----------Príncipios de modularidade----------

1-Modulo

A)Def física: Unidade de compilação independente

B)Def Logica: Trata de um unico conceito

2-Abstração de sistema

Abstrair:

Processo de considerar apenas o que é necessario numa situalão e descaretar com segurança o que não é necessario.

O res de uma abstração é um escopo

Nivel de abstração:

Sistem > Prog > Modulos > Funções > Blocos de Código > Linhas de codigo

OBS: Artefato - Um item de identidade propria criado dentro de um processo, coisas que não podem ser versionadas

Construto(build) - Primeiro resultado apresentavel, mesmo incompleto

3-Interface

Mecanismo de troca de dados, estados e eventos de um mesmo nivel de abstração

a)Exs de interface

>Arqv - Entre sis

>Func de acesso - Entre mod

>Passagem de param - Entre func

>Var globais - Entre blocos

B)Relacionamento cliente-server

(Desenhos feitos a parte)

C)Interface fornecida por terceiros

Depende de um terceiro componente para fazer dois modulos conversarem

(Desenho a parte)

D)Interface em detalhe

>Sintaxe - Regras

>Semantica - Significado

E)Analise de interface

-Cliente interface: Ponteiro para dados validos ou NULL

-Server interface: Int válido

Ambos: Int ja conhecido

4-Processo de desenvolvimento

(Desenhos no caderno)

5-Bibliotecas estaticas e dinamica

-->Estatica:

>Vantagem:

Lib ja acoplada

>Desvantagem:

Existe uma copia dessa biblioteca estatica para cada exe na memoria

Aumenta o tamanho do programa

-->Dinamica:

>Vantagem

So carrega uma instancia de biblioteca dinamica na memoria

>Desvantagem:

a dll precisa estar na maquina

6-Modulo de definição - .h

>Interface do modulo

>Contem prototipos das funções de acesso

>Documentação voltada para o programador do mod cliente

7-Modulo de implementação - .c

>Codigo das func de acesso

>Codigos e prototipos da func internas

>Var internas

8-Tipo abstrato de dados (TAD)

Uma estrutura encapsulada em um modulo que somente é conhecida pelos mod clientes atraves de func de acesso na intertface

Ex: Se um mod manipula uma matriz usa listas para fazer suas col e linhas

(Desenho caderno)

Se o cliente precisar usar mais e uma instancia da estrutura dada pelo TAD, uma solução é trabalhar com ponteiros para a cabeça da estrutura

9-Encapsulamento

Objetivo:

Facilitar a manutenção

Impedir utilização ou modificação indevida de estrutura de mod

-->Outros tipos de encapsulamento:

Documentação:

>Documentação interna: Implementação .c

>Documentação externa: Definição .h

>Documentação de uso: README

Codigo:

>Blocos de codigos visiveis apenas no modulo ou dentro de outro bloco de codigos

>Codigo de uma função

Variaveis:

>Private, public,...

10-Acoplamento

Propriedade relacionada com a interface entre modulos

-->Conector: Item da interface

>Função de acesso

>Var global

-->Criterios de qualidade

>Quantidade de conectores

>Tamanho do conector

>Complexidade do conector

11-Coesão

Propriedade relacionada com o grau de interligação dos elementos que compõe o modulo

-->Niveis de coesão

>Incidental - pior

>Lógica - elementos logicamente relacionados

>Temporal - itens que funcionam em um mesmo periodo de tempo

>Procedural - itens em sequencia (ex: .bat)

>Funcional

>Abstração de tados - melhor coesão, trabalha um unico conceito (ex: TAD)

----------Teste Automatizado----------

1-Objetivo

Testar de forma automatica um modulo recebendo um conjunto de casos de teste na forma de um script e gerando um log de saida com a analise entre o que é esperado e o obtido

OBS:Apartir do primeiro retorno esperado diferente do obtido no log de saida todos os resultados de exec de casos de teste não são confiaveis

2-FRAMEWORK de teste

(Desenhos no caderno)

3-Script de teste

// - > comentario

== caso de teste - > testa determinada situação

= comando de teste ->associado a uma função de acesso

obs: teste completo -> caso de teste para todas as condiçoes de retorno de cada função de acesso do modulo

4-Log de saida

== caso 1

== caso 2

== caso 3

1>> função espera 0 e retorna 1

0<< Recuperar

5-Parte especifica

A parte especifica que necessita ser implementada para que o framework(arcabouço) possa acoplar na aplicação chama-se hotspot

ex: testarv.c

#define criar_arv CMD "=Criar"

efetuar comando(comando teste)

if(strcmp(comando teste,Criar Arv CMD)

numlidos = ler_leparametros("i", &com ret esperada)

obs: ret esperada = 0

if(Numlidos != 1)

return TST_Condret

Cond ret obtido = Arv_grav

Cont ret obtida //Erro ao criar arv

----------Processo de desenvolvimento em engenharia de software----------

(Desenho caderno)

>Lider do projeto

>Gerencia de configuração

>Qualidade de software

(Desenho caderno)

Requisitos:

>Elicitação: Entrar em contato com o cliente e ver o que é necessario

>Documentação

>Verificação

>Validação

Analise de projeto:

>Projeto logico

>Projeto fisico

Implementação:

>Programas

>Concretiza a linguagem de programação

>Teste unitário

Testes:

>Teste integrado

Homologação:

>Sugestão: não estava nas especificações, adicionar especificações

>Erro: estava nas especificações, a equipe não cumpriu

----------Especificação de requisitos----------

1-Definição de requisitos

O que deve ser feito.
Não como deve ser feito.

Requisito tem que ser uma frase só. ser objetiva, clara e certeira.

2-Escopo de requisitos

>Requesitos mais gerais/genericos

>Requesitos mais específicos

3-Fases da especificação

A)Elicitação

Adaptar informações do cliente para realizar a documentação do sistema do sistema a ser desenvolvido.

>Entrevista (o que é necessário para gerar o sistema, a dificuldade é filtrar o que é importante e oq perguntar)(só ajuda o cliente a decidir coisas técnicas, não decidir nada do negócio do cliente)

>Brainstorm (precisa saber controlar)

>Questionário (péssima opção)

B)Documentação

>Requisitos descritos em itens diretos quando você pega essas informações desorganizadas e põe no formato de requisitos simples

>Uso da língua natural. CUIDADO com ambiguidade

>Dividir requisitos em seus diversos tipos

-->Requisitos funcionais:

O que deve ser feito em relação a informatização das regras de negócio

-->Requisitos não funcionais:

São propriedades que a aplicação deve ter e que não estão diretamente relacionadas com as regras de negócio (como por login e senha para segurança)

-->Requisitos inversos:

É o que não é para fazer, quem define os requisitos inversos é o analista de requisitos e não o cliente

C)Verificação

A equipe técnica verifica se o que está descrito na documentação é viável de ser desenvolvido

D)Validação

Cliente valida a documentação

4-Exemplos de requisitos

A)Bem formulados

>A tela de resposta da consulta de aluno apresenta nome e matrícula

>Todas as consultas devem retornar respostas em no máximo 2 segundos

B)Mal Formulado

>O sistema é de fácil utilização

>A consulta deverá retornar uma resposta em um tempo reduzido

>A tela mostra seus dados mais importantes

----------Modelagem de dados

Modelos:

(Desenho no caderno)

Exemplos:

A)Vetor de 5 posicoes que armazenam inteiros:

(Desenho no caderno)

B)Arvore binaria com cabeça que armazena caracteres:

(Desenho no caderno)

C)Lista duplamente encadeada com cabeça que armazena caracteres:

(Desenho no caderno)

Assertivas estruturais:

São regras utilizadas para desempatar dois modelos iguais. Estas regras complementam o modelo, definindo características que o desenho não consegue representar.

Lista:

Se pCorr→ pAnt != nulo entao pCorr→ pAnt→ pProx == pCorr

Se pCorr→ pProx != nulo entao pCorr→ pProx→ pAnt == pCorr

Árvore:

>Um ponteiro de uma subárvore à esquerda nunca referencia um nó de uma subárvore à direita.

>pAnt e pProx de um nó nunca aponta para o pai.

D)Vetor de listas duplamente encadeadas com cabeca e genericas(void):

(Desenho no caderno)

E)Matriz tridimensional generica construida com listas duplamente encadeadas com cabeca:

(Desenho no caderno)

F)Grafo criado com listas:

(Desenho no caderno)

-->Assertivas

Definição:

Assertivas são regras consideradas válidas em determinado ponto do código(exemplo: a função recebe 3 parâmetros). Elas são compostas de texto em linguagem natural, mas podem ser implementadas em C por um printf ou um comando assert(exemplo: se não receber um int aborta)

Onde por as assertivas:

>Argumentação de corretude

>Instrumentação

>Trechos complexos em que é grande o risco de erros

Assertivas de entrada e saida:

>Entrada: São regras que devem ser verdadeiras antes da entrada em um bloco de código.(tipo: a função recebe 3 parâmetros não nulos)

>Saida: Devem ser verdadeiras no final do bloco(ex: função retorna um float)

Ex: Excluir nó corrente intermediário de uma Lista duplamente encadeada

-->Entrada:

Ponteiro corrente referencia o nó a ser excluído e este é intermediário

-->Saida:

Nó referenciado foi excluído. Nó corrente aponta para o anterior

----------Implementação da programação modular----------

1-Espaço de dados

São áreas de armazenamento:

>Alocadas em um meio (disco, memória, arquivo…)

>Possui um tamanho(malloc(sizeof))

>Possui um ou mais nomes de referência

2-Tipos de dados

Determinam a organização, codificação, tamanho em bytes, e conjunto de valores permitidos

Organização:

Formato do inteiro na memória, tipo bits de float, int, etc (formato das coisas na memória(bits))

Codificação:

Como entender um texto dividido em subdivisoes, tipo datas

3-Tipos de tipos de dados

>Tipos computacionais: int, char, char*

>Tipos básicos/personalizados: enum, typedef, union, struct

>Tipos abstratos de dados

4-Tipos Básicos

A)Typedef

Define um novo tipo explicitamente dando um nome novo a outro tipo

(Desenho no caderno)

B)Enum

enum{

texto1,

texto2,

texto3,

} tpexemplo

C)Struct

O tamanho do struct é igual à soma dos tamanhos dos seus campos

(Desenho no caderno)

D)Union

É um typecast mais organizado. Mesmo campo que pode mudar a forma de interpretação

(Desenho no caderno)

5-Declaração e definição de elementos

Definir:

>Alocar espaço

>Amarrar o espaço a um nome

Declarar:

>Associar espaço a um tipo

6-Implementação em C/C++

A)Declarações e definições de nomes globais exportados pelo módulo servidor

B)Declarações externas contidas no módulo cliente, somente declara o nome sem associá-lo a um espaço de dados

(Desenho no caderno)

7-Resolução de nomes externos

Um nome externo somente declarado em um determinado modo necessariamente deve estar declarado e definido em algum outro módulo

Obs: Colocar extern em uma variavel faz virar global

8-Pré-processamento

(Desenho no caderno)

#define nome valor

-> Substitui nome por valor

#undef nome

#if defined (nome) ou #ifdef nome

Texto verdade

#else

Texto falso

#endif

#if !defined (nome) ou #ifndef nome

#endif

#include <arquivo> -> Inclui o conteúdo texto da biblioteca

Exemplo:

#if !defined (Exemp.mod)

#define Exemp.mod

Arq.h

#endif

(Desenho no caderno)

#define EXEMP_TXT

#if !defined(EXEMP_OWN)

#define EXEMP_EXT extern

#endif

EXEMP_TXT int vet[7];

#if defined(EXEMP_OWN) = {1,2,3,4,5,6,7};

#else;

#endif

----------Estruturas de funções----------

1-Paradigma

Forma de programar:

>Procedural: Programação estruturada

>Orientada objeto

2-Estrutura de funções

(Desenho no caderno)

3-Estrutura de chamadas

(Desenho no caderno)

>F4 -> F4:

Chamada recursiva direta

>F9 -> F8 -> F3 -> F5 -> F9:

Chamada recursiva indireta

>F10:

Função morta (em outra aplicação ela pode ter

utilidade)

>F8 -> F3 -> F5 -> F6 -> F7:

Dependência circular entre módulos

>F6 -> F7:

Arco de chamada

4-Função

É uma porção auto-contida de código

Contem:

>Nome

>Assinatura

>1 ou + corpos de código

5-Especificação da função

Objetivo:

>Acoplamento

>Condições de acoplamento (assertivas)

>Interface com o usuário

>Requisitos

6-Interfaces

Conceitual:

Definição da interface da função sem preocupação com a implementação

Físico:

Implementação do conceitual

Implícito:

Dados de interface diferntes de parâmetros e valores de retorno

7-Housekeeping

Código responsável por liberar componentes e recursos alocados a programas ou

funções ao terminar a execução

8-Repetição

(Desenho no caderno)

9-Recursão

(Desenho no caderno)

10-Estado

Descritor de estado:

Conjunto de dados que define um estado

Estado:

Valoração do descritor

11-Esquema de algorítimo

O esquema de algorítmo é a parte genérica e os hotspots as partes específicas. Juntos formam o conceito de Framework

Esquemas de algorítmo permitem encapsular a estrutura de dados utilizada. É correto, independe de estrutura e é incompleto

12-Parâmetros do tipo ponteiro para função

(Desenho no caderno)

----------Decomposição sucessiva----------

1-Conceito

>Divisão e conquista

>Decomposição sucessiva é um método de divisão e conquista

2-Estrutura de decomposição

(Desenho no caderno)

Um componente abstrato se subdivide em subproblemas. Um componente concreto não se subdivide. É possível a resolução dele como está

3-Critérios de qualidade

>Complexidade

>Necessidade

>Suficiência

>Ortogonalidade

4-Passo de projeto

(Desenho no caderno)

5-Direção de projeto

(Desenho no caderno)

Bottom-Up: começa em baixo, testando e vai implementando.

Top-Down: faz o principal primeiro, pra depois ir implementando

----------Argumentação de corretude----------

(Desenho no caderno)

1-Definição

É um método utilizado para argumentar que um bloco de código está correto

2-Tipos de argumentação

>Sequência

>Seleção

>Repetição

3-Argumentação de sequência

(Desenho no caderno)

4-Argumentação de seleção

(Desenho no caderno)

5-Argumentação de repetição

(Desenho no caderno)

----------Instrumentação----------

1-Problemas ao realizar testes

Esforço de Diagnosenão saber resolver e ter que buscar a origem do erro:

>Grande

>Muito sujeito a erros

Contribuem para este esforço:

>Não estabelece com exatidão a causa a partir dos problemas observados

>Tempo decorrido entre o instante da falha e o observado

>Falhas intermitentes

>Causa externa ao código que mostra a falha

>Ponteiros loucos

>Comportamento inesperado do hardware

2-O que é argumentação

>Fragmentos inseridos nos módulos

>Não contribuem para o objetivo do programa

>Monitora o serviço enquanto o mesmo é executado

>Consome recursos de execução

>Custa para ser desenvolvida

3-Objetivos

>Detectar falhas de funcionamento do programa o mais cedo possível de forma automática

>Impedir que falhas se propaguem

>Medir propriedades dinâmicas do programa

4-Conceitos

Programa robusto:

>Intercepta a execução quando observa um problema

>Mantém o dano confinado

Programa tolerante a falhas:

>É robusto

>Possui mecanismos de recuperação

Deterioração controlada:

>Habilidade de continuar operando corretamente mesmo com uma perda de funcionalidade

5-Esquema de inclusão de instrumentos no código em C/C++

#ifdef _DEBUG

codigo da instrumentação

dados

#endif

6-Assertivas Executáveis

Vantagens:

>Informam o problema quase imediatamente após ter sido gerado

>Controle de integridade feito pela máquina

>Reduz o risco de falha humana

Precisam ser:

>Completas

>Corretas

7-Depuradores

Ferramenta utilizada para executar o código passo a passo permitindo que se distribua breakpoints com objetivo de confinar os erros a serem pesquisados

8-Trace

Instrumento utilizado para apresentar uma mensagem no momento em que é executado

9-Controlador de cobertura

(Desenho no caderno)

Instrumento composto de um vetor de contadores que tem como objetivo acompanhar os testes caixa aberta de uma aplicação monitorando todos os caminhos percorridos

10-Verificador estrutural

Instrumento responsável por realizar uma verificação completa da estrutura em questão. É a implementação de código relacionado com as assertivas estruturais e modelo

11-Depurador Estrutural

Instrumento responsável por inserir erros na estrutura com o objetivo de testar o verificador

12-Recuperador Estrutural

Instrumento responsável por recuperar automaticamente uma estrutura a partir de um erro observado em uma aplicação tolerante a falhas

13-Estrutura Auto Verificável

É a estrutura que contém todos os dados necessários para que seja totalmente verificada

É possível verificar todos os tipos apontados pela estrutura?

>Incluir um campo tipo

>No modelo tem que aparecer diferente

>Inclusão de um campo tipo no cabeça

É possível acessar qualquer parte da estrutura a partir de qualquer origem?

>Inclusão de um campo pAntes e um pCabeça