Universidade Tecnológica Federal do Paraná

Disciplina: Análise e Projeto Orientado a Objetos
Aluno: Jean Carlos Martins Miguel R.A: 1640593
Professor: Dr. Lucio Geronimo Valentin

1 Capítulo 1

1.1 Qualidade do software

A Engenharia de Software tem várias finalidades,porém o principal objetivo é a contribuição do desenvolvimento de softwares de qualidade, sendo que esses programas tem algumas qualidades tais como : eficiência, facilidade de uso, extensibilidade, que são as qualidades externas do código, que são detectados na utilização do software, pelos usuários.Já os fatores relacionados a qualidade de software interno que são legibilidade, modularidade, só são percebidos pelos profissionais da computação.

1.2 Fatores de qualidade externa

- Corretude: O programa cumpre sua finalidade de acordo com o que foi especificado anteriormente, produzindo respostas corretas, é a qualidade primordial, pois se o software não cumpre seu objetivo, então ele é irrelevante.

- Robustez: O programa é capaz de funcionar mesmo diante de condições anormais, caso apareça uma situação imprevista, o programa não sofrerá efeitos colaterais.

- Extensibilidade: É a capacidade,facilidade de um software ser adpatado para mudanças em sua especificação, Neste aspecto, dois princípios são essenciais:
  "Simplicidade de design: uma arquitetura simples sempre é mais simples de ser adaptada ou modificada.
  Descentralização: quanto mais autônomos são os módulos de uma arquitetura, maior será a probabilidade de que uma alteração implicará na manutenção de um ou poucos módulos."

- Capacidade de reuso: É a capacidade do programa ser reutitilizado parcialmente ou totalmente para novas aplicações,isso facilita projetos que tem padrões similares,uma forma de polpar tempo.

- Compatibilidade: Facilidade do programa ser combinado com outros.

Outros aspectos:

- Eficiência: "É o bom aproveitamento dos recursos computacionais como processadores,memória, dispositivos de comunicação, etc".

- Portabilidade: "É a facilidade com que um programa pode ser transferido de uma plataforma (hardware, sistema operacional, etc.)"

- Facildade de uso: É a facilidade de aprendizagem de como o programa funciona, sua operação,etc.

1.3 Modularidade - qualidade interna

Olhando os 5 aspectos apresentados anteriormente que são beneficiados com o uso da orientação a objetos, temos dois subgrupos:

Extensibilidade, reuso e compatibilidade demandam arquiteturas que sejam flexíveis, design descentralizado e a construção de módulos coerentes conectados por interfaces bem definidas.

Corretude e robustez são favorecidos por técnicas que suportam o desenvolvimento de sistemas baseados em uma especificação precisa de requisitos e limitações.

"Da necessidade da construção de arquiteturas de programas que sejam flexíveis, surge a questão de tornar os programas mais modulares".

Critérios para modularidade

Decomposição: É a ideia de ter um problema maior e dividí-lo em sub problemas menores conectados por estruturas simples, sendo que juntando cada sub problema, resolvendo-os então resolve-se o problema maior, também conhecido como "dividir para conquistar",no método top-down os programadores tem uma visão mais ampla do problema e então começam a refiná-lo com passos menores, para facilitar a implementação.

Composição: Está diretamente ligada a reutilização, nesse método a ideia é utilizar "pedaços de programas"usados anteriormente para resolver determinado problema em um outro projeto que tem o mesmo problema ou seja similar, nesse contextos temos as ideias

das bibliotecas.

Entendimento: O objetivo desse método é ajudar na produção de módulos que podem ser compreendidos separadamente pelos programadores.

Continuidade: Nesse método, mudanças feitas no sistema devem apenas alterar os módulos onde elas foram implementadas, e nao o sistema como um todo."Um exemplo simples deste tipo de critério é a utilização de constantes representadas por nomes simbólicos definidos em um único local. Se o valor deve ser alterado, apenas a definição deve ser alterada".

Proteção: Visa evitar a propagação de erros, caso ocorra algum problema em tempo de execução por exemplo,a ideia é que o erro fique restrito à apenas aquele módulo em especifico em que se encontra o problema e que não se propague para os outros módulos.

Princípios da modularidade

Linguística modular: "Módulos devem correspoder às unidades sintáticas da linguagem utilizada.", Este princípio segue de diversos critérios mencionados anteriormente como à: composição, decomposição e a proteção.

Poucas interfaces: "Cada módulo deve se comunicar o mínimo possível com outros".

Pequenas interfaces: "Se dois módulos possuem canal de comunicação, estes devem trocar o mínimo de informação possível; isto é, os canais da comunicação intermodular devem ser limitados".

Interfaces explícitas: Impõe limitações no número de módulos que se comunicam, também no número de informações trocadas. Este princípio segue de diversos critérios mencionados anteriormente: Decomposição e entendimento.

Ocultação de informação (information hiding): "Toda informação sobre um módulo deve ser oculta (privada) para outro a não ser que seja especificamente declarada pública,a razão fundamental para este princípio é o critério de continuidade".

Calculadora RPN em C: Calculadora RPN (notação polonesa reversa), utilizando uma pilha para armazenas os dados:

em um módulo à parte. O header file etá listado abaixo:

```
#ifndef stack_h
#define stack_h

#define MAX 50

struct Stack {
  int top;
  int elems[MAX];
};

void push(struct Stack* s, int i);
int  pop(struct Stack* s);
int  empty(struct Stack* s);
struct Stack* createStack(void);

#endif
```

A implementação destas funções está no arquivo stack-c.c:

```
#include <stdlib.h>
#include "stack-c.h"

void push(struct Stack*s, int i) { s->elems[s->top++] = i; }
int  pop(struct Stack*s)         { return s->elems[--(s->top)]; }
int  empty(struct Stack*s)       { return s->top == 0; }

struct Stack* createStack(void)
{
   struct Stack* s = (struct Stack*)malloc(sizeof(struct Stack));
   s->top = 0;
   return s;
}
```

A calculadora propriamente dita utiliza estes arquivos:

```
#include <stdlib.h>
#include <stdio.h>
#include "stack-c.h"

/* dada uma pilha, esta função põe nos
   parâmetros n1 e n2 os valores do topo
   da pilha. Caso a pilha tenha menos de dois
   valores na pilha, um erro é retornado */
int getop(struct Stack* s, int* n1, int* n2)
{
   if (empty(s))
   {
      printf("empty stack!\n");
      return 0;
   }
   *n2 = pop(s);
   if (empty(s))
   {
      push(s, *n2);
      printf("two operands needed!\n");
      return 0;
   }
   *n1 = pop(s);
   return 1;
}
```

```
/* a função main fica em um loop
   lendo do teclado os comandos da calculadora.
   Se for um número, apenas empilha.
   Se for um operador, faz o calculo.
   Se for o caracter 'q', termina a execução.
   Após cada passo a pilha é mostrada. */
int main(void)
{
   struct Stack* s = createStack();
   while (1)
   {
      char str[31];
      int i;
      printf("> ");
      gets(str);
      if (sscanf(str, " %d", &i)==1) push(s, i);
      else
      {
         int n1, n2;
         char c;
         sscanf(str, "%c", &c);
         switch(c)
         {
            case '+':
               if (getop(s, &n1, &n2)) push(s, n1+n2);
               break;
            case '-':
               if (getop(s, &n1, &n2)) push(s, n1-n2);
               break;
            case '/':
               if (getop(s, &n1, &n2)) push(s, n1/n2);
               break;
            case '*':
               if (getop(s, &n1, &n2)) push(s, n1*n2);
               break;
            case 'q':
               return 0;
            default:
               printf("error\n");
         }
      }
      {
         int i;
         for (i=0; i<s->top; i++)
            printf("%d:%6d\n", i, s->elems[i]);
      }
   }
}
```

Códigos retirados do livro Programação Orientada a Objetos com C++,dos autores Renato Borges e André Luiz Clinio

1.4 Tipos abstratos de dados

"Os objetos devem ser classificados de acordo com o seu comportamento esperado. Esse comportamento é expresso em termos das operações que fazem sentido sobre esses dados; estas operações são o único meio de criar, modificar e se ter acesso aos objetos".

Classes em C++: É a base da orientação a objetos,onde ela está apoiada, aqui os tipos abstratos de dados não são definidos pela sua representação interna mas sim pelas operções sob o tipo, no código da calculadora acima a pilha tornaria-se uma classe.

O paradigma da orientação a objetos

- Objeto: "É uma abstração encapsulada que tem um estado interno dado por uma lista de atributos cujos valores são únicos para o objeto. O objeto também conhece uma lista de mensagens que ele pode responder e sabe como responder cada uma".

- Mensagem:"É representada por um identificador que implica em uma ação a ser tomada pelo objeto que a recebe. Mensagens podem ser simples ou podem incluir parâmetros que afetam como o objeto vai responder à mensagem. A resposta também é influenciada pelo estado interno do objeto ".

- Classe: "É um modelo para a criação de um objeto. Inclui em sua descrição um nome para o tipo de objeto, uma lista de atributos (e seus tipos) e uma lista de mensagens com os métodos correspondentes que o objeto desta classe sabe responder".

- Instância: "É um objeto que tem suas propriedades definidas na descrição da classe. As propriedades que são únicas para as instâncias são os valores dos atributos".

- Método: "É uma lista de instruções que define como um objeto responde a uma mensagem em particular. Um método tipicamente consiste de expressões que enviam mais mensagens para objetos. Toda mensagem em uma classe tem um método correspondente ".

"Objetos são instâncias de classes que respondem a mensagens de acordo com os métodos e atributos, descritos na classe".