2
Engenharia de Software

2.1
Design de Sistemas Orientados a Objetos

Os Sistemas Orientados a Objetos não são mais novidade hoje em dia já estando há muitos anos no mercado. A orientação a objetos permite uma melhor modelagem dos problemas, permitindo melhores abstrações e possibilitando o tratamento de problemas mais complexos.

Alguns principais conceitos de orientação a objetos são indispensáveis para qualquer modelo de desenvolvimento de software. Esses conceitos são: objetos, classes, métodos e atributos. As suas descrições seguem abaixo:

- **Objetos**

  Objetos são a unidade fundamental de qualquer sistema orientado a objetos. Um objeto é qualquer coisa, real ou abstrata, a respeito da qual armazena-se dados.

- **Classes**

  São padrões a partir dos quais Objetos são criados, especificando comportamentos comuns aos Objetos. Classes contêm a declaração de Métodos e Atributos.

- **Métodos**

  Representam o comportamento dinâmico das Classes, definindo serviços oferecidos pela Classe.

- **Atributos**

  Definem o comportamento estático das instâncias e normalmente têm seu acesso através de Métodos.

Abaixo segue um exemplo dos conceitos descritos acima:

Classe: Automóvel

Objetos: Palio, Gol

Métodos: Acelerar(), Freiar()

Atributos: Cor, Direção, Número de portas

Esses conceitos fazem parte de qualquer estrutura orientada a objetos. Algumas definições de relações entre classes são importantes de serem destacadas, como a relação de herança e dependência.

A herança representa a hierarquia das classes, possibilitando a especialização, evolução e principalmente a reutilização de classes. O conceito de herança permite que métodos e atributos de classes mais genéricas sejam herdados por classes mais especializadas. Se a classe B herda da classe A, diz-se que a classe A é a super-classe da classe B. Normalmente, uma classe só pode ter uma super-classe, mas uma classe pode possuir várias herdeiras.

Um exemplo da relação de herança pode ser visto na Figura 2 abaixo.

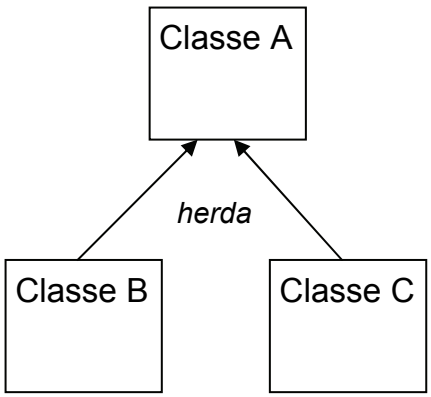

Figura 2 – Exemplo da relação de Herança

No exemplo, a Classe A representa as funcionalidades mais genéricas, e as Classes B e C especializações da Classe A. Numa aplicação, poderia-se ter:

- **Classe A**: Meio de Transporte
- **Classe B**: Automóvel
- **Classe C**: Trem

As relações de dependência simplesmente especificam que uma classe é dependente de outra. Neste trabalho, a dependência estará sendo interpretada da seguinte forma: se uma classe A é dependente de uma classe B, a classe A possui um atributo do tipo classe B, tendo assim, uma referência para a mesma.

Continuando o exemplo acima, a relação de dependência pode ser vista na Figura 3:

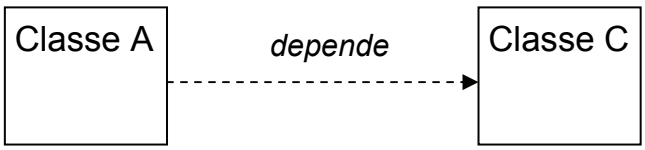

Figura 3 – Exemplo da relação de Dependência

Na Figura 3, a Classe A possui um atributo do tipo da Classe C. Uma aplicação exemplo poderia ser:

- **Classe A**: Automóvel
- **Classe C**: Motor

Assim, se a Classe C tivesse que ser modificada, provavelmente as classes dependentes dela sofreriam algum impacto, no caso, a Classe A.

Todos esses conceitos têm sido utilizados pela engenharia de software para modelar problemas complexos e têm sido contemplados na maioria das ferramentas CASE (Computer-Aided Software Engineering). Com o avanço da engenharia de software, as ferramentas CASE estão cada vez mais complexas e com processos automatizados sendo que, na maioria das vezes, fornecem suporte para o desenvolvimento de documentos em UML (Unified Modeling Language). (OMG, 2003)

A UML é uma linguagem que foi padronizada pela OMG (Object Management Group) para a representação de sistemas orientados a objetos. Utilizar a UML em um modelo orientado a objetos permite integrar mais eficientemente as etapas do processo de desenvolvimento.

A UML possui 12 diagramas principais, nesta pesquisa será utilizado somente um dos diagramas promovidos pela UML, o diagrama de classes. Esse diagrama mostra a estrutura estática do modelo, em particular os elementos que existem no sistema, a sua estrutura interna e suas relações com outros elementos. No diagrama de classes estão representadas as classes, métodos, atributos, heranças e as dependências entre classes, conceitos que foram descritos acima.

O uso de UML no desenvolvimento de software orientado a objetos facilita a modelagem de problemas mais complexos e aumenta a qualidade do

produto final. Essa qualidade pode ser monitorada nas diversas fases do processo de desenvolvimento, mas o controle de qualidade nas fases iniciais é mais importante e eficiente, pois pode ajudar o desenvolvedor a tomar melhores decisões. Existem métricas que podem ajudar a estimar a qualidade em diagramas UML, facilitando o trabalho do desenvolvedor (Genero et al, 2003).

2.2
Métricas de Qualidade de Software

Em cada modelagem de software, um conjunto de atributos de qualidade podem ser identificados. As modelagens que satisfazem a estes atributos de qualidade podem ser classificadas como as "boas" modelagens. O software, quando projetado visando atributos específicos de qualidade, faz a extensão, manutenção e reuso da modelagem de forma mais simples e mais facilmente. Por exemplo, modelagens que são flexíveis, confiáveis e eficazes nos termos de arquitetura do sistema e dos recursos (espaço, tempo) são boas candidatas ao reuso, e economizam tempo e despesas consideráveis quando as extensões e as modificações são necessárias.

Pelo fato de existirem muitos pontos de vista para o conceito de qualidade, a tarefa de determinar e de avaliar a presença de atributos de qualidade não é nem direta nem fácil. Um conjunto de atributos que possa representar a qualidade é subjetivo. Além disso, não existe maneira de se avaliar diretamente os produtos do processo de desenvolvimento de software para detectar a presença ou a ausência desses atributos, que são considerados, pela maioria dos especialistas, representantes de qualidade.

Os atributos de qualidade em que essa pesquisa foi baseada são os atributos utilizadas por Bansiya (2002) no QMOOD. Esses atributos são: Reutilização, Flexibilidade, Inteligibilidade, Funcionalidade, Extensibilidade e Efetividade.

Esses atributos de qualidade, assim como a qualidade em geral, são conceitos abstratos e portanto não são diretamente observáveis. Ainda, como a qualidade, não existe um acordo universal sobre definições para cada um desses atributos de alto nível. Uma breve descrição das definições e problemas

relacionados aos atributos de qualidade citados acima é fornecida nas próximas sub-seções. Um resumo dessas definições pode ser visto na Tabela 1.

Atributos de Qualidade	Definição
Reutilização	Reflete a presença de características na modelagem orientada a objetos que permitam que a mesma seja reaplicada em um novo problema sem esforço significante.
Flexibilidade	Características que permitam a incorporação de mudanças na modelagem a habilidade da modelagem ser adaptada a fim de prover capacidades diferentes, embora similares.
Inteligibilidade	As propriedades da modelagem que a habilitam a ser facilmente aprendida e compreendida. Isso se relaciona diretamente à complexidade da estrutura da modelagem.
Funcionalidade	Diz respeito às responsabilidades atribuídas às classes de uma modelagem, as quais são disponibilizadas pelas classes através das interfaces públicas.
Extensibilidade	Refere-se à presença e uso de propriedades numa modelagem existente que permita a incorporação de novos requisitos na modelagem.
Efetividade	Refere-se à habilidade da modelagem para atingir a funcionalidade desejada e comportamento usando conceitos e técnicas de orientação a objetos.

Tabela 1 – Definição dos Atributos de Qualidade

2.2.1 Reutilização

Reutilização é um dos principais benefícios que o paradigma da orientação a objetos traz ao desenvolvimento de software. Enquanto o encapsulamento promove a modelagem de componentes auto-contidos melhorando a reutilização, a herança promove a adaptação dos componentes para reuso através da especialização.

Reutilização pode ser definida como a capacidade de uma modelagem poder ser reutilizada sem qualquer mudança. Portanto, para estimar a reutilização precisa-se determinar propriedades que permitam que uma modelagem seja reutilizada sem qualquer mudanças. Cada componente da modelagem precisa ser projetado para ser tão reutilizável quanto possível. O reuso de modelagens existentes numa nova modelagem pode reduzir significantemente o tempo de projeto e simplificar o processo de desenvolvimento.

2.2.2
Flexibilidade

Flexibilidade é a capacidade de se adaptar uma modelagem existente dentro de um dado *framework*. Esse atributo mede a capacidade da modelagem ser modificada a fim de prover as funcionalidades. Uma modelagem que não possibilita ser modificada para incorporar mudanças, como adicionar ou apagar um componente, é uma modelagem rígida, e não é desejável. Uma modelagem que não é aberta a pequenas mudanças também restringe drasticamente a reutilização.

2.2.3
Inteligibilidade

Para uma modelagem ser modificada, melhorada ou reutilizada, ela precisa ser entendida. A complexidade e a inteligibilidade de modelagens estão fortemente relacionadas; tipicamente, modelagens que são complexas são menos propícias a serem facilmente entendidas. Outras propriedades que influenciam a inteligibilidade de uma modelagem são seu domínio, estilo de representação, coesão, acoplamento, documentação, etc. A inteligibilidade de uma modelagem está relacionada diretamente com a manutenibilidade da implementação do software resultante da modelagem.

2.2.4
Funcionalidade

Funcionalidade se refere às responsabilidades (operações) designadas para as classes da modelagem. As classes disponibilizam suas operações através de interfaces públicas. A qualidade da interface é influenciada pelo número de métodos, tipos dos métodos, tipos e número de parâmetros requeridos pelos métodos, e os tipos de atributos definidos na classe. Em uma modelagem, classes que refletem um grande grau de funcionalidade podem justificar o esforço que seria necessário para fazê-las reutilizáveis e flexíveis.

2.2.5
Extensibilidade

Extensibilidade é a medida da facilidade em se adicionar novas funcionalidades (operações) a um componente de uma modelagem existente. É uma avaliação da arquitetura do componente e das técnicas usadas numa modelagem que ajude na estimativa do esforço que pode ser necessário para se adicionar novos requisitos e/ou estender a funcionalidade de modelagens existentes. Se modelagens existentes podem ser facilmente modificadas para poder acomodar novas operações ou comportamentos, o reuso pela extensão dessa modelagem pode reduzir esforços de projeto em outros problemas.

2.2.5
Efetividade

Efetividade é a medida da habilidade da modelagem atingir a funcionalidade e comportamento desejados usando os conceitos e técnicas de orientação a objetos. Enquanto no senso tradicional a efetividade representa a medida das características da modelagem como o custo-benefício entre espaço (requisitos de memória) e tempo (execução), nesta pesquisa, a efetividade será medida pela verificação de que princípios como herança, polimorfismo, encapsulamento e composição foram utilizados na modelagem.