los em um único conceito (TAD).

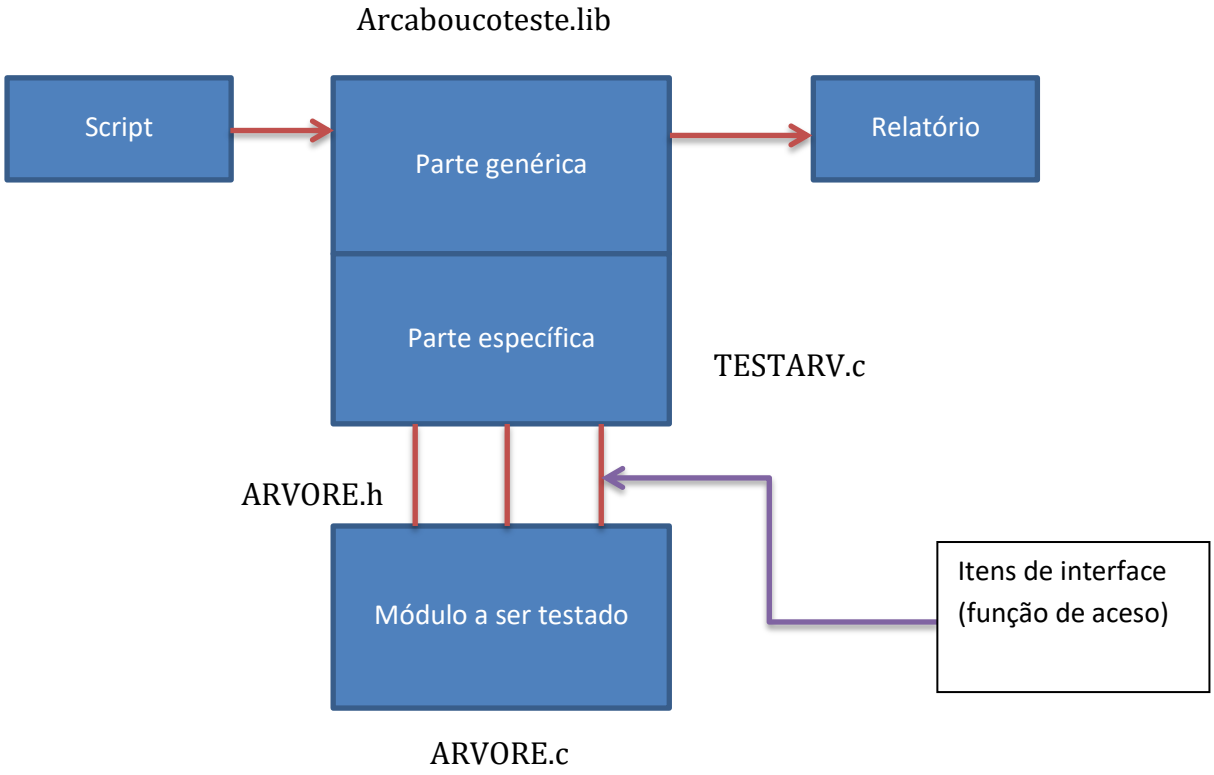
3 - ***TESTE AUTOMATIZADO***

I) Objetivo:

- Testar de forma automática um módulo recebendo um conjunto de casos de teste na forma de um script e gerando um relatório de saída com a análise entre o resultado esperado e o obtido.

OBS: A partir do primeiro retorno esperado diferente do obtido no relatório de saída, todos os resultados de execução de casos de teste não serão confiáveis.

II) Framework de testes:



III) Script de teste:

- "==" indica um determinado teste em uma situação;

- "=" indica um comando de teste, diretamente associado a uma função de acesso;

OBS: O teste completo consiste em testar todos os casos de teste para todas as condições de retorno de cada função de acesso do módulo (exceto para condições de estouro de memória).

IV) Relatório de saída:

== Caso 1

== Caso 2

== Caso 3

1 >> Função esperava 0 e retorna 1

0 << <- Recuperar (Parte genérica).

V) Parte específica:

- A parte específica, que necessita ser implementada a fim de que o framework (arcabouço) possa acoplar na aplicação, chama-se **HOTSPOT**.

Por exemplo: O módulo de implementação testArvore.c.

4 – PROCESSO DE DESENVOLVIMENTO DE ENGENHARIA DE SOFTWARE

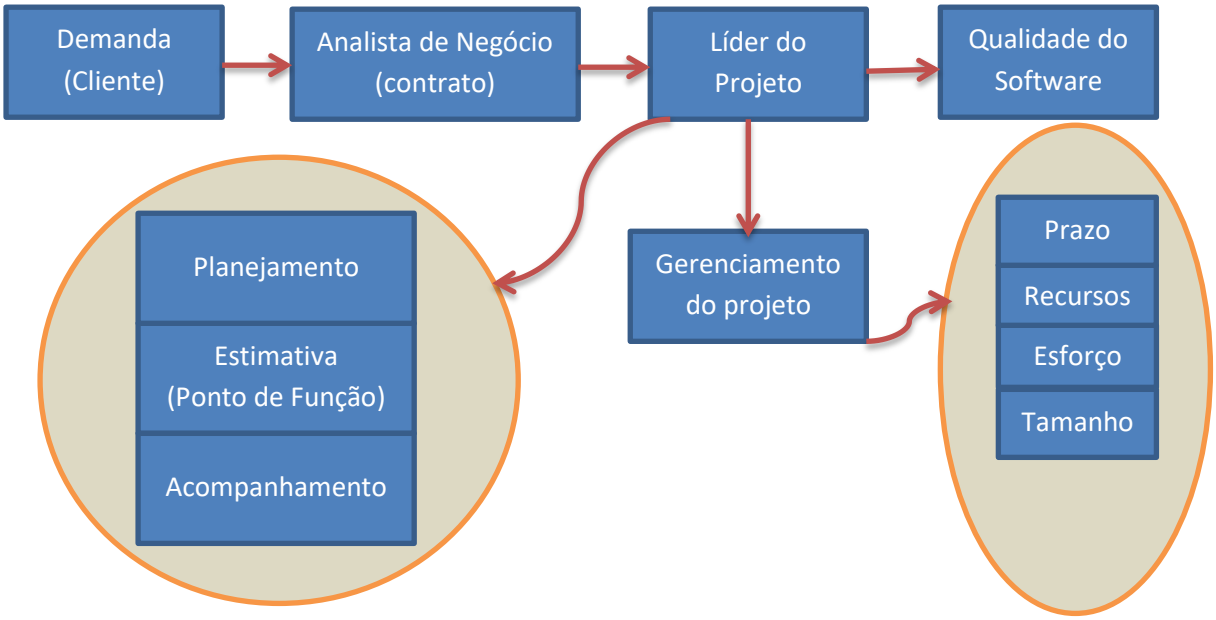
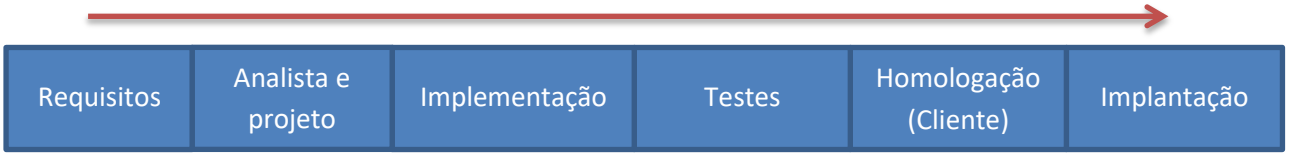


Gráfico sobre as etapas de um projeto:



I) Requisitos:

- Elicitação (Pegar informações com o cliente necessárias para o projeto).
- Documentação (Sem ambiguidades).
- Verificação (O que é possível ou não implementar).
- Validação

II) Análise e projeto:

- Projeto Lógico.
- Projeto Físico.

III) Implementação:

- Programas.
- Teste unitário (Do próprio implementador).

IV) Testes:

- Teste integrado, com todos os casos de erros “forçados”, devendo resultar em um programa sem erros.

V) Homologação:

- Sugestão: Adicionamento de mais especificações **->** Retrabalho
- Erro: Especificação do cliente que a equipe não cumpriu.