Escola Técnica Estadual de São Paulo

Análise e Projeto de Sistemas

Projeto Orientado a Objetos (Objetos, classes) e Projetos com Reuso (desenvolvimento baseado em componentes, padrões de projeto)

São Paulo

2021

Beatriz Damas

Louisy Dalchiavon Tomazi

Pedro Cauã Silva Alves

Maria Eduarda Expedita Oliveira Canto

Vanessa Mayta Apaza

Projeto Orientado a Objetos (Objetos, classes) e Projetos com Reuso (desenvolvimento baseado em componentes, padrões de projeto)

Projeto de Pesquisa apresentada ao Curso Integrado de Desenvolvimento de Sistemas ao Ensino Médio da Escola Técnica Estadual de São Paulo

Orientador(a): Coordenador de Curso Luiz Ricardo de Souza

São Paulo

2021

RESUMO

O tema Projeto Orientado a Objetos (Objetos, Classes) e Projetos com Reuso (desenvolvimento baseado em componentes, padrões de objetos) foi abordado na presente pesquisa com o intuito de informação sobre como e porque esses métodos são úteis e importantes para a programação atual. A pesquisa aborda os principais conceitos utilizados dentro dos temas, sendo essa baseada na consulta aprofundada de seus componentes. É importante frisar que esses não são os únicos projetos existentes dentro da programação, essa sendo uma extensa e abrangente área. Com o avanço tecnológico, esses tipos de projeto tornaram-se cada vez mais presentes nos softwares, sendo o Projeto com Reuso, um efetivo agente a favor do tempo e do programador, sendo um método oportuno e eficiente, mas podendo resultar em grandes complicações quando não aplicado corretamente. Pode-se concluir que esse tema é de grande importância para as pessoas que almejam programar no futuro, sendo o aprofundamento dos tópicos de extrema importância para seu entendimento.

Palavras-chave: Programação; Software; Projetos.

ABSTRACT

The subject of Object-Oriented Design (Objects, Classes) and Reuse Design (component-based development, object patterns) was addressed in this research to provide information on how and why these methods are useful and important for current programming. The research addresses the main concepts used within the themes, which is based on in-depth consultation of its components. It is important to emphasize that these are not the only existing projects within the schedule, this being an extensive and comprehensive area. With technological advances, these types of projects have become increasingly present in software, with the Project with Reuse, an effective agent in favor of time and the programmer, being a timely and efficient method, but which can result in major complications when not applied correctly. It can be concluded that this theme is of great importance for people who aspire to program in the future, and the deepening of topics and their components is extremely important for their understanding.

Keywords: Programming; Software; Projects.

SUMÁRIO

1. INTRODUÇÃO.....5
2. PROJETO ORIENTADO A OBJETOS.....6
2.1 Programação Orientada a Objetos.....7
3. OBJETOS E CLASSES.....9
3.1 Encapsulamento.....10
3.2 Herança.....11
3.3 Interface.....11
3.4 Polimorfismo.....12
4. REUTILIZAÇÃO DE SOFTWARES.....13
5. PROJETO COM REUSO.....14
5.1 Especificação de Requisitos.....14
5.2 Análise de Componentes.....15
5.3 Modificação de Requisitos.....15
5.4 Projeto de Sistemas com Reuso.....15
5.5 Desenvolvimento e Implementação.....16
5.5.1 Desenvolvimento Baseado em Componentes.....16
5.5.2 Padrões de Projetos.....17
6. CONSIDERAÇÕES FINAIS.....18

REFERÊNCIAS BIBLIOGRÁFICAS

1. INTRODUÇÃO

Na área de Tecnologia e Informação, em desenvolvimento de softwares principalmente, a etapa de projeto é extremamente necessária, sendo essa um processo que possui uma sequência de atividades a serem feitas, todas elas com início, meio e fim. Seu objetivo é "construir" esse esquema, onde seria aplicada a identificação de elementos necessários do software de acordo com o propósito pelo qual ele está sendo criado, dando mais agilidade e segurança à parte da programação propriamente dita. Diminuindo riscos e facilitando a manipulação dos códigos, são implementados os componentes reunidos pelo projeto na linguagem de programação escolhida.

Retratado o projeto como um planejamento da programação oficial, esse é claramente um aspecto do desenvolvimento de software temporário, pois ele terá fim, uma vez que seus objetivos de conversões de aspectos da vida real, planejamento, decisão sobre a quantia que será cobrada e de tempo que será gasto forem alcançados, entretanto, seus resultados serão duradouros.

Visando demonstrar tamanha eficiência e relevância, realizamos esse trabalho apresentado os principais tópicos dentro do Projeto Orientado a Objetos e do Projeto com Reuso, explicando suas funções e onde podem ser aplicados.

2. PROJETO ORIENTADO A OBJETOS

Projeto orientado a objetos (OOD, do inglês *Object-oriented design*) é o processo de traduzir um requisito em uma especificação de implementação (o projeto). Geralmente é usado um modelo formal para essa especificação, como a UML (*Unified Modeling Language*), com o objetivo de descrever como as coisas têm que ser feitas.

Como o termo "projeto" em desenvolvimento de software é geralmente considerado uma etapa anterior à programação, antes de entrarmos no Projeto Orientado a Objetos propriamente dito, veremos três conceitos que se sobrepõem e combinam para formar o OOD: análise, programação e projeto.

Análise orientada a objetos (OOA, do inglês *Object-oriented analysis*) é um processo de desenvolvimento de software orientado a objetos no qual se busca observar um problema, sistema ou tarefa e identificar os objetos e as interações entre eles. Ou seja, é identificar o que precisa ser feito. Por exemplo: Um professor precisa corrigir um texto enviado por um aluno. Isso pode ser traduzido em um requisito de software na forma: o sistema deve permitir que um professor possa corrigir um texto enviado por um aluno, logo isso é o que deve ser feito. Ainda como parte da análise, o objetivo é identificar ações e objetos nesse requisito e poderíamos concluir que: professor é um objeto, texto enviado por um aluno é um objeto e corrigir é uma ação (do professor).

Projeto orientado a objetos (OOD, do inglês *Object-oriented design*) utiliza objetos, seus atributos (dados) e comportamentos para descrever ou modelar um domínio e isso faz jus o significado do seu termo "orientado a objeto", pois ele tem o sentido de “estar em função de" ou "ser dirigido por". Considerando o requisito anterior, o objetivo do processo de projeto é descrever o domínio formalmente, na forma de classes e interfaces.

Essa visão de "modelar um domínio" foi o propósito da primeira linguagem de programação orientada a objetos, a Simula 67. O objetivo foi criar uma ferramenta que permitisse simular um domínio por meio de objetos e trocas de mensagens entre eles. Posteriormente, a linguagem de programação Smalltalk 80 aprimorou os conceitos e é considerada, ainda hoje, a única linguagem de programação puramente orientada a objetos.

Programação orientada a objetos (OOP, do inglês *Object-oriented programming*) é o processo de utilizar uma linguagem de programação para traduzir o projeto em um programa (software) que faz exatamente o que foi requisitado anteriormente. A Programação Orientada a Objetos surgiu com a finalidade de facilitar a vida daqueles que trabalham com desenvolvimento de software, pois na POO o difícil não é desenvolver bem um software, mas sim desenvolver um software que satisfaça o cliente, ou seja, garantir que o que será entregue será realmente o que foi pedido.

Esses processos não são executados de forma exclusiva, mas, geralmente, ao mesmo tempo, com idas e vindas, com revisões e alterações conforme necessário.

Para um melhor entendimento do Projeto Orientado a Objetos, devemos aprender mais um pouco sobre a Programação Orientada a Objetos.

2.1 Programação Orientada a Objetos

Uma das características da POO é fazer com que o programador pense as coisas de forma distintas, transformando-as assim em objeto, aplicando propriedades e métodos, reduzindo assim a complexidade no desenvolvimento e manutenção de software, aumentando a produtividade.

Como a maioria das atividades que fazemos no dia a dia, programar também possui modos diferentes de se fazer. Esses modos são chamados de paradigmas de programação e, entre eles, estão a programação orientada a objetos (POO) e a programação estruturada. Quando começamos a utilizar linguagens como Java, C#, Python e outras que possibilitam o paradigma orientado a objetos, é comum errarmos e aplicarmos a programação estruturada achando que estamos usando recursos da orientação a objetos.

Na programação estruturada, um programa é composto por três tipos básicos de estruturas: Sequências que são os comandos a serem executados; As condições que são sequências que só devem ser executadas se uma condição for satisfeita (exemplos: if-else, switch e comandos parecidos); E as repetições que também são sequências, mas que devem ser executadas repetidamente até uma condição for satisfeita (for, while, do-while etc). Essas estruturas são usadas para processar a entrada do programa, alterando os dados até que a saída esperada seja gerada.

A principal diferença é que na programação estruturada, um programa é tipicamente escrito em uma única rotina (ou função) podendo, é claro, ser quebrado em sub-rotinas. Mas o fluxo do programa continua o mesmo, como se pudéssemos copiar e colar o código das sub-rotinas diretamente nas rotinas que as chamam, de tal forma que, no final, só haja uma grande rotina que execute todo o programa.

Além disso, o acesso às variáveis não possui muitas restrições na programação estruturada. Em linguagens fortemente baseadas nesse paradigma, restringir o acesso à uma variável se limita a dizer se ela é visível ou não dentro de uma função (ou módulo, como no uso da palavra-chave *static*, na linguagem C), mas não se consegue dizer de forma nativa que uma variável pode ser acessada por apenas algumas rotinas do programa. O contorno para situações como essas envolve práticas de programação danosas ao desenvolvimento do sistema, como o uso excessivo de variáveis globais. Vale lembrar que variáveis globais são usadas tipicamente para manter estados no programa, marcando em qual parte dele a execução se encontra.

Já o Projeto Orientado a Objetos é uma alternativa para o programador poder aproximar o manuseio das estruturas de um programa ao manuseio das coisas do mundo real e, é particularmente, útil na tentativa de compreender um problema a fim de propor soluções computacionais na criação de sistemas de informação. O funcionamento deste sistema se dá através do relacionamento e troca de mensagens entre os objetos e podendo implementar classes entre eles.

As vantagens do Projeto Orientado a Objetos podem ser inúmeras dependendo do tipo de software que o programador está desenvolvendo, mas as principais são a facilidade de manutenção, a reutilização dos objetos e para vários sistemas, existe um nítido mapeamento entre as entidades do mundo real para objetos no sistema.

A Programação Orientada a Objetos surgiu como uma alternativa para as características da programação estruturada. O intuito da sua criação também foi o de aproximar o manuseio das estruturas de um programa ao manuseio das coisas do mundo real, daí o nome "objeto" como uma algo genérico, que pode representar qualquer coisa tangível.

Esse novo paradigma se baseia principalmente em dois conceitos chave: classes e objetos. Todos os outros conceitos, igualmente importantes, são construídos em cima desses dois.

> “Cada projeto possui um ciclo de vida, que ajuda a definir o início e término de cada etapa, o que deve ser realizado e por quem deve ser executado (matriz de responsabilidade do projeto). Serve para dar alicerce ao tripé de sucesso dos projetos: tempo/custo/qualidade. A entrega deve ser feita no prazo estipulado, dentro do orçamento apontado, com nível de qualidade atendendo as necessidades do cliente comprador” (VARGAS, 2009).

3. OBJETOS E CLASSES

“Objeto: é uma abstração encapsulada que tem um estado interno dado por uma lista de atributos cujos valores são únicos para o objeto. O objeto também conhece uma lista de mensagens que ele pode responder e sabe como responder cada uma”. (BORGES; CLINIO, 2015, p. 13).

Imagine que você comprou um carro recentemente e decide modelar esse carro usando programação orientada a objetos. O seu carro tem as características que você estava procurando: um motor 2.0 híbrido, azul escuro, quatro portas, câmbio automático etc. Ele também possui comportamentos que, provavelmente, foram o motivo de sua compra, como acelerar, desacelerar, acender os faróis, buzinar e tocar música. Podemos dizer que o carro novo é um objeto, onde suas características são seus atributos (dados atrelados ao objeto) e seus comportamentos são ações ou métodos.

Indo para o lado mais técnico, um objeto é uma entidade no sistema de software que é utilizada para declarar uma variável que referenciará o objeto (declarar o objeto) ou instanciar o objeto (alocar o objeto em memória). Ele possui um conjunto de operações que operam nesse estado e representam instâncias de entidades do mundo real e do sistema. As operações (lógica contida em uma classe para designar-lhe um comportamento, onde comportamento, pode ser considerado um conjunto de ações possíveis sobre o objeto, vulgo, métodos de classe) associadas ao objeto fornecem serviços para outros objetos. Eles são independentes entre si e encapsulam representações de informação e estado, esse sendo representado por um conjunto de atributos que podem ser armazenados por meio dos seus estados e a reagir a mensagens enviadas a ele, assim como se relacionar e enviar mensagens a outros objetos. A comunicação entre objetos acontece por meio de passagem de mensagens, que são basicamente solicitações de um objeto para outro, para que o objeto receptor produza um resultado. Elas são implementadas como chamadas de procedimentos.

Uma coisa a se prestar muita atenção, é o fato de que em algumas linguagens (como Java Script e Self) os objetos são projetados pela declaração de protótipos, enquanto em outras (como Java e PHP) eles são declarados como classes.

Voltando para o exemplo, seu carro é um objeto seu, mas na loja onde você o comprou existiam vários outros, muito similares, com quatro rodas, volante, câmbio, retrovisores, faróis, dentre outras partes. Observe que, apesar do seu carro ser único (por exemplo, possui um registro único no Departamento de Trânsito), podem existir outros com exatamente os mesmos atributos, ou parecidos, ou mesmo totalmente diferentes, mas que ainda são considerados carros. Podemos dizer então que seu objeto pode ser classificado (isto é, seu objeto pertence à uma classe) como um carro, e que seu carro nada mais é que uma instância dessa classe chamada "carro".

Assim, abstraindo um pouco a analogia, uma classe é um conjunto de características e comportamentos que definem o conjunto de objetos pertencentes à essa classe. Repare que a classe em si é um conceito abstrato, como um molde, que se torna concreto e palpável através da criação de um objeto. Chamamos essa criação de instanciação da classe, como se estivéssemos usando esse molde (classe) para criar um objeto.

A classe inclui declarações de todos os atributos e serviços que devem ser associados a um objeto dessa classe. Dentro da classe pode vir a existir várias subclasses, que são as características particulares de uma classe.

Para organizar essas classes surge o conceito de pacote. Pacote é um conjunto de classes, ou seja, guarda classes e outros pacotes logicamente semelhantes ao pacote que os contém. Podemos visualizar os pacotes como diretórios ou pastas, nos quais podemos guardar arquivos (classes) e outros diretórios (pacotes) que tenham conteúdo relacionado com o pacote que os contém.

Na “produção” de um pseudocódigo, primeiro define-se uma classe e, então, define-se os atributos. Essa classe pode ser usada/importada/incluída em outros arquivos/módulos. Ela pode ser inicializada, construindo (guarde essa palavra) um objeto a partir dela, que é preenchido com os valores específicos da instância.

Sabendo que as duas bases da POO (Programação Orientada a Objetos) são os conceitos de classe e objeto, desses derivam alguns outros conceitos extremamente importantes ao paradigma, que não só o definem como são as soluções de alguns problemas da programação estruturada. Os conceitos em questão são o encapsulamento, a herança, as interfaces e o polimorfismo.

3.1 Encapsulamento

Ainda usando a analogia do carro, sabemos que ele possui atributos e métodos, ou seja, características e comportamentos. Os métodos do carro, como acelerar, podem usar atributos e outros métodos do carro como o tanque de gasolina e o mecanismo de injeção de combustível, respectivamente, uma vez que acelerar gasta combustível.

No entanto, se alguns desses atributos ou métodos forem facilmente visíveis e modificáveis, como o mecanismo de aceleração do carro, isso pode dar liberdade para que alterações sejam feitas, resultando em efeitos colaterais imprevisíveis. Nessa analogia, uma pessoa pode não estar satisfeita com a aceleração do carro e modifica a forma como ela ocorre, criando efeitos colaterais que podem fazer o carro nem andar, por exemplo.

Dizemos, nesse caso, que o método de aceleração do seu carro não é visível por fora do próprio carro. Na POO, um atributo ou método que não é visível de fora do próprio objeto é chamado de "privado" e quando é visível, é chamado de "público".

Mas então, como sabemos como o nosso carro acelera? É simples: não sabemos. Nós só sabemos que para acelerar, devemos pisar no acelerador e de resto o objeto sabe como executar essa ação sem expor como o faz. Dizemos que a aceleração do carro está encapsulada, pois sabemos o que ele vai fazer ao executarmos esse método, mas não sabemos como - e na verdade, não importa para o programa como o objeto o faz, só que ele o faça.

O mesmo vale para atributos. Por exemplo: não sabemos como o carro sabe qual velocidade mostrar no velocímetro ou como ele calcula sua velocidade, mas não precisamos saber como isso é feito. Só precisamos saber que ele vai nos dar a velocidade certa. Ler ou alterar um atributo encapsulado pode ser feito a partir de getters e setters (colocar referência).

Esse encapsulamento de atributos e métodos impede o chamado vazamento de escopo, onde um atributo ou método é visível por alguém que não deveria vê-lo, como outro objeto ou classe. Isso evita a confusão do uso de variáveis globais no programa, deixando mais fácil de identificar em qual estado cada variável vai estar a cada momento do programa, já que a restrição de acesso nos permite identificar quem consegue modificá-la.

3.2 Herança

No nosso exemplo, você acabou de comprar um carro com os atributos que procurava. Apesar de ser único, existem carros com exatamente os mesmos atributos ou formas modificadas. Digamos que você tenha comprado o modelo Fit, da Honda. Esse modelo possui uma outra versão, chamada WR-V (ou "Honda Fit Cross Style"), que possui muitos dos atributos da versão clássica, mas com algumas diferenças bem grandes para transitar em estradas de terra: o motor é híbrido (aceita álcool e gasolina), possui um sistema de suspensão diferente, e vamos supor que além disso ele tenha um sistema de tração diferente (tração nas quatro rodas, por exemplo). Vemos então que não só alguns atributos como também alguns mecanismos (ou métodos, traduzindo para POO) mudam, mas essa versão "cross" ainda é do modelo Honda Fit, ou melhor, é um tipo do modelo.

Quando dizemos que uma classe A é um tipo de classe B, dizemos que a classe A herda as características da classe B e que a classe B é mãe da classe A, estabelecendo então uma relação de herança entre elas. No caso do carro, dizemos então que um Honda Fit "Cross" é um tipo de Honda Fit, e o que muda são alguns atributos (paralama reforçado, altura da suspensão etc.), e um dos métodos da classe (acelerar, pois agora há tração nas quatro rodas), mas todo o resto permanece o mesmo, e o novo modelo recebe os mesmos atributos e métodos do modelo clássico.

3.3 Interface

Muitos dos métodos dos carros são comuns em vários automóveis. Tanto um carro quanto uma motocicleta são classes cujos objetos podem acelerar, parar, acender o farol etc., pois são coisas comuns a automóveis. Podemos dizer, então, que ambas as classes "carro" e "motocicleta" são "automóveis".

Quando duas (ou mais) classes possuem comportamentos comuns que podem ser separados em uma outra classe, dizemos que a "classe comum" é uma interface, que pode ser "herdada" pelas outras classes. Note que colocamos a interface como "classe comum", que pode ser "herdada", porque uma interface não é exatamente uma classe, mas sim um conjunto de métodos que todas as classes que herdarem dela devem possuir (implementar) - portanto, uma interface não é "herdada" por uma classe, mas sim implementada. No mundo do desenvolvimento de software, dizemos que uma interface é um "contrato": uma classe que implementa uma interface deve fornecer uma implementação a todos os métodos que a interface define, e em compensação, a classe implementadora pode dizer que ela é do tipo da interface. No nosso exemplo, "carro" e "motocicleta" são classes que implementam os métodos da interface "automóvel", logo podemos dizer que qualquer objeto dessas duas primeiras classes, como um Honda Fit ou uma motocicleta da Yamaha, são automóveis.

Um pequeno detalhe: uma interface não pode ser herdada por uma classe, mas sim implementada. No entanto, uma interface pode herdar de outra interface, criando uma hierarquia de interfaces. Usando um exemplo completo com carros, dizemos que a classe "Honda Fit Cross" herda da classe "Honda Fit", que por sua vez herda da classe "Carro". A classe "Carro" implementa a interface "Automóvel" que, por sua vez, pode herdar (por exemplo) uma interface chamada "MeioDeTransporte", uma vez que tanto um "automóvel" quanto uma "carroça" são meios de transporte, ainda que uma carroça não seja um automóvel.

3.4 Polimorfismo

Vamos dizer que um dos motivos de você ter comprado um carro foi a qualidade do sistema de som dele. Mas, no seu caso, digamos que a reprodução só pode ser feita via rádio ou bluetooth, enquanto que no seu antigo carro, podia ser feita apenas via cartão SD e pendrive. Em ambos os carros está presente o método "tocar música”, mas, como o sistema de som deles é diferente, a forma como o carro toca as músicas é diferente. Dizemos que o método "tocar música" é uma forma de polimorfismo, pois dois objetos, de duas classes diferentes, têm um mesmo método que é implementado de formas diferentes, ou seja, um método possui várias formas, várias implementações diferentes em classes diferentes, mas que possuem o mesmo efeito ("polimorfismo" vem do grego poli = muitas, morphos = forma).

4. REUTILIZAÇÃO DE SOFTWARES

O desenvolvimento de sistemas de informação é tipicamente conhecido como um processo caro e lento. A pressão que acompanha esse processo, bem como a busca constante por redução significativa de custos e alcance de ganhos incomparáveis em qualidade do produto, associados à redução do tempo de desenvolvimento fazem com que o resultado final e a excelência fiquem aquém do desejado. Uma possível solução, que pode não resolver todos os problemas, mas pode ajudar a lidar com esse processo é a reutilização.

Na maioria das disciplinas de engenharia de software, os sistemas são projetados com base na composição de componentes existentes que foram utilizados em outros sistemas. Até então, esse tipo de engenharia tinha como base o desenvolvimento tradicional, porém tem sido reconhecido que, para alcançar software com mais qualidade, de forma mais rápida e com baixo custo, é necessário adotar um processo de desenvolvimento baseado na reutilização generalizada e sistemática de software.

Existem diferentes tipos de Reuso de Softwares: Por meio do Reuso de sistemas de aplicações, onde todo o sistema pode ser reutilizado pela sua incorporação, sem mudança, em outros sistemas (sistemas de prateleira) ou pelo desenvolvimento de famílias e aplicações. Já no Reuso de Componentes, esses de uma aplicação que variam desde subsistemas até objetos isolados podem ser reutilizados. E no Reuso de funções, os componentes de software que implementam uma única função podem ser reutilizados.

Para utilizar essa ferramenta, é preciso encontrar componentes reutilizáveis apropriados, sendo o responsável pelo reuso desse, ou seja, o programador, certo de que os componentes se comportarão como especificado e de que serão confiáveis. Esses devem ser bem documentados para ajudar o usuário a compreendê-los e adaptá-los a uma nova aplicação.

No contexto social, a imitação ou reutilização de ideias são vistas como falta de originalidade, porém em desenvolvimento de software isso pode resultar em um aumento significativo na produtividade e uma redução considerável no tempo de execução das tarefas.

Apesar de os primeiros conceitos sobre reutilização de software terem sido idealizados no final da década de 1960, eles não receberam muita atenção da indústria e das universidades até os anos 80, quando a complexidade dos sistemas começou a aumentar e as empresas foram forçadas a procurar por métodos mais eficientes de construção de sistemas para refletir as necessidades do mercado.

Em 1980 foi estabelecido o primeiro projeto de pesquisa universitário no tema de reutilização, na Universidade da Califórnia, coordenado por Peter Freeman. Atualmente, existem duas principais conferências sobre reutilização, a ICSR – *International Conference on Software Reuse* e o SSR – *Symposium on Software Reusability*.

É notório que a reutilização, quando feita de forma intencional, planejada e controlada, os benefícios alcançados são relevantes.

5. PROJETO COM REUSO

Para que essa reutilização de software seja realizada de forma controlada para a obtenção de benefícios, o Projeto com Reuso tem papel importante. Ele constrói o software a partir de componentes reutilizáveis, tendo o objetivo reaproveitar partes do projeto em si, diminuindo o custo, aumentando a produtividade no desenvolvimento e proporcionando o compartilhamento do conhecimento durante as fases de desenvolvimento.

Existem muitos motivos para usar o mecanismo de reutilização. A manutenção e qualidade do projeto e do processo de desenvolvimento são um dos principais motivos para aumentar a produtividade. A qualidade do projeto para reutilização também é geralmente superior à projetada para esse fim. Isso ocorre porque geralmente é investido mais dinheiro em design, documentação e testes.

Entre os benefícios da utilização do reuso, estão a diminuição da "quantidade" de software a ser desenvolvido, a redução dos custos e riscos e uma entrega mais rápida do produto ao cliente. Porém, há desvantagens, como o desenvolvimento de um produto que não atenda aos requisitos do cliente, uma gerência complexa das versões dos componentes, a dificuldade de compreender a parte "reutilizável" do projeto original e a dificuldade na evolução dos sistemas.

O início do Projeto de Sistemas com Reuso tem alguns aspectos já pré-definidos. Sendo eles a Especificação de Requisitos, a Análise de Componentes, Modificação de Requisitos, Projeto de Sistemas com Reuso, o Desenvolvimento e a Implementação.

5.1 Especificação de Requisitos

A Especificação de Requisitos tem o intuito de entender o que é desejado pelo cliente o qual solicitou o software, após essa etapa, o método de Engenharia de software orientada a reuso entra em cena com a etapa de "Análise de componentes".

5.2 Análise de Componentes

Essa etapa diz respeito a análise dos componentes (elemento de que encapsula diversas funcionalidades uteis para softwares) de outros projetos que sejam reutilizáveis para suprir a necessidade de implementações para requisitos comuns entre o software sendo desenvolvido e de onde esses componentes estão sendo retirados, para essa análise, existem certas características as quais os componentes devem seguir: deve refletir abstrações estáveis do domínio do problema, ocultar a maneira como seu estado é representado, ser independente o máximo possível e todas as exceções devem ser parte da interface do componente.

Analisados os componentes, passamos para a próxima etapa, a “Modificação de requisitos”.

5.3 Modificação de Requisitos

Nessa etapa os requisitos implementados antes são discutidos e analisados novamente com o conhecimento dos componentes que temos a disposição, fazendo com que requisitos possam ser excluídos, incluídos ou modificados de acordo com os componentes encontrados, e dependendo do caso, de necessidades extrema, pode-se até organizar mais uma “Análise de componentes” em busca de soluções e alternativas.

Reinventados os requisitos e talvez novamente analisados os componentes, partimos para a principal fase desse tópico, o “Projeto de sistemas com reuso”.

5.4 Projeto de Sistemas com Reuso

Nessa parte é feito um tipo de plano, é projetado o framework (conjunto de instruções com o objetivo de ajudar o setor de TI em tarefas como implantação, manutenção, suporte, controle e gerenciamento) do sistema ou algo pronto é reutilizado, tendo em mente que os projetistas já sabem os componentes que serão utilizados, sejam eles reutilizados ou criados por necessidades do próprio software. Ele fica como uma espécie de mapa para guiar a próxima etapa e até para ter uma pequena noção de como o software irá ficar. Após essa nossa importante etapa, temos o “Desenvolvimento e implementação”.

5.5 Desenvolvimento e Implementação

Se trata do desenvolvimento (no caso de nenhum software de fora servir para alguma função necessária) e implementação tanto de softwares desenvolvidos quanto outros que já estavam prontos, além dos componentes identificados antes, tudo isso a partir do Projeto com reuso. Dentro dessa etapa, existe uma vertente chamada “Desenvolvimento baseado em Componentes”.

5.5.1 Desenvolvimento Baseado em Componentes

O desenvolvimento baseado em componentes, têm como objetivo reutilizar o código e artefatos utilizados no processo do desenvolvimento. Esse objetivo surgiu da frustação de que o desenvolvimento orientado a objetos não tinha um meio para se reutilizar o código.

Um componente é uma parte independente e substituível que preenche uma clara função. Eles são mais abstratos do que as classes de objetos.

Um dos maiores desafios para que a reutilização seja possível é a construção de componentes o mais genéricos possível, para que eles possam ser utilizados em diversas situações. Dentro da engenharia de componentes, são observadas algumas características positivas e outras nem tanto.

Os riscos são menores ao usar um componente já existente em relação ao desenvolvimento de algo a partir do zero, existe um aumento da produtividade, tendo em vista a redução de esforços pela equipe de desenvolvimento. A qualidade e confiabilidade do produto são maiores, pois o componente reutilizado já foi testado e aprovado. E no caso de componentes comerciais, pode-se não ter acesso ao código fonte ou depender de direitos autorais e licenças.

Apesar de parecer bastante promissor, existe uma resistência da parte das equipes de desenvolvimento, pois exige forte padronização (investimento de tempo e controle de qualidade) e documentação (investir mais tempo nos artefatos). A adoção de novas práticas de desenvolvimento geralmente encontra forte resistência a mudanças por parte da equipe.

Além desse importante vertente do desenvolvimento, temos o Padrão de Projeto.

5.5.2 Padrões de Projetos

Um Padrão de Projeto é uma descrição (sendo essa, abstrata, podendo ser reutilizada em diferentes casos) de problemas reais resolvidos através de implementações orientadas a objetos, sendo também, a essência da sua solução. Um padrão nomeia, abstrai e identifica as classes e instâncias participantes, seus papéis colaborações e a distribuição de responsabilidades.

Esses padrões apresentam elementos essenciais, como nome, descrição do problema, descrição da solução e as consequências.

De acordo com o livro de *Gamma* (2000, p.63), ele apresenta as divisões necessárias para se fazer a documentação de um padrão, sendo elas o nome e classificação, intenção do padrão, motivação, aplicabilidade, estrutura, participantes, colaborações, consequências, implementação, exemplo de código, usos conhecidos e padrões relacionados. Ainda de acordo com esse livro, ele apresenta 23 padrões de projeto: *Abstract Factor*y (95), *Adapter* (140), *Bridge* (151), *Builder* (104), *Chain of Responsability* (212), *Command* (222), *Composite* (160), *Decorator* (170), *Façade* (179), *Factory Method* (212), *Flyweight* (187), *Interpreter* (231), *Iterator* (244), *Mediator* (257), *Memento* (266), *Observer* (274), *Prototype* (121), *Proxy* (198), *Singleton* (130), *State* (284), *Strategy* (292), *Template Method* (301) e *Visitor* (305)

Esses Padrões de Projetos ainda podem ser classificados segundo o seu propósito, que no caso seriam os padrões de criação (refere-se à criação de objetos para subclasses ou para outros objetos), padrões estruturais (se preocupam com a composição das classes e dos objetos), padrões comportamentais (tratam a forma como classes e objetos, interagem e distribuem responsabilidade). Eles também podem ser classificados segundo o seu escopo, que seria as classes (que lidam com os relacionamentos entre classes e subclasses) e os objetos (que lidam com os relacionamentos entre objetos).

Com isso, conclui-se que os Padrões de Projeto são abstrações de alto nível que documentam soluções de projetos.

Por fim, como em qualquer outro desenvolvimento de software, o sistema tem que passar pela “Validação de Sistema” a fim de conferir se está tudo nos parâmetros com o cliente e com testes.

“Cada padrão descreve um problema no nosso ambiente e o cerne da sua solução, de tal forma que você possa usar essa solução mais de um milhão de vezes, sem nunca o fazer da mesma maneira” (CHRISTOPHER ALEXANDER)

6. CONSIDERAÇÕES FINAIS

No presente trabalho, abordamos a utilidade e eficiência de tais projetos para o mundo da programação, sendo eles, partes fundamentais no aprendizado da área e notórios facilitadores do desenvolvimento de software por parte do programador. Podemos perceber que, apesar de serem trabalhosos, a satisfação do cliente sempre é a melhor opção. Em conjunto com outras vertentes da criação de software, esses terão um exímio desempenho e notória funcionalidade.

A pesquisa foi de extrema importância para um melhor compreendimento do tema, sendo esse, imprescindível para a área de Tecnologia da Informação.

REFERÊNCIAS BIBLIOGRÁFICAS

ALÉSSIO, Simone Cristina. **Projeto Orientado a Objetos**. Uniasselvi, 2016. Acesso em 27 de jun. de 2021 às 20:53.

CARVALHO, Victorio Albani; TEIXEIRA, Giovany Frossard. **Programação Orientada a Objetos: Curso técnico de informática**. Colatina: IFES, 2012. Acesso em 27 de jun. de 2021 às 20:59.

ENTENDENDO A ORIENTAÇÃO A OBJETOS. DevMedia, 2011. Disponível em: <https://www.devmedia.com.br/entendendo-a-orientacao-a-objetos/22342>. Acesso em 30 de jun. de 2021 às 17:01.

FIGUEIREDO, Eduardo. POO e Padrões de Projetos. UFMG, 2012. Disponível em: <https://homepages.dcc.ufmg.br/~figueiredo/disciplinas/2012a/reuso/reuso-aula03.pdf>. Acesso em 04 de jul. de 2021 às 13:08.

GAMMA, Erich; HELM, Richard; JOHNSON, Ralph; VLISSIDES, John; tradução Luiz A. Meireles Salgado. **Padrões de Projeto: Soluções reutilizáveis de software orientado a objetos** – Porto Alegre: Bookman, 2007. Acesso em 04 de jul. de 2021 às 13:13.

GOMES, Jackson. **Programação Orientada a Objetos com Phyton**. Open-source utilizado na disciplina Linguagem de Programação Orientada a Objetos do Departamento de Computação do CEULP/ULBRA. Acesso em 04 de jul. de 2021 às 15:48.

PADRÕES DE PROJETO DE SOFTWARE BASEADO EM COMPONENTES APLCIADOS A UM SISTEMA JAVA PARA ACADEMIA DE MUSCULAÇÃO. DevMedia, 2011. Disponível em: <https://www.devmedia.com.br/padroes-de-projeto-de-software-baseado-em-componentes-aplicados-a-um-sistema-java-para-academia-de-musculacao/19138>. Acesso em 04 de jul. de 2021 às 16:40.

POO: O QUE É PROGRAMAÇÃO ORIENTADA A OBJETOS? Alura, 2019. Disponível em:<https://www.alura.com.br/artigos/poo-programacao-orientada-a-objetos?gclid=Cj0KCQjwh_eFBhDZARIsALHjIKct5fAxxedYP7e2srfBjHa0yoAEv7_YbXEKBdj0akhjlpEvgEgoW4caAulrEALw_wcB>. Acesso em 03 de jul. de 2021 às 12:33.

TORRES, Márcio. **Programação Orientada a Objetos: Conceitos e Princípios implementados na linguagem Java**, 2017. Acesso em 27 de jun. de 2021 às 21:16.

UNESP. Projeto com Reuso: Engenharia de Software, 2005. Disponível em: <https://www.dcce.ibilce.unesp.br/~ines/cursos/eng_soft/aula10.PDF>. Acesso em 04 de jul. de 2021 às 16:47.

UNESP. Projeto Orientado a Objetos: Engenharia de Software, 2006. Disponível em: <https://www.dcce.ibilce.unesp.br/~ines/cursos/eng_soft/2006/aula11_ProjetoOrientadoObjeto.pdf>. Acesso em 27 de jun. de 2021 às 22:37.

UNICAMP. Programação Orientada a Objetos: Reuso em POO. André Santachè, Abril 2015. Disponível em: <https://www.ic.unicamp.br/~santanch/teaching/oop/2015-1/slides/poo0601-reuso-poo-v02.pdf>. Acesso em 02 de jul. de 2021 às 17:48.

ZAKAS, Nicholas – **The Principles of Object-Oriented Java Script** – San Francisco: No Starch Press, 2014. Acesso em 5 de jul. de 2021 às 18:57.