**Resumo**

**4 Modelos**

O capítulo começa com uma introdução geral sobre modelos de software, seguido por uma visão geral da UML, a notação gráfica mais utilizada para criar modelos de software. É explicado que o objetivo do estudo da UML é criar esboços de software e não desenhos técnicos detalhados. Nas seções seguintes, são apresentados quatro tipos de diagramas UML com mais detalhes: Diagramas de Classes, Diagramas de Pacotes, Diagramas de Sequência e Diagramas de Atividades.

## **4.1 Modelos de Software**

No capítulo anterior aprendemos que há uma lacuna entre os requisitos e o código fonte de um sistema, e para preencher essa lacuna, a engenharia de software utiliza modelos que são mais detalhados do que os requisitos, mas menos complexos do que o código fonte. Esses modelos são criados para ajudar no entendimento e análise do sistema. Modelos também são amplamente utilizados em outras engenharias, como a engenharia civil, onde uma maquete ou modelo matemático pode ser criado para simular e testar propriedades de uma construção, como resistência a ventos, ondas e terremotos.  
 Infelizmente, os modelos de software ainda são menos efetivos do que os modelos matemáticos usados em outras engenharias, devido à complexidade essencial do software que é descartada ao abstrair detalhes. Frederick Brooks comenta sobre essa questão em seu ensaio "Não Existe Bala de Prata", onde destaca que a complexidade é uma propriedade essencial do software, e modelos que abstraem essa complexidade também abstraem sua essência. Enquanto os modelos simplificados funcionam em outras áreas da engenharia, isso não se aplica ao software, onde a complexidade é uma propriedade essencial e não pode ser ignorada.  
 A frase de George Box "todos os modelos estão errados, mas alguns são úteis" se aplica não apenas a modelos matemáticos, mas também a modelos de software. Embora os modelos de software sejam simplificações da realidade, eles podem ser úteis para estudar propriedades de um sistema. Embora os modelos de software não tenham a mesma efetividade que modelos em outras engenharias e sejam representações gráficas, eles ainda podem ter um papel importante no desenvolvimento de sistemas de software.  
 É importante destacar que a criação de modelos não se restringe apenas à fase de projeto, mas também pode ser utilizada em outras fases do ciclo de vida do software. Por exemplo, modelos podem ser usados na fase de teste para criar casos de testes a partir de diagramas de sequência ou de atividades, ou na fase de manutenção para entender a estrutura e o comportamento do sistema existente. Além disso, modelos podem ser utilizados para documentar o sistema, tornando mais fácil sua compreensão e manutenção em longo prazo. Modelos de software são uma ferramenta importante para a Engenharia de Software Moderna, permitindo que desenvolvedores e stakeholders visualizem e comuniquem aspectos importantes do sistema de software, além de serem úteis em várias etapas do ciclo de vida do software.  
 Neste capítulo, é apresentado um subconjunto dos diagramas propostos pela UML (Unified Modelling Language), que é uma linguagem visual para modelagem e especificação de sistemas de software. Inicialmente, é descrita a história e o contexto que levaram à criação da UML, que foi desenvolvida por um grupo de empresas e pesquisadores de software com o objetivo de unificar diversas linguagens de modelagem existentes na época.  
 Em seguida, são apresentados alguns dos principais diagramas UML com mais detalhes, como o diagrama de casos de uso, que representa as funcionalidades do sistema sob a perspectiva do usuário, e o diagrama de classes, que descreve as classes do sistema, seus atributos e métodos, e suas relações. Também é abordado o diagrama de sequência, que descreve a interação entre objetos em um cenário específico, o diagrama de atividades, que representa o fluxo de atividades em um processo de negócio ou operação de sistema, e o diagrama de componentes, que descreve os componentes de software e suas interações. Além disso, é apresentado um aprofundamento sobre métodos formais, que são uma abordagem matemática para especificação de sistemas de software. Esses métodos utilizam notações matemáticas, como lógica e teoria de conjuntos, para derivar especificações precisas e não-ambíguas para sistemas de software, que podem ser usadas para provar propriedades do sistema mesmo antes de sua implementação. Embora o uso de métodos formais em engenharia de software tenha sido investigado desde os anos 70, seu uso ainda é limitado na prática, com exceção de sistemas de missão crítica.

## **4.2 UML**

A UML é uma linguagem gráfica para modelagem de software, que foi criada na década de 80 em meio ao crescimento do paradigma de orientação a objetos. A UML oferece um conjunto de diagramas para documentar e ajudar no design de sistemas de software, especialmente sistemas orientados a objetos. Naquela época, várias linguagens orientadas a objetos surgiram, como o C++, e algumas notações gráficas foram desenvolvidas para modelagem de software. Como os sistemas eram desenvolvidos seguindo o Modelo Waterfall, que exigia uma longa fase de design, a UML propôs a criação de modelos gráficos nessa fase, que seriam posteriormente convertidos em código fonte pelos programadores. A UML permite que as equipes de desenvolvimento possam visualizar, especificar, construir e documentar a estrutura e o comportamento do sistema de software, facilitando a comunicação e a compreensão entre as partes envolvidas no projeto.  
 UML é uma notação gráfica para modelagem de software que foi proposta em 1995 com o objetivo de unificar as notações gráficas que surgiram no final das décadas de 80 e início da década de 90. A ideia era criar modelos gráficos na fase de design de sistemas de software, que posteriormente seriam repassados para os programadores para serem convertidos em código fonte. UML foi desenvolvido por três Engenheiros de Software: Grady Booch, Jim Rumbaugh e Ivar Jacobson. Desde 1997, UML é um padrão gerenciado pela OMG, uma organização de padronização financiada por indústrias de software. Durante todo o desenvolvimento de UML, consultores influentes e grandes empresas de ferramentas ou consultoria, como a Rational, estiveram envolvidos. Ferramentas CASE (Computer-Aided Software Engineering) foram desenvolvidas para desenhar diagramas UML e ajudar no design de sistemas orientados a objetos.  
 O uso de UML pode ser classificado em três formas: como blueprint, como linguagem de programação ou como esboço. UML como blueprint corresponde à visão inicial dos criadores da linguagem na década de 90, onde após o levantamento de requisitos, um conjunto de modelos seria criado por analistas de sistemas utilizando diagramas UML e ferramentas CASE. Esses modelos seriam repassados para programadores para a codificação do sistema, sendo recomendado para processos de desenvolvimento do tipo Waterfall ou quando se adota o Processo Unificado (UP). No entanto, o uso de UML como blueprint é cada vez mais raro, principalmente com a adoção de métodos ágeis, onde as decisões de design são tomadas e refinadas ao longo do desenvolvimento, em cada uma das iterações (ou sprints). Por isso, neste capítulo, não é abordado o uso de UML como blueprint.  
 O uso de UML como linguagem de programação corresponde ao uso de UML vislumbrado pela OMG após a padronização da linguagem de modelagem. A ideia era que modelos UML fossem usados para gerar código automaticamente, através do Desenvolvimento Dirigido por Modelos (MDD). UML foi expandida com novos recursos e diagramas para tornar MDD viável, mas mesmo assim, o uso de UML para geração de código não se tornou comum na maioria dos sistemas. A complexidade extra da linguagem é apontada como uma das razões para isso.  
 O uso de UML como esboço é uma forma informal e leve de se construir diagramas que são usados para comunicação entre desenvolvedores em duas situações principais: Engenharia Avante (Forward Engineering) e Engenharia Reversa (Reverse Engineering). Em ambas as situações, o objetivo é discutir e analisar alternativas de design antes de começar a codificar ou analisar e discutir uma funcionalidade já implementada. É importante ressaltar que o objetivo não é gerar modelos completos e detalhados, e por isso não é necessário o uso de ferramentas complexas e caras.  
 O uso de UML como esboço evita dois extremos, não assume o emprego rígido, detalhado e sistemático de UML, mas também evita o uso de uma notação informal e ad hoc, cuja semântica pode não ser clara para todos os desenvolvedores. Além disso, UML costuma ser usada em livros, tutoriais e documentos que explicam o uso de frameworks ou técnicas de programação.  
 Segundo pesquisa realizada por Baltes e Diehl em 2013 com 394 desenvolvedores de software de mais de 32 países, 24% criaram o último esboço no mesmo dia em que responderam ao questionário e 39% no intervalo de tempo máximo de uma semana antes da resposta. 58% dos últimos esboços criados foram arquivados, sugerindo que os desenvolvedores consideram que os esboços carregam informação importante, que talvez seja útil no futuro. Por fim, 40% dos esboços foram feitos em papel, 18% em quadros e 39% em computadores, sendo que 52% dos esboços foram feitos para ajudar no projeto (design) da arquitetura do sistema.

## **4.3 Diagrama de Classes**

Diagramas de classes são uma forma gráfica de representar um conjunto de classes e seus atributos, métodos e relacionamentos. As classes são desenhadas em retângulos com três compartimentos, que contêm o nome da classe, seus atributos e métodos. Os atributos e métodos são indicados como públicos ou privados, e o tipo de cada atributo e o retorno de cada método também são especificados.  
 Os diagramas de classes não se limitam a mostrar as classes isoladas, mas também ilustram os relacionamentos entre elas. Existem três tipos de relacionamentos que podem ser representados: associação, herança e dependência. A associação é representada por uma linha que liga duas classes, com um possível valor multiplicador em cada extremidade da linha. O valor multiplicador indica a quantidade de objetos de uma classe que podem estar associados a objetos da outra classe. As associações podem ser unidirecionais ou bidirecionais.  
 A herança é representada por uma linha pontilhada com uma seta que aponta para a classe mãe. A classe mãe é a classe da qual a classe filha herda atributos e métodos. A dependência é representada por uma linha pontilhada sem setas que indica que uma classe depende de outra, sem necessariamente herdar seus atributos ou métodos. Os diagramas de classes são amplamente utilizados na UML e são uma ferramenta útil para visualizar a estrutura e os relacionamentos de classes em um sistema.

### **4.3.1 Associações**

As associações são uma maneira de representar a relação entre duas classes em diagramas de classe UML. Quando uma classe A tem um atributo b do tipo B, dizemos que há uma associação de A a B, que é representada por uma seta de A a B. O nome do atributo responsável pela associação (neste caso, b) é mostrado no final da seta. As associações nos permitem mostrar como as classes estão conectadas em um sistema. No exemplo dado, a classe Pessoa tem uma associação com a classe Fone, uma vez que Pessoa tem um atributo do tipo Fone. No diagrama UML, essa associação é representada por uma seta de Pessoa para Fone, com o nome do atributo (fone) no final da seta. Isso deixa claro que o atributo pertence a Pessoa, mas aponta para um objeto do tipo Fone.  
 Na UML, existem três tipos de associações: unidirecional, bidirecional e reflexiva. Em uma associação unidirecional, a seta aponta em apenas uma direção, da classe com o atributo para a classe com o tipo do atributo. Em uma associação bidirecional, há setas apontando em ambas as direções entre as duas classes. Em uma associação reflexiva, há uma seta apontando de uma classe para si mesma, indicando que um objeto dessa classe pode ter uma relação com outro objeto da mesma classe.  
 Associações podem incluir informações de multiplicidade que indicam quantos objetos podem estar associados ao atributo responsável pela associação. As informações de multiplicidade mais comuns são 1 (exatamente um objeto), 0..1 (zero ou um objeto) e \* (zero ou mais objetos). No exemplo dado, a associação entre Pessoa e Fone tem uma multiplicidade de 0..1, indicando que uma Pessoa pode ter zero ou um único telefone. O atributo fone de Pessoa pode ter o valor null, indicando que a Pessoa em questão não tem um Fone associado, ou pode ser associado a um único objeto do tipo Fone.  
 A UML oferece diferentes notações para representar associações entre classes, como nomear a associação e mostrar setas para indicar a navegabilidade. No entanto, essas notações podem ser confusas ou ambíguas, especialmente quando se trata de associações bidirecionais sem setas. Além disso, a UML também define os conceitos de composição e agregação para relacionamentos entre classes, mas eles tendem a gerar confusão e não são necessários na maioria dos casos. Alguns autores recomendam ignorar a agregação em diagramas de classes.

### **4.3.2 Herança**

Em diagramas de classes, as relações de herança são representadas por setas com a extremidade não preenchida, indicando que a classe conectada é uma subclasse da classe base. As subclasses herdam todos os atributos e métodos da classe base, mas também podem adicionar novos membros. No exemplo mencionado, as subclasses PessoaFisica e PessoaJuridica são subclasses da classe base Pessoa, e adicionam os atributos específicos cpf e cnpj, respectivamente.

### **4.3.3 Dependências**

Em diagramas de classes, a dependência entre duas classes é representada por uma seta tracejada que aponta de uma classe para outra. Essa dependência ocorre quando uma classe usa outra, mas não por meio de associação ou herança. Um exemplo de dependência pode ser um método de uma classe que usa uma variável do tipo de outra classe como parâmetro ou variável local. A dependência é considerada uma modalidade menos forte de relacionamento entre classes do que associação e herança.   
 Diagramas de classes nem todas as dependências de uma classe são representadas, apenas as mais relevantes para a funcionalidade do sistema. As dependências são representadas por setas tracejadas de uma classe para outra, indicando que a primeira classe usa a segunda de alguma forma. É possível indicar o tipo de dependência usando palavras como create ou call, escritas entre sinais de menor e maior. Dependências são uma modalidade menos forte de relacionamento entre classes do que associação e herança.