UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
CAMPUS CAMPO MOURÃO

RESENHA CAPÍTULO 1 - ARTIGO "Renato Borges e André Luiz Clinio. "Programação Orientada a Objetos com C++". Em: Apostila, Rio de Janeiro, 101p."

1 Qualidade de *Software*?

Quando almejamos utilizar a engenharia de *software* a fim de construir um aplicativo de qualidade, devemos seguir um conjunto de fatores para atingir tal objetivo.

Necessitamos distinguir dois tipos de qualidades, sendo elas: **qualidade externa de *software*** (considera aspectos como eficiência, facilidade de uso, etc.) e **qualidade interna de *software*** (pondera aspectos como legibilidade, modularidade, etc.).

1.1 Fatores de Qualidade Externa

Existem vários motivos que ressalta a qualidade externa de um *software*, algumas são beneficiadas com o uso da **orientação por objetos**, sendo elas: corretude, robustez, extensibilidade, reuso e compatibilidade. Com objetivo de esclarecer o significado de tais qualidades, listaremos suas definições e de algumas outras:

- **Corretude:** O *software* é capaz de produzir respostas adequadas e corretas cumprindo rigorosamente suas especificações.
- **Robustez:** Quando o *software* é capaz de funcionar mesmo em condições incomuns.
- **Extensibilidade:** É a facilidade com que o *software* pode ser modificado quando ocorrer alterações em suas especificações.
- **Capacidade de Reuso:** Quando o *software* pode ser reutilizado em novas aplicações, totalmente ou em partes.
- **Compatibilidade:** É a facilidade com que o *software* pode ser combinado com outros.

- **Eficiência:** Quando o *software* aproveita de maneira correta os recursos computacionais.
- **Portabilidade:** É a facilidade com que um *software* pode ser compilado/executado em arquiteturas diferentes.
- **Facilidade de Uso:** O nível de clareza que seu *software* oferece para o usuário.

Alguns aspectos citados acima possuem necessidades especificas para atingir tal qualidade, subdividindo-se em dois grupos:

1. **Extensibilidade, Reuso e Compatibilidade:** Demandam de uma arquitetura flexível, modelos coerentes e design descentralizado.
2. **Corretude e Robustez:** Necessitam de desenvolvimento utilizando especificações precisas de requisitos e limitações.

1.2 Fatores de Qualidade Interna

Analisando as necessidades citadas na seção anterior, percebemos que algumas demandam de flexibilidade. Desta forma introduzimos um conceito muito importante na qualidade interna de um *software*.

Como não existe uma definição precisa sobre modularidade, seguimos cinco critérios e cinco princípios para obter êxito durante o seu processo.

1.2.1 Critérios para Modularidade

Segue listado abaixo, critérios para se obter uma boa modularidade.

- **Decomposição:** Dado um problema qualquer, necessitamos subdividir o mesmo em problemas menores, com objetivo de atingir a solução separadamente. Desta forma reduzimos a complexidade inicial, conectando-os utilizando uma estrutura simples.
- **Composição:** O critério é satisfeito quando podemos combinar elementos de ambientes diferentes de um *software*, com objetivo de produzir novos sistemas. Um grande exemplo são as bibliotecas, que acabam oferecendo diversos códigos que podemos utilizar em nosso sistema.
- **Entendimento:** Quando os módulos de nosso *software* podem ser compreendidos de forma separada, ou seja, com pouca dependência de módulos externos.

- **Continuidade:** Ao realizar mudanças em nosso sistema, não deverá ocorrer reflexos em nossa arquitetura geral, ou seja, as alterações somente deverão ter impactos em um único ou poucos módulos.
- **Proteção:** Este critério é estabelecido quando os erros de tempo de execução não são propagados a todos os outros módulos, no máximo a poucos vizinhos.

1.2.2 Princípios de Modularidade

Após estabelecer os critérios de modularidade, necessitamos listar os princípios desta característica. Segue abaixo estes conceitos:

- **Linguística Modular:** Os módulos devem ser implementados seguindo as regras da linguagem utilizada, de forma bem delimitada. Em *C++* temos arquivos de extensão .h que nos permite separar o código fonte dos módulos de nosso programa.
- **Poucas Interfaces:** Cada modulo deve se comunicar o mínimo possível com outros.
- **Pequenas Interfaces:** A quantidade de informações trocadas pelos módulos, deve ser a mínima possível.
- **Interfaces Explícitas:** Quando existir comunicações entre módulos, as conexões devem ser bem claras.
- **Ocultação de Informação:** Devemos deixar privado informações não relevantes para a interface pública, ou seja, somente será visível a "casca" do programa. Desta forma qualquer informação pública modificada, será necessário propagar alterações em todos os módulos dependentes. Somente em modificações no "miolo" do programa, os módulos dependentes não necessitaram de manutenção (caso o contrato da interface pública for assegurado).

2 O paradigma da Orientação a Objetos

Quando trabalhamos com orientação a objetos e precisamos representar uma abstração, classificamos o objeto de acordo com o comportamento esperado. Em outras palavras, devemos expressar este comportamento via operações que fazem sentido, e utilizando-as conseguimos realizar criações, modificações e leituras em nosso objeto.

Antes de aprofundar o debate sobre orientação a objetos, precisamos de antemão, definir alguns componentes que compõe este paradigma.

2.1 Objeto

Descrevemos objeto como uma entidade, onde seu estado é definido por uma lista de atributos, e seus valores são únicos para cada instância do mesmo. Sua comunicação com outros objetos é feita via mensagens, as quais o mesmo sabe como responder.

2.2 Mensagem

São ações representadas por identificadores, que o objeto receptor deverá executar. Podendo ser simples ou com algum parâmetro, afetando a maneira como o objeto receptor responderá a mensagem. Lembrando que o estado interno do objeto também influencia na resposta.

2.3 Classe

Uma classe é o modelo para a criação de um objeto, onde estão descritos como serão os atributos do mesmo, as mensagens com os métodos que o objeto desta classe sabe responder.

2.4 Instância

São objetos onde suas propriedades são definidas por sua classe, e os valores de suas propriedades são únicos para cada instância.

2.5 Método

É uma lista de instruções que define como um objeto responderá a uma mensagem em particular, basicamente o mesmo consiste de expressões que enviam mais mensagens para um método correspondente de um objeto.

2.6 Explicando definições em C++

Como estamos utilizando a linguagem de programação C++ para aprender o paradigma orientado a objetos, podemos simplificar todas as definições acima utilizando jargões da linguagem: classes são estruturas, objetos são variáveis do tipo de alguma classe (instância de uma classe), métodos são funções de classes e enviar uma mensagem a um objeto é chamar um método de um objeto.