ECCI XX ENCONTRO CIENTÍFICO CULTURAL INTERINSTITUCIONAL
Uniprime
CENTRO UNIVERSITÁRIO FAG

IMPORTÂNCIA DO DIAGRAMA DE PACOTES PARA O DESENVOLVIMENTO DE SOFTWARE

MONTEIRO, Miguel Angelo[1]
DOS SANTOS, Pedro Augusto da Costa[2]
SILVA, Renata Hansen[3]
ZANINI, Elaine de Oliveira[4]

RESUMO

Este artigo descreve o embasamento teórico a respeito dos diagramas de pacotes, explicitando sua importância para o desenvolvimento de software. Para a elaboração do artigo foi realizada uma pesquisa bibliográfica qualitativa, valendo-se dos conceitos abordados por autores conceituados no assunto. A UML é fundamental no processo de desenvolvimento de software porque é uma forma padronizada e simples de explicitar os requisitos e funcionalidades de um produto de software. Como resultados, foi possível constatar a importância dos diagramas de pacotes para modelagem de sistemas complexos, pelo fato de agrupar e simplificar os elementos dos modelos de software de acordo com um critério conveniente para cada situação. Assim, conclui-se que os diagramas de pacotes são parte fundamental na organização dos modelos e na construção de softwares, possibilitando melhor compreensão e leitura das estruturas descritas por meio de UML, que facilita o desenvolvimento de um software com maior confiabilidade e usabilidade.

PALAVRAS-CHAVE: Engenharia de Software; Modelagem UML; Diagrama de Pacotes

1 INTRODUÇÃO

No processo de produção de software a etapa de modelagem é fundamental para embasar o desenvolvimento e traduzir os requisitos da aplicação trazendo compreensão e comunicação para a equipe. A importância da UML (*Unified Modeling Language* ou Linguagem Unificada de Modelagem em tradução livre) se dá devido à ampla adoção e padronização entre os desenvolvedores, especialmente voltados ao paradigma da orientação à objetos. Tal é sua dominância, que se tornou também técnica popular e difundida inclusive no desenvolvimento estruturado, ou seja, não orientado a objetos (FOWLER, 2007).

O projeto de sistemas em uma escala razoavelmente grande é difícil. Desde uma aplicação *desktop* simples até um "ERP" grande pode ser feito com centenas e potencialmente milhares de componentes de software e hardware. Para gerenciar estes níveis de complexidade é necessário modelar os sistemas, fazendo abstrações dos detalhes irrelevantes ou confusos e simplificando o sistema, deixando claro seu projeto e viabilidade de forma a compreendê-lo e

[1] Acadêmico de engenharia de software do Centro Universitário FAG. E-mail: mamonteiro5@minha.fag.edu.br.
[2] Acadêmico de engenharia de software do Centro Universitário FAG. E-mail: pacsantos@minha.fag.edu.br
[3] Acadêmica de engenharia de software do Centro Universitário FAG. E-mail: rhsilva3@minha.fag.edu.br
[4] Doutora em desenvolvimento rural sustentável. Docente do Centro Universitário FAG. E-mail: ezanini@fag.edu.br.

validá-lo muito mais rápido do que “vasculhar” pelo próprio sistema. A UML é uma linguagem formal, que é abstrata e precisa, como uma linguagem de programação, podendo ser lida, interpretada, executada e transformada entre sistemas (HAMILTON e MILES, 2006).

Com respeito aos diagramas da UML, existem aqueles com a função de descrever comportamento e os diagramas estruturais. Destes, será abordado especificamente neste artigo sobre o diagrama de pacotes. Segundo a ISO/IEC 19505-1 (2012) os pacotes são um *container* para variáveis, classes e outros pacotes, representando uma forma de agrupar esses elementos. Os pacotes são representados por dois retângulos, desenhados de maneira a formar uma pasta com guia. É possível mostrar todo o conteúdo ou apenas o nome do pacote. O diagrama de pacotes, de forma simplificada, descreve as relações e os comportamentos estruturais entre os diversos pacotes modelados de uma aplicação, representando estas operações por meio de setas e palavras chave, de forma a representar a relação de dependência, união, mescla e as demais que ocorrem entre pacotes.

Diante do exposto, este artigo tem o objetivo de descrever o embasamento teórico a respeito dos diagramas de pacotes, explicitando a sua importância para o desenvolvimento de software.

A estrutura do presente artigo está consolidada da seguinte forma: no primeiro tópico se apresenta a introdução; no segundo tópico está exposto o referencial teórico, que contextualiza o diagrama de pacotes e uma de suas aplicações; o terceiro tópico discorre sobre a metodologia adotada para elaboração do artigo; no quarto tópico está descrita a análise e discussão dos conteúdos abordados no levantamento teórico, e por fim; no quinto tópico foram elaboradas as considerações finais, que apontam as conclusões retiradas do estudo com relação à importância e necessidade dos diagramas de pacotes.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 O PROCESSO DE ENGENHARIA DE SOFTWARE

O software é o derivado de um conjunto de informações executáveis que possuem um objetivo e utilizam da lógica para concluir esse objetivo, podendo retornar um resultado ou não. O papel do software tornou-se tão abrangente e necessário ao longo do tempo por ser o distribuidor do produto mais importante de nossa era, a informação (PRESSMAN, 2021).

As possíveis áreas de aplicação podem variar de sistemas para controle de vendas, que auxiliam estabelecimentos a organizar seu fluxo de saídas e entradas, até sistemas de “cálculos

em massa", que fazem previsões de terremotos, eventos meteorológicos e astronômicos, assim como, sistemas de inteligência artificial, que fazem uso de algoritmos não numéricos para a análise de problemas complexos (PRESSMAN, 2021).

Ainda segundo o mesmo autor, o software, mesmo sendo uma das mais importantes tecnologias no cenário mundial, ainda tem seu desenvolvimento dificultoso em relação ao cumprimento dos prazos determinados, orçamentos estabelecidos e à boa qualidade do produto final.

A engenharia de software é o estudo sistemático da produção e desenvolvimento de um software, desde o objetivo pelo qual ele será desenvolvido, funcionalidades das quais deverá conter, ou metodologias que serão utilizadas, até sua definitiva produção e comercialização. Porém deve-se ter em mente que os processos não são igualmente realizados para todos os projetos. Suas definições mudam drasticamente de acordo com o tipo de software, os envolvidos no processo, do desenvolvimento ou aqueles que serão beneficiados com o produto. Sendo assim, não há uma regra a ser seguida, pois todas as práticas irão mudar conforme suas necessidades (SOMMERVILLE, 2011).

Para definir com assertividade quais serão as metodologias, ferramentas e *frameworks* utilizados, um dos fatores para analisar é o tipo de aplicação a ser desenvolvida, que pode variar de aplicações *Stand-alone* ou programas executados de uma máquina remota, assim como sistemas embutidos que podem fazer leituras externas. A partir disso é possível ter um norte de qual caminho seguir, entretanto, outro fator a ser considerado é a plataforma na qual o sistema deverá rodar, pois esse requisito pode mudar totalmente o rumo do planejamento do projeto. Outros fatores são a confiabilidade do sistema com execução eficiente e o processo de gerenciamento de produção do software, que irá definir a dificuldade de inclusão de novos desenvolvedores ao projeto. Todos esses fatores mencionados podem ser usados como alicerce para o desenvolvimento profissional de um bom software (PRESSMAN, 2021).

A modelagem de sistemas tem o intuito de facilitar o desenvolvimento e gerar um produto com maior qualidade. No início do processo de produção de um sistema existe a diagramação dos requisitos, onde há diversas ferramentas da UML (*Unified Modeling Language)* que ajudam neste sentido, como os diagramas de casos de uso, de atividade e de estados. No próximo passo, que é o projeto do software, são úteis ferramentas mais técnicas, como o diagrama de classes, de sequência, de pacotes e de distribuição. Em um estilo de desenvolvimento iterativo os desenhos de projeto são elaborados previamente à codificação, sendo também realimentados durante o desenvolvimento com as alterações para servirem de documento final do software (FOWLER, 2007).

2.2 UNIFIED MODELING LANGUAGE (UML)

2.2.1 Histórico e evolução da UML

Apesar da orientação a objetos estar em crescente uso na década de 80, havia diferentes formas de representar os softwares. Portanto, no início de cada projeto se tornava necessário escolher uma notação e treinar os envolvidos para utilizá-la (DEBONI, 2003).

Ainda de acordo com Deboni (2003), as principais metodologias de representação de sistemas de software existentes na época eram a OOSE de Jacobson (1992), o método de Booch (1994) e o OMT de Rumbaugh (1994). No ano de 1995 Booch e Rumbaugh criaram o conceito do *Unified Method*, que com o ingresso de Jacobson, se tornou *Unified Modeling Language*. Em 1996 os três autores lançaram a versão UML 0.9, solicitando que a comunidade opinasse e, com o material da discussão pública, criaram em seguida as revisões UML 1.0 e 1.1, enviadas e aprovadas em 1997 pelo *Object Management Group*.

Segundo Fowler (2007), das revisões subsequentes, a 1.3 foi a mais significativa, sendo que a 1.2 teve a finalidade de melhorar as aparências. A revisão 1.4 adicionou detalhamento aos conceitos e a 1.5 semântica de ação. A partir daí, da UML 1.0 para a versão 2.0 houve a maior mudança, com a introdução de novos tipos de diagramas, sendo que os diagramas de pacotes passaram a ser oficiais apenas nesta versão, além de trazer os diagramas de visão geral da interação, diagramas de temporização e diagramas de estrutura composta.

A partir daí foram emitidas versões com correções de problemas reportados pela comunidade e também propostas de melhoria dos conteúdos, em 2007 foi lançada a versão 2.1.2, em 2009 a versão 2.2, em 2010 a versão 2.3, em 2011 a versão 2.4.1 e em 2017 a versão atual que é a 2.5.1. A cada lançamento são alterados diversos trechos da documentação, pois ela é editada por diversos colaboradores e é muito extensa, sendo feitas melhorias para uso dos modelos propostos e para deixar mais claros os conceitos abordados (OMG, 2017).

Para Fowler (2007), a UML é um conjunto de notações gráficas que auxiliam a descrição e o projeto de softwares, sobretudo aqueles que utilizam o paradigma da orientação a objetos. O principal motivo para a existência dos diagramas gráficos de modelagem é que as linguagens de programação não viabilizam os debates referentes ao projeto, por não terem um nível de abstração suficiente.

A UML possibilita uma modelagem mais próxima do resultado final, em decorrência dos sistemas serem descritos de forma mais completa. Ela é derivada da união das principais metodologias existentes no início dos anos 90 (DEBONI, 2003).

Seu surgimento em 1997 resolveu o problema da existência de diversas linguagens gráficas de modelagem orientada a objetos, unificando-as. A UML deve, de modo preciso, especificar os projetos de software, possibilitando que outras pessoas sejam capazes de realizar os referidos projetos. Um diagrama bem feito, além de facilitar o fluxo de ideias, também ajuda a compreender um projeto de software ou um processo de negócio. Os diagramas auxiliam na compreensão e na comunicação (FOWLER, 2007).

2.2.2 Formas de uso da UML

De acordo com Fowler (2007), utiliza-se a UML para três diferentes finalidades: esboço, projeto e linguagem de programação. O modo mais comum de se utilizar a UML é para esboçar alguns aspectos do sistema. É possível utilizar o esboço na engenharia reversa, enquanto no desenvolvimento faz-se o diagrama antes de programar o código, na engenharia reversa desenha-se o diagrama UML usando como base o código já escrito, para facilitar a compreensão. O esboço não aborda todo o código, e sim seleciona os pontos fundamentais que se deseja transmitir à equipe. Os esboços mudam rapidamente e tem como característica a informalidade.

Em contrapartida, o diagrama UML cuja finalidade é projeto deve ser completo e detalhado para que o programador escreva o código de maneira simples. Esse modo de utilização é inspirado em outras engenharias, nas quais os engenheiros desenvolvem os desenhos e outros profissionais executam a obra (FOWLER, 2007).

Abordado por Valente (2020), a última forma possível de se utilizar o modelo UML corresponde ao uso como linguagem de programação, também conhecida como: Desenvolvimento Dirigido por Modelos. Essa forma não se popularizou e consistia na codificação ser compilada diretamente do diagrama UML, não sendo necessário programar da maneira tradicional.

Os diagramas UML se dividem em estruturais e comportamentais, os quais são apresentados na sequência.

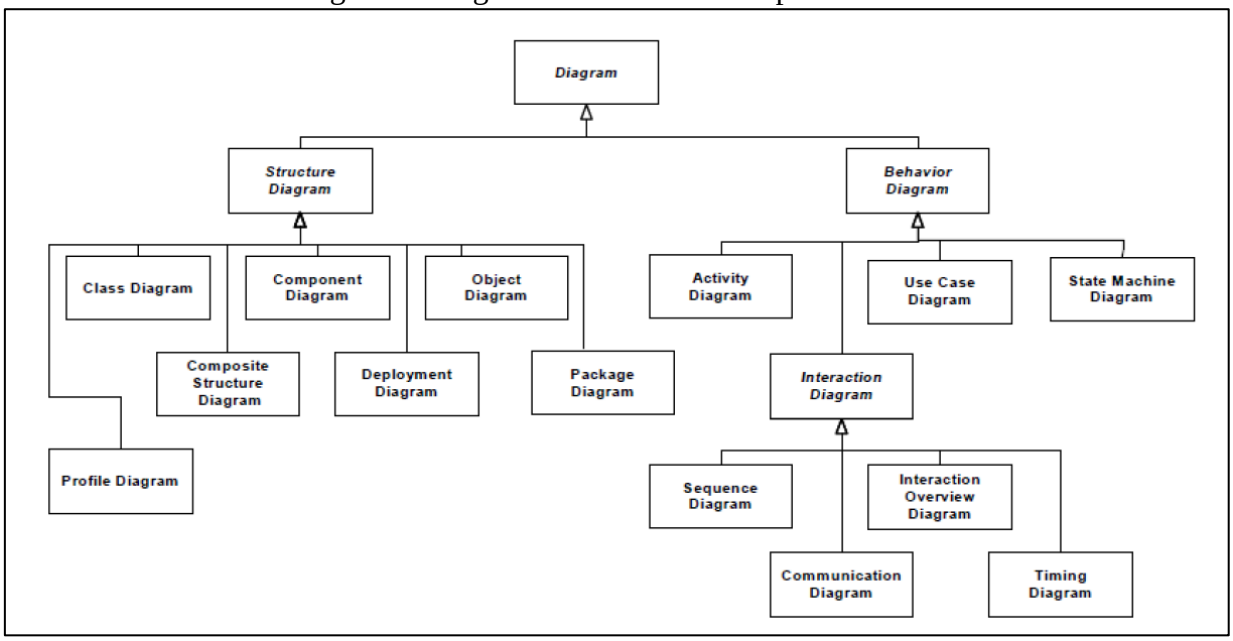
2.3 DIAGRAMAS ESTRUTURAIS

Diagramas estruturais apresentam a estrutura estática de objetos em um sistema. Ou seja, os elementos são diagramados em uma especificação independente de tempo. Nestes diagramas os conceitos significantes de uma aplicação são representados, sejam eles abstratos,

reais ou conceitos de implementação. Por exemplo, um diagrama de estrutura para um sistema de reservas pode incluir classificadores que representam algoritmos para atribuição de assentos, passagens e uma consulta de crédito para autorização do serviço. Estas estruturas não demonstram detalhes do comportamento dinâmico, que é responsabilidade dos diagramas comportamentais, porém podem representar os relacionamentos entre os comportamentos de cada classe (OMG, 2017).

Na Figura 1 estão ilustrados os diagramas estruturais e de comportamento.

Figura 1 – Diagramas de estrutura e comportamento



Fonte: OMG (2017)

Quanto aos estruturais, são eles:

- Diagrama de classes: descreve os tipos de objetos de um sistema e os relacionamentos estáticos entre as classes destes;
- Diagrama de perfil: diagrama de estrutura que apresenta relacionamentos entre perfis de um metamodelo, seja referência, extensão ou aplicação do perfil;
- Diagrama de componentes: estabelece relações entre os componentes de um sistema;
- Diagrama de objetos: é uma representação dos objetos em um sistema em um determinado tempo, demonstrando suas instâncias;
- Diagrama de estrutura composta: é um diagrama que permite observar a estrutura interna de uma classe, útil para objetos complexos;
- Diagrama de pacotes: representa as relações e operações entre os pacotes de uma aplicação;

- Diagrama de implantação: apresentam o layout físico de um sistema, mostrando quais partes do software são executadas em quais partes do hardware.

Quanto aos diagramas de comportamento, são eles:

- Diagrama de atividades: exibe o fluxo das atividades no software, explicitando ações do usuário sobre o software;
- Diagrama de casos de uso: descrevem como os atores, que são os vários tipos de usuário, utilizam o sistema;
- Diagrama de estado de máquina: descrevem o comportamento de um sistema;
- Diagrama de visão geral de interações: é uma mistura entre diagramas de atividades e de sequência onde as atividades são substituídas por pequenos diagramas de sequência;
- Diagrama de sequência: captura o comportamento de um único cenário do sistema, mostrando as interações entre os objetos os métodos chamados;
- Diagrama de comunicação: representa as interações entre os elementos do sistema de forma a enfatizar os vínculos de dados entre os vários participantes;
- Diagrama temporal: representação que tem função de representar as restrições de tempo nas mudanças de estado em diferentes objetos.

2.4 DIAGRAMA DE PACOTES

O diagrama de pacotes é a principal ferramenta da UML para organizar e estruturar modelos de projetos de média a grande escala. Existem especializações destes diagramas para modelos e perfis que organizam extensões da UML. Pode-se dizer que um pacote é um campo da estrutura para seus membros, que compreende os elementos associados, os quais podem ser contidos ou importados. Além disso, a definição de pacote pode ser estendida ao conteúdo de outros pacotes por meio da união dos elementos (OMG, 2017).

Segundo a ISO/IEC 19505-1 (2012) pacotes são um container para variáveis e outros pacotes, representando uma forma de agrupar esses elementos. Estes elementos podem ser úteis para entender e gerenciar um modelo. Uma de suas características é que um pacote não pode conter ele mesmo. Para identificar o pacote é necessário especificar uma “URI” (*Uniform Resource Identifier* ou Identificador único de recurso em tradução livre). Esta identificação

serve a vários propósitos e deve ser tipificada como *string*. A URI deve ser única e imutável uma vez que que é associada a um pacote.

A URI pode ser classificada como um localizador, um nome ou ambos. O termo *Uniform Resource Locator* (URL) é um caso específico da URI que identifica recursos pela representação de seus mecanismos de acesso primário, ou seja, sua localização na rede, ao invés de identificar o recurso pelo nome ou qualquer outro de seus atributos. A sintaxe para estes endereços foi projetada para ser descrita de forma padronizada globalmente como uma de suas principais características. Podemos descrever este identificador como uma sequência de caracteres de tipos limitados que podem ser digitados em um computador de forma a respeitar restrições de teclado entre idiomas e países distintos (BERNERS-LEE, FIELDING, *et al.*, 1998).

De acordo com Fowler (2007) um pacote consiste em uma unidade que permite agrupar outros elementos na UML, elevando o nível de representação. A forma mais comum de utilizar estes pacotes é agrupar diversas classes, porém pode ser utilizado para todos os outros elementos da UML. Além disso, os pacotes estabelecem uma relação de hierarquia estrutural, onde pacotes de nível superior podem ser divididos em subpacotes e assim sucessivamente. Os pacotes permitem distinguir classes de nomes idênticos, porém que são distintas em suas funcionalidades. Além disso, cada classe ou item do pacote deve ter um nome único.

A representação de nomes dos pacotes em UML pode ser feita utilizando dois pontos duplos, como exemplo, System::Date, onde “System” é o pacote e “Date” é a classe indicada contida no pacote. Esta sintaxe é chamada de nome totalmente qualificado. O autor ainda descreve que o diagrama de pacotes é útil para entender a lógica do sistema, visualizar as dependências mantendo o devido controle, funcionando como um mapa lógico do sistema (FOWLER, 2007).

Seidl e Scholz et. al (2015) afirmam que ao diagramar sistemas grandes existe uma grande chance de que o modelo se torne muito complicado. A imensidão de elementos, sejam eles classes, ações, estados e outros, rapidamente sobrepujam as capacidades de uma pessoa ler e interpretar o diagrama em questão. Se um sistema em geral consiste de vários subsistemas os quais têm elementos apenas minimamente relacionados, então é desejável ter um mecanismo para agrupar componentes apropriadamente. Por exemplo, em muitos casos, é confuso se os elementos de uma interface de usuário estão juntos com os elementos do banco de dados.

Para contornar este problema, existem critérios para agrupar classes descritos pela literatura. De acordo com Seidl e Scholz et. al (2015) é possível citar:

- Coesão funcional: elementos com propósito similar são agrupados;

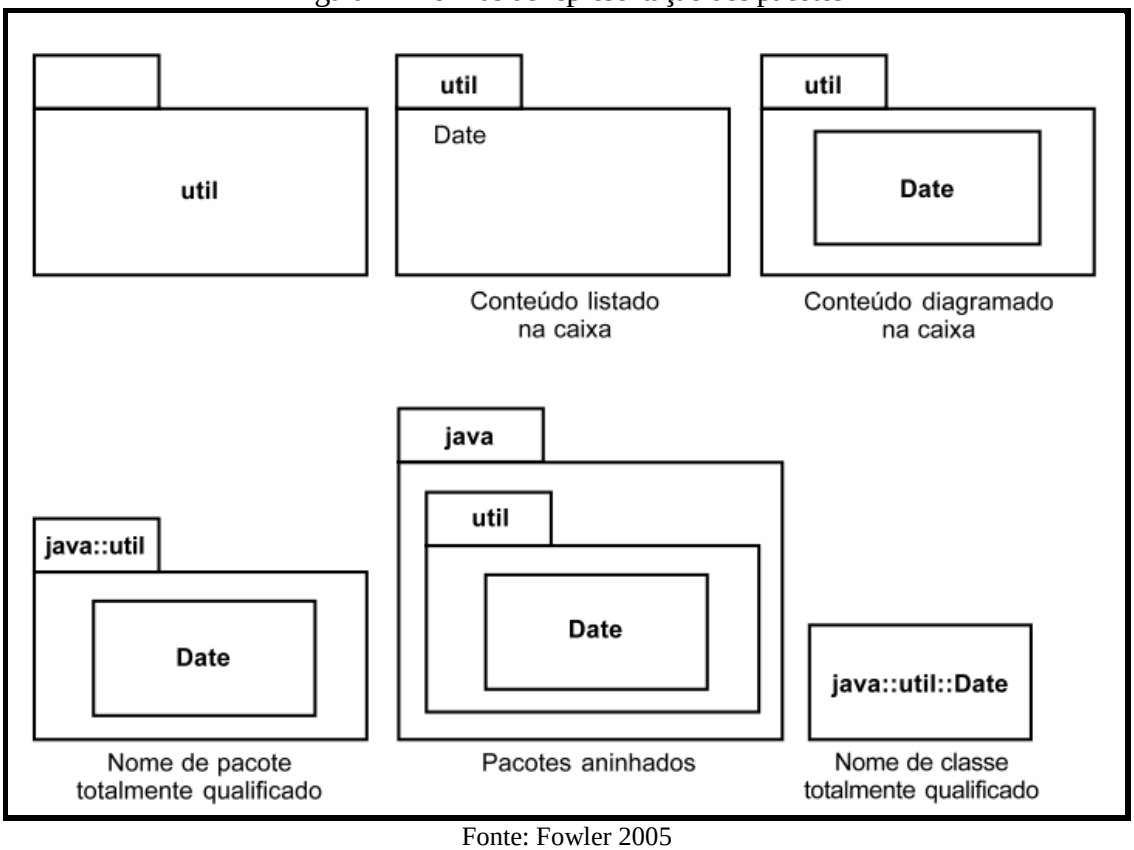
- Coesão informacional: elementos com relacionamento forte são agrupados em detrimento dos elementos com relacionamento menos importante;
- Estrutura de distribuição: ao desenvolver um sistema distribuído, pode-se dividir os elementos de acordo com sua distribuição física, por exemplo, lado do cliente e lado do servidor;
- Estruturando o desenvolvimento: divide-se os pacotes de acordo com as tarefas de desenvolvimento. Critério importante para trabalhos envolvendo mais de um time.

Hamilton e Miles (2006) citam que códigos relacionados à interface gráfica podem pertencer a um pacote específico chamado *gui package* (Graphic User Interface), enquanto algoritmos relacionados à busca estão em outro de título *search* e as utilidades comuns de uma aplicação estão em outro entitulado *util*. Isto facilita encontrar classes em uma API complexa ou qualquer outra estrutura de software. Este tipo de agrupamento exemplifica a coesão funcional das classes desenvolvidas, mas além disso, também ajuda na estruturação do desenvolvimento, dado que um programador de interfaces trabalhando no pacote correspondente sofre e causa pouca interferência no desenvolvedor que trabalha no pacote de pesquisa, e ambos utilizam classes do pacote de utilidades que deve ser suficientemente consistente pois afeta todos os envolvidos nas tarefas.

2.4.1 Representação gráfica

Os pacotes são representados por dois retângulos, desenhados de maneira a formar uma pasta com guia. É possível mostrar todo o conteúdo ou apenas o nome do pacote. Em seus diversos campos é possível utilizar nomes simples ou totalmente qualificados. É possível representar um diagrama de classe no interior do pacote, assim como também é razoável apenas nomeá-las quando só é necessário conhecer quais classes pertencem a quais pacotes. Na Figura 2 estão ilustradas as representações descritas anteriormente, cada uma com sua respectiva legenda, sendo a primeira uma representação genérica de pacote que apenas indica seu nome, sem qualquer conteúdo (FOWLER, 2007).

Figura 2 – Formas de representação dos pacotes

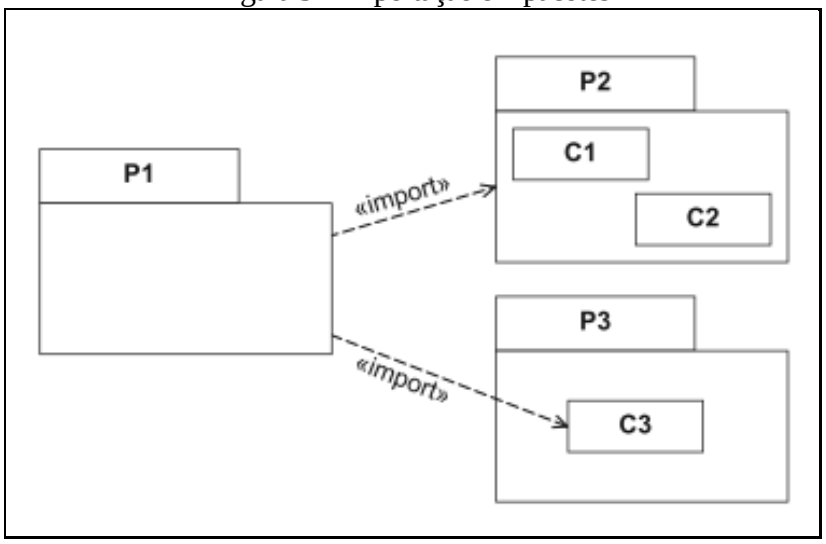


Fonte: Fowler 2005

2.4.2 Importações de Elementos/Pacotes

Os elementos em um pacote podem referenciar e se comunicar com outros elementos de outros pacotes dado que não existam restrições de visibilidade. Por exemplo, um elemento E que está localizado no pacote P1 pode ser utilizado no pacote P2 caso não haja um elemento com o mesmo nome em P2. É possível importar ou referenciar elementos de outros pacotes utilizando nomes qualificados, desta forma todos os elementos do pacote importado se tornam visíveis no pacote que importa esses elementos, tornando possível referenciá-los diretamente. Relacionamentos de importação são descritos por uma seta tracejada, que aponta para o pacote importado e tem um rótulo escrito "<<*import*>>". Apenas elementos com visibilidade externa ao pacote pode ser importado, sendo possível fazer esta operação com apenas um elemento de determinado pacote, ou mesmo um pacote inteiro como indica a Figura 3 (SEIDL, SCHOLZ, *et al.*, 2015).

Figura 3 – Importação em pacotes



Fonte: Seidl, Scholz, (2015)

2.4.3 União de pacotes - *PackageMerge*

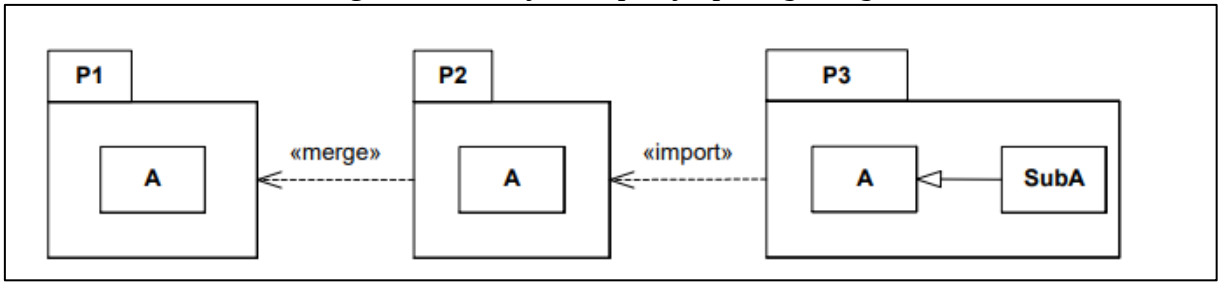
O *packageMerge* ou União de Pacotes em tradução livre é uma relação direta entre dois pacotes, que indica que o conteúdo do pacote alvo chamado *mergedPackage* está combinado no pacote origem chamado *receivingPackage* de acordo com um conjunto de regras definidas. É muito similar a uma generalização no sentido que o elemento de origem conceitualmente adiciona as características do elemento alvo às suas próprias, resultando em um elemento que combina as características de ambos. Assim como em uma subclasse que não é representada com suas características herdadas, um pacote que recebe informações de outros pacotes não as tem representadas em sua simbologia. Além disso, assim como em generalizações, um pacote não pode se unir consigo mesmo direta ou indiretamente (OMG, 2017).

Este mecanismo deve ser utilizado quando elementos definidos em diferentes pacotes têm o mesmo nome e representam o mesmo conceito. Esta notação é comumente utilizada para proporcionar diferentes definições de um conceito para diferentes propósitos, partindo de uma definição comum de base. Assim, este conceito é estendido em incrementos, com cada um destes incrementos definidos em uma *mergedPackage* à parte. Selecionando quais incrementos serão unidos, é possível obter uma definição customizada de um conceito para um fim específico. Este conceito é particularmente útil em modelagem de metadados sendo extensivamente utilizado na definição de metamodelos UML (ISO/IEC 19505-1, 2012).

Uma operação de união entre pacotes implica em um conjunto de transformações nas quais o conteúdo do pacote a ser unido (*mergedPackage*) é combinado com o conteúdo do

pacote recebedor (*receivingPackage*). Em casos nos quais alguns elementos contidos nos dois pacotes representam a mesma entidade, os conteúdos são conceitualmente mesclados em um único elemento resultante, de acordo com as regras formais descritas adiante. Assim como nas generalizações uma união entre pacotes em um modelo meramente implica as transformações, não sendo os resultados incluídos no modelo. Mesmo assim, o pacote de origem e seus conteúdos representam o resultado de uma união, da mesma forma que uma herança para classes. Assim, qualquer referência a um elemento contido no pacote de origem implica que é também uma referência aos resultados da operação de união dos pacotes, ao invés de representar apenas o incremento que está fisicamente contido em determinado pacote (ISO/IEC 19505-1, 2012).

Figura 4 – Ilustração da operação *packageMerge*



Fonte: OMG, (2017)

Na Figura 4 os pacotes P1 e P2 representam incrementos diferentes de uma mesma classe “A”. O pacote P2 é o pacote de origem que recebe os dados do pacote P1, que implica adicionar o incremento de P1::A no elemento P2::A. O pacote P3 faz uma importação do conteúdo resultante para que possa definir uma subclasse de “A”, chamada SubA. Neste caso, o elemento A no pacote P3 representa o resultado da operação de união entre P1::A e P2::A, sendo todas estas características herdadas pela classe SubA (ISO/IEC 19505-1, 2012).

É importante ressaltar que esta representação pode ser confusa, sendo que o pacote de origem P2::A é tanto um operando quanto o resultado da operação de merge, dependendo do contexto de análise. Na representação do merge da Figura 4, P2 representa o incremento que é um operador do merge. Entretanto, com respeito à operação de importação, P2 representa o resultado do merge, carregando o incremento de P1 para o pacote P3 (OMG, 2017).

2.5 APLICAÇÃO DO DIAGRAMA DE PACOTES

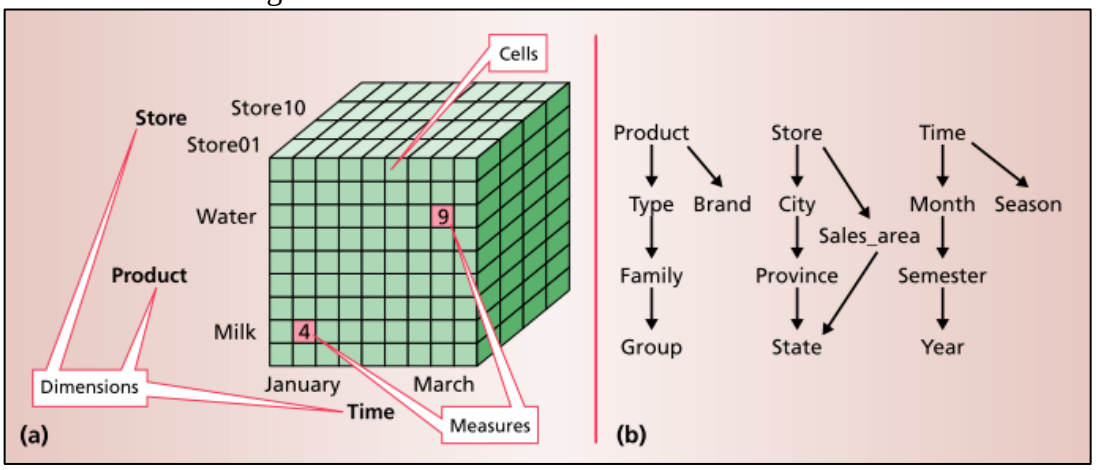
2.5.1 Modelagem Multidimensional de dados

A modelagem multidimensional organiza a informação de forma estruturada em fatos e dimensões. Um fato é algo do interesse de uma organização, descrito como um conjunto de atributos chamados medidas, contidos nas células ou pontos de um "*data cube*". A base deste conjunto de medidas é feita sobre um conjunto de dimensões, que derivam da granularidade escolhida para representar os fatos. Desta forma, as dimensões representam o contexto para analisar os fatos. Já os atributos das dimensões fornecem a especificidade que as caracterizam (TRUJILLO, SANZ, *et al.*, 2001)

Como exemplo, considere a categoria de venda de produtos de uma grande rede de lojas, que tem as seguintes dimensões: produto, loja, cliente e tempo. A

Figura 5 ilustra um cubo de dados em (a) e as hierarquias de cada dimensão em (b). Este é um caso particular que analisa medidas ao longo de produto, loja e tempo. É usual considerar os fatos relacionados como "muitos-para-muitos" entre todas as dimensões, assim como "muitos-para-um" entre um fato e cada dimensão em particular. Nota-se que a venda de produtos está relacionada com um produto, que é vendido em uma loja para um cliente. Em alguns casos, fatos também podem ter relacionamentos "muitos-para-muitos" entre dimensões em particular. Supondo que seja necessário associar venda de produtos com os recibos de compra que cada loja gera, neste caso as vendas e os recibos formam um relacionamento "muitos-para-muitos" com a dimensão dos produtos, porque um recibo pode conter vários produtos, entretanto, este recibo é gerado em uma loja, para um cliente em um tempo (TRUJILLO, SANZ, *et al.*, 2001).

Figura 5 - Um modelo multidimensional “data cube”



Fonte: Trujillo, Sanz (2001)

São abordagens comuns utilizar designs planos para modelagem multidimensional, nas quais todos os elementos modelados são representados no mesmo diagrama. Estas abordagens não são adequadas para modelos multidimensionais grandes e complexos. Entretanto, é possível utilizar os “mecanismos de pacotes” da UML com sucesso para modelagem multidimensional em três níveis de complexidade, conforme demonstrado em Trujillo et al. (2002). Neste trabalho é apresentado um guia para modelagem multidimensional em pacotes, que segundo o autor, traz os benefícios de obter modelos multidimensionais conceituais que podem ser compreendidos por designers e analistas, facilitando a comunicação entre as partes.

3 METODOLOGIA

Para o desenvolvimento deste trabalho foram utilizados alguns artigos retirados de periódicos e/ou localizados pelo Google Acadêmico, normas que regem os diagramas UML e livros, tendo base em autores renomados no assunto, possibilitando o levantamento bibliográfico dos dados, caracterizando-se uma pesquisa bibliográfica e exploratória.

As palavras-chave utilizadas foram: diagramas UML (*UML Diagrams*), diagramas de pacotes (*package diagrams*), engenharia de software (*Software Engineering*), software, qualidade de software (*quality software*), entre outras. Quanto ao período da pesquisa, o mesmo ocorreu entre o mês de maio e o mês de outubro do ano de 2022.

A pesquisa se caracteriza por uma pesquisa bibliográfica que busca a fundamentação em documentação e levantamento bibliográfico, colocando o pesquisador em contato com o

que já produziu a respeito do seu tema de pesquisa. Essa pesquisa também é definida como exploratória, que tem como objetivo proporcionar maior familiaridade com o assunto (PAULA FILHO, 2009).

Conforme aponta Gil (2008), as pesquisas exploratórias são desenvolvidas para proporcionar uma visão geral sobre o assunto, especialmente quando o tema escolhido é pouco explorado e torna-se difícil formular hipóteses precisas e operacionalizáveis. A pesquisa exploratória também depende da curiosidade e da percepção do pesquisador, pois é um processo de descoberta de informações, assim as habilidades do pesquisador não determinam uma pesquisa de qualidade (MALHOTRA, 2011).

Em relação aos resultados, o trabalho caracteriza-se como pesquisa qualitativa. Segundo Prodanov e Freitas (2013), a pesquisa qualitativa é entendida como uma relação dinâmica entre o mundo e o sujeito, ou seja, um vínculo entre o mundo objetivo e a subjetividade do sujeito que não pode ser trazido em números. O uso da pesquisa qualitativa não requer o uso de métodos e técnicas estatísticas, dessa forma o ambiente natural se torna a fonte direta para a coleta de dados, em que o pesquisador é o instrumento-chave.

4 ANÁLISE E DISCUSSÕES

É fato que a representação de sistemas em diagramas é um processo consolidado e necessário para a produção de software. Além disso, está relacionado com a qualidade do produto final e, consequentemente, ao atendimento aos requisitos da aplicação, além de permitir o correto desenvolvimento das aplicações.

Ao analisar os conceitos abordados especificamente sobre o diagrama de pacotes, constata-se que estes são uma ferramenta essencial para que a organização crie um modelo, principalmente para o desenvolvimento de sistemas de grande porte e complexos, de forma eficiente, contribuindo para sua legibilidade e facilitando a implementação, especialmente voltada ao paradigma de programação orientada a objetos. Cooperando para que o cenário exposto por Pressman (2021) possa ser melhorado, segundo o autor, o software, mesmo sendo uma das mais importantes tecnologias no cenário mundial, ainda tem seu desenvolvimento dificultoso em relação ao cumprimento dos prazos determinados, orçamentos estabelecidos e à boa qualidade do produto final.

Conforme abordado por Seidl e Scholz et. Al (2015) o diagrama de pacotes é muito desejável na modelagem de sistemas complexos, justamente porque estes modelos superam a

capacidade analítica de uma pessoa, devido à grande quantidade de elementos que são modelados, como classes, estados e ações. Além disso, é possível agrupar um sistema de grande porte em subsistemas utilizando a representação de pacotes, dividindo de maneira lógica as classes do sistema de acordo com suas funções.

É relevante observar que a literatura recomenda alguns critérios para agrupar classes em pacotes, de forma a tornar a interpretação das relações entre os pacotes mais lógica e fácil, conforme mencionado em Seidl e Scholz et. al (2015). Como abordado anteriormente, é possível agrupar elementos por coesão funcional, informacional ou de acordo com a estrutura de distribuição.

Neste viés, o critério de agrupamento de elementos e classes de acordo com sua coesão funcional, também ajuda a estruturar o desenvolvimento, facilitar a leitura de códigos e a simplificação de uma lógica muito complexa. Além disso, diminui a chance de erros de desenvolvimento, dado que cada desenvolvedor, focado em funcionalidades específicas, consome ferramentas de um pacote de utilidades, mas desenvolve elementos coerentes entre si em um mesmo pacote.

Hamilton e Miles (2006) corroboram com o princípio de que é necessária uma lógica para organização dos pacotes, seguindo um padrão de desenvolvimento bem estabelecido. Um exemplo de desenvolvimento de sistemas bem estabelecido que traz regras para nomear pacotes e classes é o padrão REST para APIs, que estrutura um serviço em camadas de forma padronizada facilitando o uso e entendimento por outros desenvolvedores, o que agrega maior qualidade ao produto final.

Com a organização em pacotes também é possível controlar o acesso, com a declaração de elementos privados dentro de um pacote, impedindo sua importação ou leitura por outras partes do programa. Além disso, é possível organizar a implantação de um sistema montando módulos de produção, onde é possível escolher incluir ou excluir um pacote de pesquisa na compilação de um programa.

Por fim, o uso mais comum do diagrama de pacotes é para fornecer uma visão geral das partes principais do sistema e as dependências entre elas, com o propósito de gerenciar estas dependências de forma a evitar fluxos cíclicos de relacionamento de pacotes que levam à instabilidade no sistema, ou seja, estudar estes diagramas pode ajudar a identificar potenciais vulnerabilidades nos projetos de software, atribuindo mais qualidade ao software e consequentemente, garantindo a satisfação do cliente.

5 CONSIDERAÇÕES FINAIS

Com base no objetivo proposto neste artigo, o qual é descrever o embasamento teórico a respeito dos diagramas de pacotes, explicitando a sua importância para o desenvolvimento de software, foi possível constatar que o diagrama de pacotes é uma ferramenta de organização importante para a modelagem UML de software, principalmente, para softwares complexos, onde são elaborados para simplificar um projeto conceitual, apresentando uma visão geral do que será desenvolvido.

Esses diagramas possibilitam o gerenciamento de dependências e simplificações de modelo, além de permitirem agrupamentos de elementos de forma conveniente, de acordo com a finalidade ou com a fase do desenvolvimento de software, diminuindo a incidência de erros no desenvolvimento. Desta forma, pode-se concluir que os diagramas de pacotes são importantes para apoiar o processo de desenvolvimento e ajudam na garantia da qualidade do software proporcionando maior confiabilidade para os sistemas desenvolvidos e garantindo a usabilidade funcional do produto desenvolvido.

Para trabalhos futuros, sugere-se a demonstração da organização em pacotes em uma aplicação mais comum com porte suficiente para obter os benefícios deste modelo, como um software do tipo “ERP” ou a modernização de softwares “legado” a partir de modelos que utilizam diagramas de pacotes.

REFERÊNCIAS

BERNERS-LEE, T. et al. RFC2396: Uniform Resource Identifiers (URI) Generic Syntax. **Internet Engineers Task Force**, 1998. Disponivel em: <https://www.ietf.org/rfc/rfc2396.txt>. Acesso em: 07 jun 2022.

DEBONI, J. E. Z. **Modelagem orientada a objetos com a UML**. 1ª. ed. [S.l.]: Futura, 2003.

FOWLER, M. **UML Essencial:** um breve guia para a linguagem-padrão de modelagem de objetos. 3ª. ed. Porto Alegre: Bookman, 2007.

GIL, A. C. **Métodos e técnicas de pesquisa social**. 6ª. ed. São Paulo: Atlas, 2008.

HAMILTON, K.; MILES, R. **Learning UML 2.0:** a pragmatic introduction to UML. 1ª. ed. Sebastopol: O'Reilly Media, 2006.

ISO/IEC 19505-1. **Information Technology - Object Management Group Unified Modeling Language (OMG UML), infrastructure**. 1ª. ed. Genebra: [s.n.], 2012.

MALHOTRA, N. K. **Pesquisa de Marketing:** foco na decisão. 3ª. ed. São Paulo: Pearson Prentice Hall, 2011.

OMG. **OMG Unified Modeling Language**. 2.5.1. ed. Milford: Object Management Group, 2017.

PAULA FILHO, W. D. P. **Engenharia de Software:** fundamentos, métodos e padrões. 3ª. ed. Rio de janeiro: LTC, 2009.

PRESSMAN, R. S. **Engenharia de software:** uma abordagem profissional. 9ª. ed. Porto Alegre: AMGH Editora Ltda., 2021.

PRODANOV, C. C.; DE FREITAS, E. C. **Metodologia do trabalho científico:** métodos e técnicas da pesquisa e do trabalho acadêmico. 2ª. ed. Novo Hamburgo: Feevale, 2013.

SEIDL, M. et al. **UML Classroom:** an introduction to object-oriented modeling. 1ª. ed. New York: Springer, 2015.

SOMMERVILLE, I. **Engenharia de software**. 9ª. ed. São Paulo: Pearson Prentice Hall, 2011.
TRUJILLO, J. et al. Designing data warehouses with OO conceptual models. **Computer**, n. 34, p. 66-75, dezembro 2001. ISSN 1558-0814.

TRUJILLO, J.; LUJÁN-MORA, S.; SONG, I.-Y. Multidimensional modeling with UML package diagrams. **Lecture Notes in Computer Science**, Berlim, v. 2503, p. 119-213, 2002. ISSN 978-3-540-45816-6.

VALENTE, M. T. **Engenharia de software moderna:** princípios e práticas para desenvolvimento de software com produtividade. 1ª. ed. [S.l.]: [s.n.], 2020.