ANÁLISE E PROJETO DE SISTEMAS

Cleverson Lopes Ledur

SOLUÇÕES EDUCACIONAIS INTEGRADAS

Revisão técnica:

Jeferson Faleiro Leon
Desenvolvedor de Sistemas
Especialista em Formação Pedagógica de Professores
Professor de curso Técnico em informática

L475a Ledur, Cleverson Lopes
Análise e projeto de sistemas [recurso eletrônico] / Cleverson Lopes Ledur ; [revisão técnica: Jeferson Faleiro Leon]. – Porto Alegre : SAGAH, 2017.

ISBN 978-85-9502-179-2

1. Computação. 2. Projeto de sistemas. 3. Análise de projeto. I. Título.

CDU 004.41

Catalogação na publicação: Ana Paula M. Magnus – CRB 10/2052

Elaborar diagrama de classes

Objetivos de aprendizagem

Ao final deste texto, você deve apresentar os seguintes aprendizados:

- Reconhecer os conceitos básicos sobre o diagrama.
- Identificar as relações entre as classes, seus atributos e métodos básicos.
- Elaborar o diagrama de classes adequadamente.

Introdução

Os diagramas da UML são divididos em duas categorias básicas: **diagramas de estrutura** e **diagramas de comportamento**. Cada um dos diagramas da UML está dentro de uma dessas categorias. Pode-se dizer que o objetivo dos diagramas estruturais é mostrar a estrutura estática que o sistema terá. Esse tipo de categoria inclui os diagramas de classe, de componente e de objetos. Já os diagramas comportamentais possuem como objetivo mostrar o comportamento dinâmico entre os objetos. Dentro dos diagramas estruturais, encontra-se o diagrama de classes, que oferece um diagrama muito interessante em termos de estrutura, devido à sua riqueza de elementos, que permite a descrição de sistemas.

Neste texto, você vai entender como e quando utilizar o diagrama de classes, seus elementos e relações, além de alguns pontos importantes que devem ser considerados ao criá-lo.

Diagrama de classes: conceitos básicos

Serão apresentados alguns conceitos essenciais sobre o diagrama de classes. Segundo a UML, um diagrama de classes oferece três perspectivas diferentes, cada uma focada em atender às necessidades de informação de diferentes observadores (AMBLER, 2016). Essas perspectivas são classificadas em:

- **Conceitual:** direcionada para a representação dos conceitos do domínio em estudo ou destinada ao cliente.
- **Especificação:** direcionada para as principais interfaces da arquitetura, nos principais métodos e suas implementações. Também, é direcionada para aqueles que não precisam saber detalhes sobre o desenvolvimento.
- **Implementação:** esta é a representação mais utilizada que demonstra detalhes de implementação, navegabilidade, atributos e é voltada para o time de desenvolvimento.

Link

Você pode ver exemplos destes três tipos de modelos no link a seguir:

https://goo.gl/3eMdA

De forma geral, pode-se dizer que o propósito do diagrama de classes é mostrar os tipos que estão sendo modelados no sistema (AMBLER, 2016). Quando se fala em elementos do diagrama de classes, podem ser citados alguns dos seguintes itens (que logo mais serão detalhados):

- Classe.
- Interface.
- Tipo de dado.
- Componente.

É dito que na engenharia de software, um diagrama de classes na *Unified Modeling Language* (UML) é um tipo de diagrama de estrutura estática, que descreve a estrutura de um sistema, que mostra as classes do sistema, seus atributos, as operações (ou métodos) e as relações entre objetos.

Assim, o diagrama de classes é o bloco de construção principal da modelagem orientada a objetos. É usado tanto para modelagem conceitual geral, quanto para a sistemática da aplicação e para modelagem detalhada, na tradução dos modelos para o código de programação. Os diagramas de classes também podem ser usados para modelagem de dados. As classes, em um diagrama de classes, representam os principais elementos, as interações na aplicação e as classes a serem programadas.

Na concepção de um sistema, várias classes são identificadas e agrupadas em um diagrama de classes, ajudando a determinar as relações estáticas entre elas. Com a modelagem detalhada, as classes do projeto conceitual são muitas vezes divididas em várias subclasses.

Para descrever melhor o comportamento dos sistemas, esses diagramas de classes podem ser complementados por um diagrama de estado ou máquina de estado UML.

Classe

Classes são uma descrição de um grupo de objetos com papéis semelhantes no sistema. A partir das classes, podemos criar os objetos (instâncias). As características estruturais definem o que os objetos de classe sabem. Já as características comportamentais definem o que os objetos de classe podem fazer (KIM; CARRINGTON, 1999).

Os objetos derivam de:

- **Coisas:** objetos tangíveis, do mundo real etc.
- **Papéis/funções:** classes de atores em sistemas, como por exemplo, estudantes, gerentes, enfermeiros etc.
- **Eventos:** admissão, inscrição, matrícula etc.
- **Interações:** reuniões, tutoriais etc.

No diagrama de classes, se representa uma classe utilizando um retângulo. O nome da classe fica no topo do retângulo, geralmente, em negrito. Logo a seguir, são declarados os atributos e os métodos/operações (AMBLER, 2016). A Figura 1 apresenta um exemplo de classe representada no diagrama de classes, em que o nome da classe é sempre obrigatório.

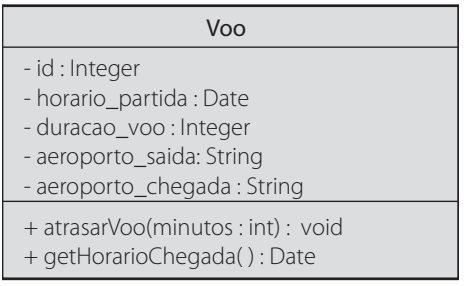

Figura 1. Representação de uma classe no diagrama de classes.

É possível ver que no compartimento superior contém o nome da classe Voo. Ele é impresso em negrito, centralizado e a primeira letra é maiúscula. Já o compartimento do meio contém os atributos da classe. Eles são alinhados à esquerda e a primeira letra é minúscula. Por fim, no compartimento inferior consta as operações que a classe pode executar, também alinhadas à esquerda e a primeira letra é minúscula.

Atributo

Atributos representam o estado de um objeto da classe. São descrições da estrutura ou das características de uma classe. No diagrama de classes, os atributos são listados na parte do meio do retângulo que representa uma classe. Nele, deve-se inserir o nome do atributo, seguido por dois pontos e qual o tipo de dado do atributo (Figura 2).

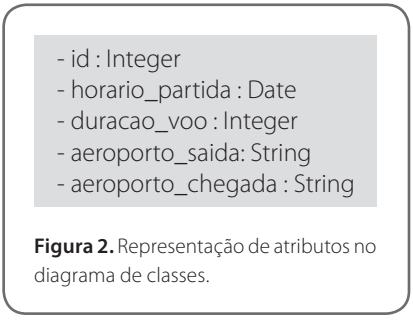

Figura 2. Representação de atributos no diagrama de classes.

Na Figura 2, tem-se um exemplo de atributos que foi extraído da classe Voo. Repare que os nomes (id, horário_partida, duracao_voo, aeroporto_saida, aeroporto_chegada) estão do lado esquerdo dos pontos, e do lado direito o tipo de dado (Integer, Date, Integer, String, String). A definição dos atributos no diagrama de classes é opcional (AMBLER, 2016).

Operações/métodos de classe

Um método na programação orientada a objetos é um procedimento associado a uma mensagem e a um objeto. Um objeto é composto, principalmente, de dados e comportamentos, que formam a interface que um objeto apresenta ao mundo exterior. Os dados são representados como propriedades do objeto e o comportamento como métodos.

Assim como os atributos, os métodos também possuem tipos. No caso dos métodos, o tipo associado é relacionado ao valor que será retornado ao fim da computação.

Em um diagrama de classes, os métodos encontram-se dentro da classe. No interior de uma classe, se tem as definições de operações/métodos na parte inferior do retângulo que representa a classe (BELL, 2016). Cada método deve ficar em uma linha, e assim como nos atributos, o tipo de retorno do método é colocado após os dois pontos, do lado direito. Neste elemento, há a adição dos argumentos que serão passados para o método. Na Figura 3, por exemplo, é possível perceber o método *atrasarVoo* e o argumento *minutos* sendo passado como um valor inteiro. No caso desse método em específico, ele retorna a um valor *void*.

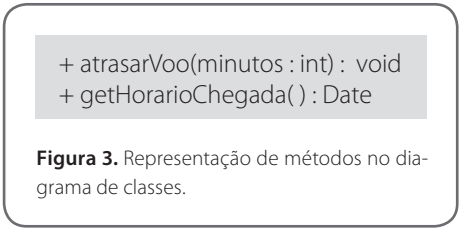

Figura 3. Representação de métodos no diagrama de classes.

Visibilidade

Uma das notações que existe no diagrama de classes é a visibilidade. A especificação UML não solicita que a visibilidade de atributos e operações seja exibida no diagrama de classes, mas requer que ela seja definida para cada atributo e operação. Para exibir a visibilidade no diagrama de classes, se deve colocar a marca de visibilidade na frente do nome do atributo ou da operação (BELL, 2016).

Embora, a UML especifique quatro tipos de visibilidade, uma linguagem de programação real pode incluir visibilidades adicionais ou pode não suportar as visibilidades definidas pela UML. Veja na listagem abaixo, as visibilidades que se encontram na especificação UML:

- + public (público)
- - private (privado)
- ~ package (package)
- # protected (protegido)

Pacotes

Um pacote é um espaço usado para agrupar elementos que são semanticamente relacionados e podem ser alterados de forma unificada. É um mecanismo de propósito geral para organizar elementos em grupos para fornecer uma melhor estrutura para o modelo do sistema.

Os membros pertencentes a um pacote devem ser todos elementos de pacotes. Se um pacote é removido de um modelo, todos os elementos pertencentes ao pacote também serão removidos. O pacote por si só é um elemento, portanto, qualquer pacote pode ser também um membro de outros pacotes ou conter outros pacotes.

Como o pacote é um espaço, os elementos relacionados ou de um mesmo tipo devem ter nomes únicos dentro do pacote envolvente. Diferentes tipos de elementos podem ter o mesmo nome. Além disso, um pacote pode importar membros individuais de outros pacotes ou todos os membros de outros pacotes. Também, o pacote pode ser mesclado com outros pacotes.

Um pacote é processado como uma pasta com abas, um retângulo com uma pequena aba anexada ao lado esquerdo da parte superior do retângulo. Se os membros do pacote não forem mostrados dentro do retângulo da embalagem, o nome da embalagem deve ser colocado no interior.

Quando se está modelando um grande sistema, é preciso modularizá-lo de forma que seja possível desenvolvê-lo de forma adequada e permitir uma fácil manutenção depois. No diagrama de classes, se pode contar com os elementos do pacote para dividir o sistema em subsistemas (BELL, 2016). Os pacotes permitem que os modeladores organizem os classificadores de modelo em *namespaces*, semelhante às pastas em um sistema de arquivamento. Dividir um sistema em vários pacotes, facilita o entendimento do sistema, principalmente se cada pacote representar uma parte específica do sistema. A Figura 4 apresenta o elemento pacote conforme a UML.

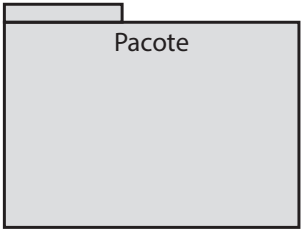

Figura 4. Representação de pacotes no diagrama de classes.

Um exemplo seria de um sistema de aeroporto, onde há um módulo de gerenciamento de voos chamado SistemaVoos. A Figura 5 apresenta o uso do pacote em um diagrama deste tipo.

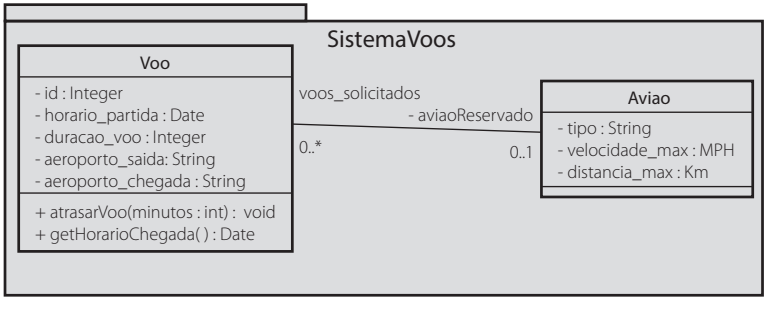

Figura 5. Exemplo do uso de pacotes no diagrama de classes.

O pacote pode ser usado como um modelo para outros pacotes. A especificação UML chama esse tipo de pacote de *template package*. O elemento do pacote pode ser usado como um parâmetro do modelo. Com isso, um parâmetro de modelo do pacote pode se referir a qualquer elemento de propriedade, podendo ser usado um modelo do pacote ou modelos aninhados dentro dele. Veja na Figura 6 um exemplo de *template package*.

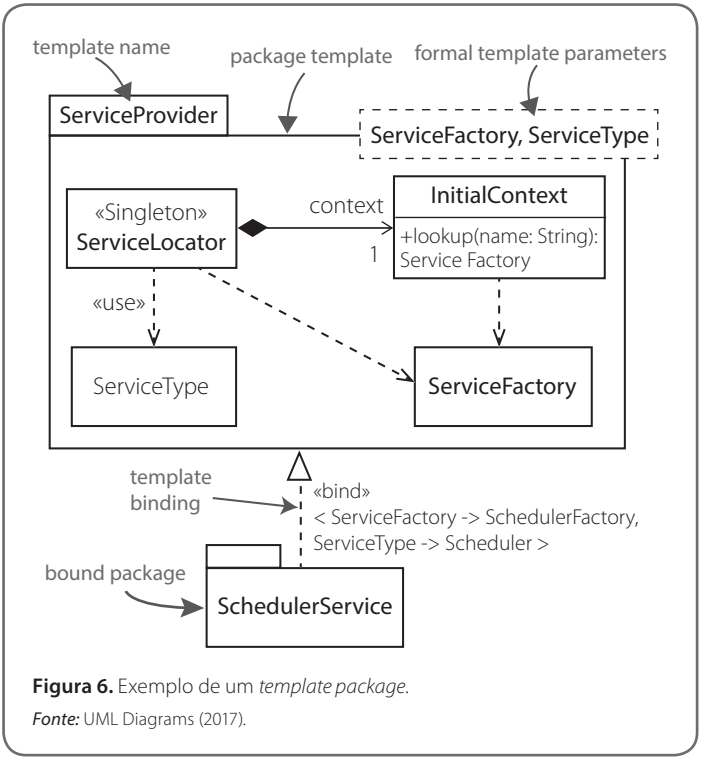

Figura 6. Exemplo de um *template package*.
Fonte: UML Diagrams (2017).

Um pacote pode também estar vinculado a um ou mais pacotes de modelos. Quando várias ligações são aplicadas, o resultado das ligações é produzido tomando-se os resultados intermediários e incorporando-os no resultado combinado com o uso da combinação de pacotes.

Nomeação de pacotes e domínio invertido

A UML não especifica uma forma para a nomenclatura de pacotes, no entanto, é recomendado que os nomes sejam definidos de forma que o pacote seja universalmente único dentro de um projeto. Para isso, normalmente, usa-se um caminho de domínio invertido, como por exemplo, **br.com.sagah** como pacote principal. O pacote principal é seguido do nome do projeto/aplicação. Dentro do pacote da aplicação são criados os pacotes para as outras partes da aplicação, conforme a arquitetura utilizada.

Os nomes dos pacotes internos da aplicação devem tentar expressar o propósito das classes da forma mais simples possível. Por exemplo, um pacote que se chama **br.com.sagah.app.validacao** deverá conter classes que auxiliam a validação. Os nomes dos pacotes não devem ser usados para dividir entres camadas a sua aplicação. Podem ser usados em casos especiais para separar entre nodos como **br.com.sagah.app.server** e **br.com.sagah.app.client**.

Relações entre as classes, seus atributos e métodos básicos

Na UML, há diversas relações que podem ser utilizadas para determinar a forma com que as classes se relacionam. As relações determinam que as instâncias de uma classe estão, de alguma forma, ligadas às instâncias da outra classe. No diagrama de classes tem-se os elementos para representar as relações de associação, herança, dependência, agregação e composição. Cada uma possui um elemento que faz a ligação entre as classes, algumas semelhantes a setas. Você vai ver cada uma dela em detalhes. Mas antes, será falado das multiplicidades que agregam informações para as relações.

Multiplicidades

As multiplicidades são utilizadas no diagrama de classes para informar a quantidade de instâncias de objetos que uma classe pode ter em relação a outra classe. Se tem definido pela UML cinco tipos diferentes de multiplicidades (SCHACH, 2009). Acompanhe a lista a seguir de diferentes multiplicidades:

- 0..1
  - No máximo um. Indica que os objetos da classe associada não precisam obrigatoriamente estar relacionados.
- 1..1
  - Um e somente um. Indica que apenas um objeto da classe se relaciona com os objetos da outra classe.
- 0..*
  - Muitos. Indica que podem haver muitos objetos da classe envolvidos no relacionamento.
- 1..*
  - Um ou muitos. Indica que há pelo menos um objeto envolvido no relacionamento.
- 3..5
  - Valores específicos.

Associação

As associações são definidas como relacionamentos estruturais entre as instâncias das classes e permitem especificar que objetos de uma classe são ligados a objetos de outras classes (BELL, 2016). Essa relação significa que as instâncias das classes são conectadas, seja fisicamente ou conceitualmente. Veja na Figura 7, um exemplo de uma associação entre a classe Voo e a classe Aviao. Fique atento também para o uso das multiplicidades nas extremidades, que foi apresentado anteriormente.

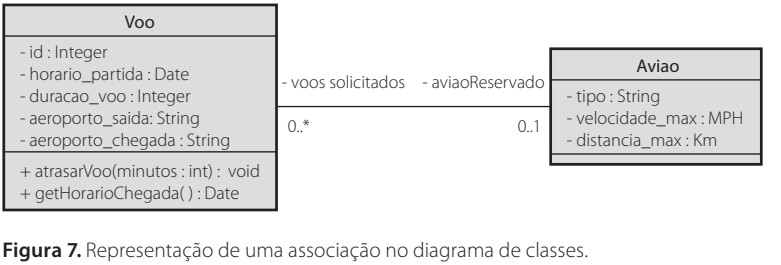

Figura 7. Representação de uma associação no diagrama de classes.

Dependência

As relações nomeadas dependência são relacionamentos de utilização. Nesse tipo de relacionamento, uma mudança na especificação de um elemento altera a especificação do elemento dependente. Assim, a dependência entre classes indica que um objeto de uma classe usa serviços de objetos de outra classe (AMBLER, 2016). Para esta relação utilizamos a seta tracejada (Figura 8).

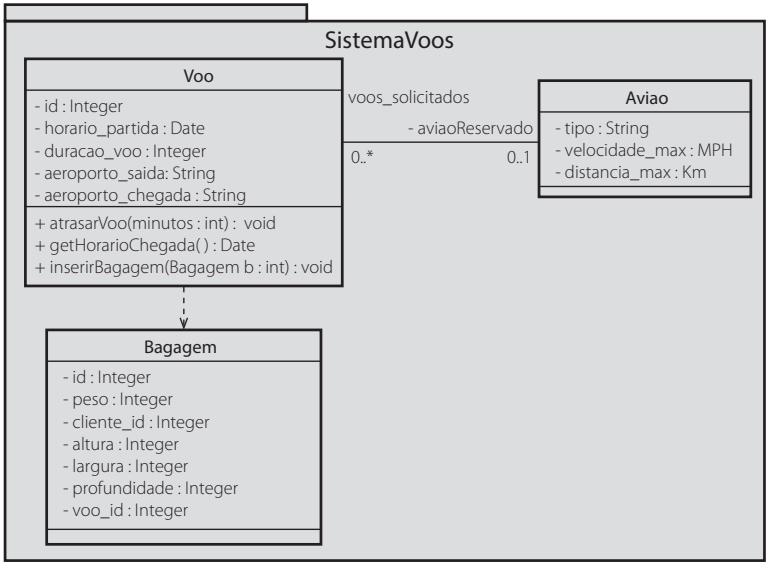

Figura 8. Representação de uma dependência no diagrama de classes.

Observe na Figura 8, um exemplo de dependência entre os elementos Voo e Bagagem. No item Voo há uma operação chamada InserirBagagem. Ela é utilizada quando o avião é carregado com as bagagens. Logo, a classe voo possui uma relação de dependência com a classe Bagagem.

Generalização

A generalização é um tipo de relacionamento entre um elemento geral (mais amplo) e outro mais específico. Ela é utilizada para expressar a herança (simples e composta) de um elemento genérico para outro elemento. Esse elemento usa a seta fechada.

Veja na Figura 9, onde há um exemplo utilizando-se uma classe Piloto (genérica), que é generalizada para as classes PilotoPrincipal e CoPiloto. As classes PilotoPrincipal e CoPiloto herdam todos os atributos da classe Piloto, bem como suas operações/métodos.

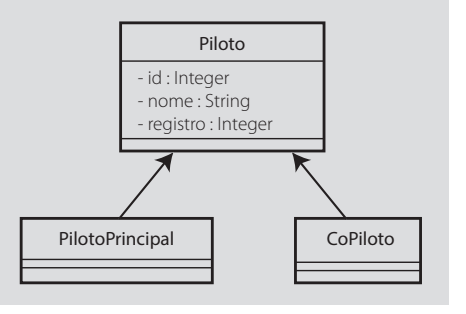

Figura 9. Representação de uma generalização no diagrama de classes.

Agregação

As agregações são um tipo de associação em que o objeto-parte é um atributo do todo. Para tanto, os objetos-parte somente são criados se o todo, ao qual estão agregados, seja criado. Utilizando a mesma classe Piloto, você verá um exemplo de agregação, com uma classe Tripulação. A Figura 10 apresenta um exemplo do uso da agregação.

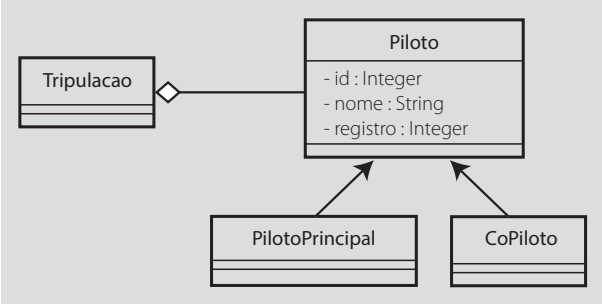

Figura 10. Representação de uma agregação no diagrama de classes.

Composição

A composição é um relacionamento entre um elemento (o todo) e outros elementos (as partes), em que as partes só podem pertencer ao todo, sendo criadas e destruídas com ele. A composição representa um vínculo forte entre duas classes, pois, uma classe FILHA só faz sentido, se uma classe PAI existir. Se a classe PAI for apagada, a classe FILHA automaticamente deixará de existir. A Figura 11 mostra um exemplo do uso da composição:

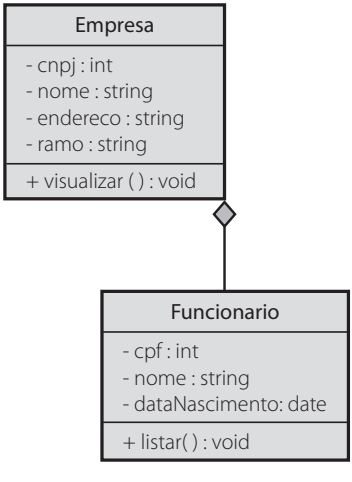

Figura 11. Representação de uma composição no diagrama de classes.

Criação de um diagrama de classes

Nesta seção será falado sobre alguns pontos importantes ao se criar um diagrama de classe, e também de outros recursos que podem ser utilizados para criar um diagrama de classes adequadamente, com solidez e de fácil compreensão.

Tópicos importantes

O *design* de um diagrama de classes pode ser conduzido pelo critério de completude, dados ou responsabilidades.

- **Projeto conduzido por dados:** identifica todos os dados e verifica se estão cobertos por uma coleção de objetos das classes do sistema.
- **Projeto conduzido por responsabilidade:** identifica todas as responsabilidades do sistema e verifica se estão cobertas por uma coleção de objetos das classes do sistema.

Passos para a criação de um diagrama de classes:

- **Identificar frases nominais:** observe os casos de uso e identifique uma frase nominal. Faça isso de forma sistemática e não elimine nenhuma possibilidade.
- **Elimine candidatos inapropriados:** aqueles que são redundantes, vagos, fora do escopo do sistema, um atributo do sistema etc.
- **Identifique os verbos:** de forma geral, os verbos indicam os métodos ou operações que são realizadas em uma classe.
- **Valide o modelo:** verifique se você não selecionou alguma classe que pode prejudicar o seu modelo.

Erros comuns de modelagem de domínio

Durante a criação de um diagrama de classes ou modelagem de domínio, existem alguns erros comuns que são cometidos pelos profissionais. Esses erros podem ocasionar modelos mal construídos, retrabalho e dificuldades

na implementação. Por isso, é muito importante sempre validar o modelo de forma que não sejam cometidos estes erros. Veja alguns erros comuns que são cometidos:

- **Análise excessiva da frase nominal:** a etapa de análise nominal é realizada de forma intensa, o que faz com que sejam coletados itens que não são necessários.
- **Nomes de classe e associação incompreensíveis:** é importante que os nomes das classes e associações sejam claros e sem ambiguidades.
- **Atribuição de multiplicidades às associações prematuramente:** muitos profissionais, ao identificar as classes, já criam as associações e inserem as multiplicidades sem antes analisar as possíveis alterações que podem ser realizadas na estrutura. O ideal é que as multiplicidades sejam inseridas ao final da estruturação do diagrama.
- **Abordagem das questões de implementação antes do tempo:** a criação do diagrama de classes é realizada para representar um modelo. O objetivo é que se for necessário realizar alterações, o modelo é facilmente modificado. Assim, o ideal é que as questões relacionadas às implementações sejam verificadas na fase do ciclo de vida de projeto, e não antes.
- **Comprometimento com as construções de implementação:** como o problema anterior, não é desejável que existam dependências com a implementação durante a criação de um modelo.

Ferramentas para criar diagramas UML

Existem diversas ferramentas que podem ser utilizadas para a criação de diagramas UML. Abaixo segue uma lista com algumas delas:

- **Umbrello** - The UML Modeller https://umbrello.kde.org/
- **Astah** http://astah.net/editions/community
- **Rational Rose** http://www-03.ibm.com/software/products/pt/rosemod
- **ArgoUML** http://argouml.tigris.org/
- **StarUML** http://staruml.io/
- **Dia** http://dia-installer.de/

Referências

AMBLER, S. W. UML 2 *Class Diagram Guidelines*. Agile Modeling, Toronto, 2016.

BELL, D. *Fundamentos básicos de UML*: o diagrama de classes. Uma introdução aos diagramas de estrutura em UML 2. IBM developerWorks, Nova York, 2016.

KIM, S. K.; CARRINGTON, D. Formalizing the UML class diagram using Object-Z. In: FRANCE, R.; RUMPE, B. (Ed.). *UML'99*: the unified modeling language. Berlin: Spring--Verlag, 1999. p. 753-753.

SCHACH, S. R. *Engenharia de Software*: os paradigmas clássico e orientado a objetos. 7. ed. Porto Alegre: McGraw-Hill, 2009.

UML DIAGRAMS. *UML Package Diagrams*, 2017. Disponível em: <http://www.uml--diagrams.org/package-diagrams.html>. Acesso em 7 set. 2017.

Leituras recomendadas

ANDRIYEVSKA, O. et al. Evaluating UML class diagram layout based on architectural importance. In: IEEE INTERNATIONAL WORKSHOP ON VISUALIZING SOFTWARE FOR UNDERSTANDING AND ANALYSIS. 3., Budpest, Sep. 2005. *Proceedings*. . . . Washington: IEEE, 2005.

GUEDES, G. T. A. *UML*: uma abordagem prática. São Paulo: Novatec, 2008.

GUEDES, G. T. A. *UML 2*: guia prático.2. ed. São Paulo: Novatec , 2014.

PRESSMAN, R. S.; MAXIM, B. R. *Engenharia de software*: uma abordagem profissional. 8. ed. Porto Alegre: AMGH, 2016.

Encerra aqui o trecho do livro disponibilizado para esta Unidade de Aprendizagem. Na Biblioteca Virtual da Instituição, você encontra a obra na íntegra.

Conteúdo:

SAGAH
SOLUÇÕES EDUCACIONAIS INTEGRADAS