PADRÕES DE PROJETO

MICHEL CÉSAR DOS SANTOS

EXPEDIENTE

Coordenador(a) de Conteúdo
Nader Ghoddosi
Projeto Gráfico e Capa
Arthur Cantareli Silva
Editoração
Renata Sguissardi Coli
Design Educacional
Lucio Ferrarese
Revisão Textual
Salen Nascimento
Ilustração
Eduardo Aparecido Alves
Geison Ferreira da Silva
Fotos
Shutterstock e Envato

FICHA CATALOGRÁFICA

N964 Núcleo de Educação a Distância. **SANTOS,** Michel César dos.

Padrões de Projeto / Michel César dos Santos. - Florianópolis, SC: Arqué, 2024.

176 p.

ISBN papel 978-65-279-0142-6
ISBN digital 978-65-279-0141-9

1. Padrões 2. Projeto 3. EaD. I. Título.

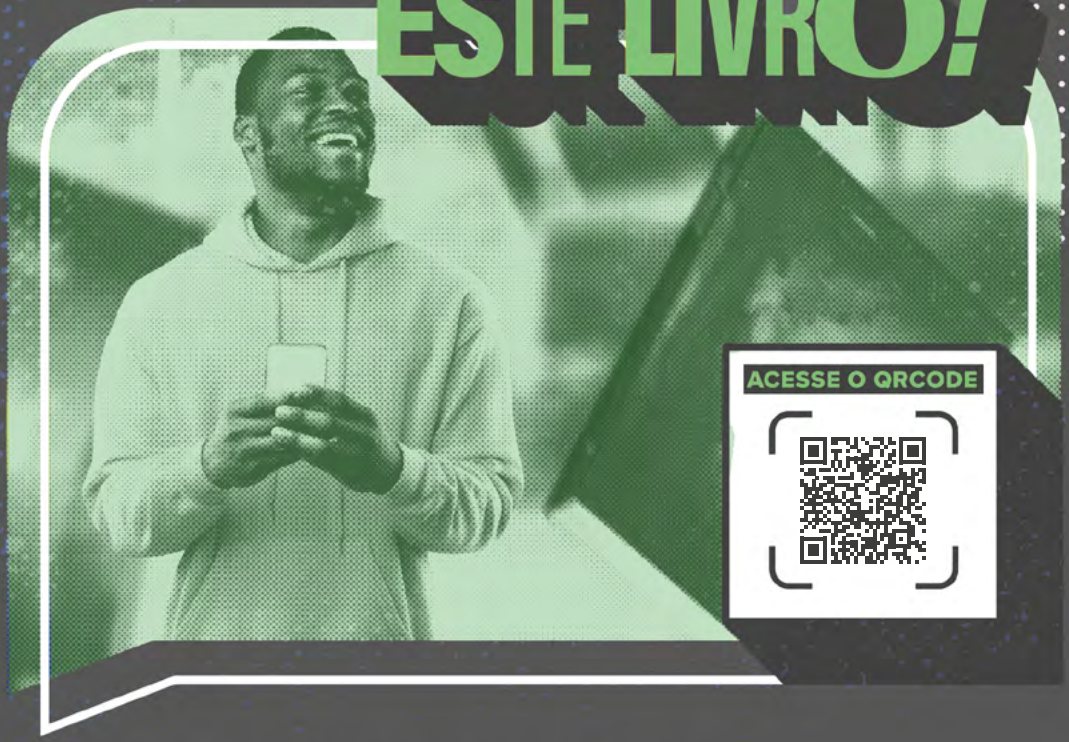
CDD - 620.0042

Bibliotecária: Leila Regina do Nascimento - CRB- 9/1722.

Ficha catalográfica elaborada de acordo com os dados fornecidos pelo(a) autor(a).

Impresso por:

AVALIE ESTE LIVRO!



CRIAR MOMENTOS DE APRENDIZAGENS INESQUECÍVEIS É O NOSSO OBJETIVO E POR ISSO, **GOSTARÍAMOS DE SABER COMO FOI SUA EXPERIÊNCIA.**

Conta para nós! leva ***menos de 2 minutos*** *Vamos lá?!*

DIGITE O CÓDIGO

03507346

Aa

RESPONDA A PESQUISA

?

•••

RECURSOS DE IMERSÃO

PENSANDO JUNTOS

Este item corresponde a uma proposta de reflexão que pode ser apresentada por meio de uma frase, um trecho breve ou uma pergunta.

APROFUNDANDO

Utilizado para temas, assuntos ou conceitos avançados, levando ao aprofundamento do que está sendo trabalhado naquele momento do texto.

EU INDICO

Utilizado para agregar um conteúdo externo.

ZOOM NO CONHECIMENTO

Utilizado para desmistificar pontos que possam gerar confusão sobre o tema. Após o texto trazer a explicação, essa interlocução pode trazer pontos adicionais que contribuam para que o estudante não fique com dúvidas sobre o tema.

PRODUTOS AUDIOVISUAIS

Os elementos abaixo possuem recursos audiovisuais. **Recursos de mídia disponíveis no conteúdo digital do ambiente virtual de aprendizagem.**

PLAY NO CONHECIMENTO

Professores especialistas e convidados, ampliando as discussões sobre os temas por meio de fantásticos podcasts.

EM FOCO

Utilizado para aprofundar o conhecimento em conteúdos relevantes utilizando uma linguagem audiovisual.

INDICAÇÃO DE FILME

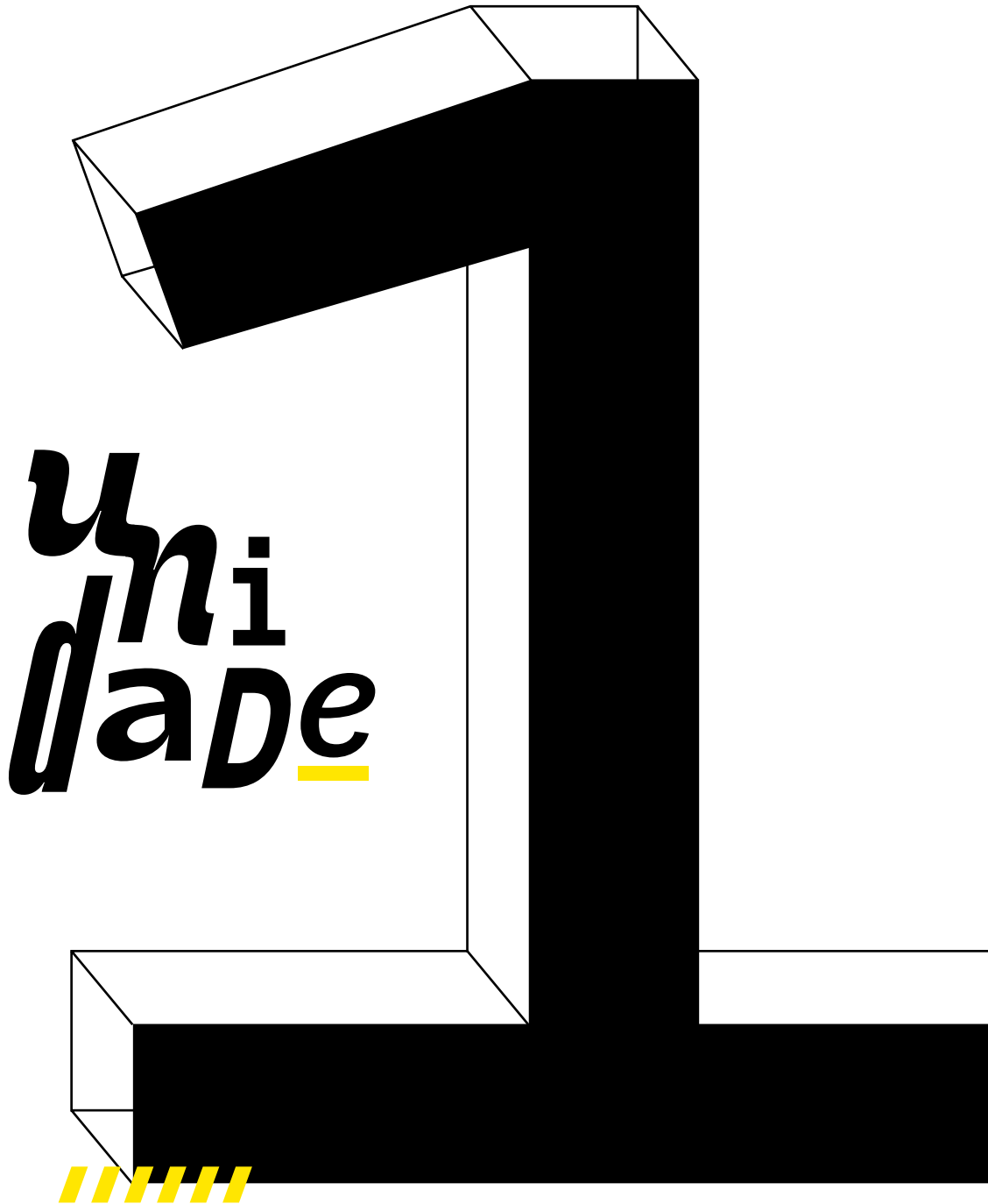
Uma dose extra de conhecimento é sempre bem-vinda. Aqui você terá indicações de filmes que se conectam com o tema do conteúdo.

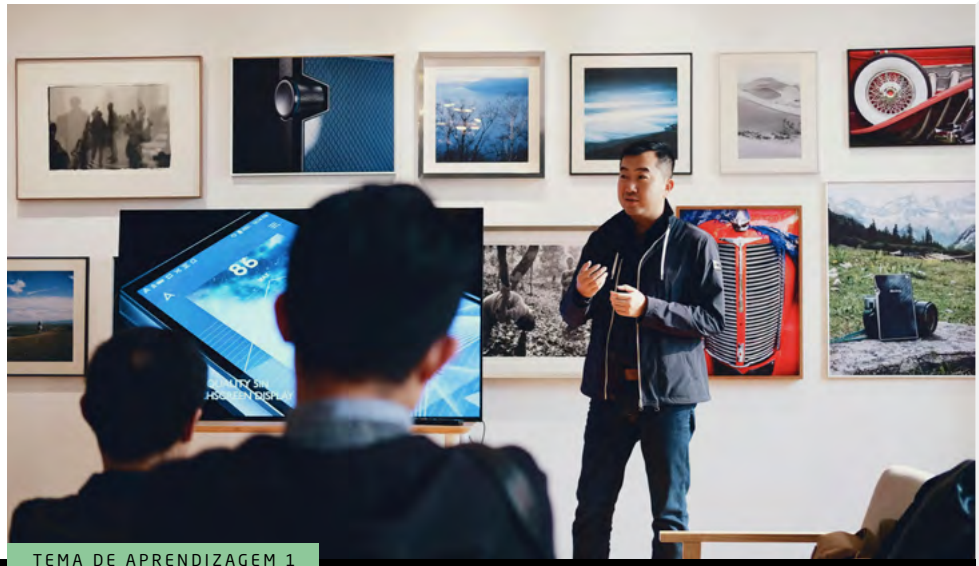
INDICAÇÃO DE LIVRO

Uma dose extra de conhecimento é sempre bem-vinda. Aqui você terá indicações de livros que agregarão muito na sua vida profissional.

SUMÁRIO

UNIASSELVI





TEMA DE APRENDIZAGEM 1

INTRODUÇÃO AOS PADRÕES DE PROJETO

MINHAS METAS

- Ressaltar as características de um bom projeto.
- Definir padrões de projeto, ressaltando a importância da sua aplicação nos problemas que ocorrem, com uma certa frequência, na engenharia de software.
- Posicionar historicamente os padrões de design, assim como sua evolução dentro do projeto de desenvolvimento de software.
- Indicar os principais tipos de padrões estabelecidos no mercado, levando em conta seu propósito.
- Questionar a necessidade de se aprender padrões de projeto.
- Elucidar os impactos na qualidade do projeto de desenvolvimento de software, assim como na implementação das soluções.
- Exemplificar padrões de projeto aplicados em outras tecnologias emergentes no cenário atual.

INICIE SUA JORNADA

Estudante, em nosso trabalho como desenvolvedor, nos deparamos com inúmeras situações em que devemos lidar com problemas e criar soluções para eles. Como o desenvolvimento de software é uma atividade criativa e, de certa forma, artesanal, como você deve identificar soluções para essas questões?

Com isso em mente, devemos refletir acerca da **profissionalização do trabalho**, pois sempre se procurou uma forma de **otimizar o tempo**, reduzir o esforço ou reaproveitar o trabalho, reutilizando o que já foi criado em situações anteriores. Com o surgimento do desenvolvimento de software, também surgiu a **necessidade** de os desenvolvedores **otimizarem o trabalho**, os quais encontravam barreiras, como a dificuldade na reutilização de código, a falta de padronização, a falta de flexibilidade no código e afins.

Embora o desenvolvimento de software seja considerado uma atividade criativa, alguma padronização pode ser utilizada, minimizando a ocorrência dos problemas, de forma que os padrões de projetos, também chamados de ***design patterns***, ocupem um importante lugar no processo de desenvolvimento por meio de soluções já conhecidas e implementadas por outros e **contribuindo para a manutenção** ao longo do tempo.

Com a implementação e aprendizado dos padrões de projeto, a atividade de desenvolvimento se tornará **melhor estruturada**, mostrando ao desenvolvedor distintos pontos de abordagem, e tornando a solução final mais flexível e mais fácil de se manter.

Como um exercício mental, considere que você está desenvolvendo um software em seu trabalho e se depara com problemas relacionados à criação de objetos, como a ocorrência de muitas peças repetidas, de forma que poderia aplicar uma solução baseada em padrões de projeto, como você resolveria esse problema no desenvolvimento?

Os princípios de padronização e design de software devem ser analisados e estudados, pois proporcionarão um maior envolvimento do desenvolvedor no processo de criação de software, o design aplicado e seu reaproveitamento.

Faz-se necessária, assim, uma reflexão a respeito do tema da padronização de projetos, como esses padrões podem auxiliar na condução do seu projeto de desenvolvimento de software.

PLAY NO CONHECIMENTO

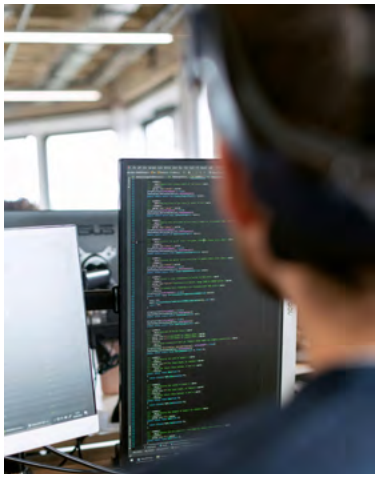
Os padrões de projeto auxiliaram inúmeras empresas ao redor do mundo a obter sucesso em sua trajetória de desenvolvimento, fazendo com que elas abordassem as melhores práticas em sua rotina de desenvolvimento. Neste podcast, você verificará casos em que o *design patterns* trouxe histórias de sucesso para essas organizações. **Recursos de mídia disponíveis no conteúdo digital do ambiente virtual de aprendizagem.**

VAMOS RECORDAR?

Os padrões de projeto envolvem inúmeros conceitos associados à arquitetura de software ou programação orientada a objetos. Acerca desse assunto, sugiro o vídeo, a seguir, que apresenta os conceitos de padrões de projetos e sua aplicação, e faz parte da série de vídeos chamada *Dicionário do Programador*.

Disponível em: **https://www.youtube.com/watch?v=J-lHpiu-Twk**.

DESENVOLVA SEU POTENCIAL

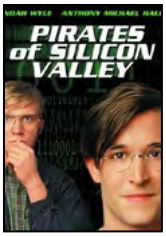


Nas atividades de planejamento, as lições aprendidas são fundamentais para entender o que se passou durante a execução, avaliando as soluções adotadas. Com isso, as soluções poderão ser adotadas em um futuro, minimizando o problema e facilitando o caminho a ser percorrido.

Os padrões de projeto funcionam de forma semelhante, oferecendo soluções e um caminho já percorrido por desenvolvedores na implementação de seu código.

INDICAÇÃO DE FILME

Piratas do Vale do Silício



O longa-metragem conta a trajetória das empresas Apple e Microsoft, desde a sua criação em garagens até sua ascensão, ao tornarem-se as potências que conhecemos na atualidade. Cenas do filme retratam a definição das equipes, negociações e o desenvolvimento de softwares, buscando produtividade e redução de tempo.

Refletindo sobre a história: no que diz respeito à informática e o desenvolvimento de software, o filme é considerado um clássico. Ele mostra os bastidores da criação dessas grandes empresas e suas contribuições.

UM BOM PROJETO

Criar um bom projeto é algo que sempre deve ser perseguido pela equipe de desenvolvimento. Diversos padrões e ferramentas auxiliam nessa criação, como parâmetros de qualidade, testes, arquitetura, outros.

No desenvolvimento não é diferente. Os desenvolvedores devem buscar constantemente as melhores práticas, como código reutilizável e extensibilidade, ou seja, o software deve ser flexível para melhorias futuras e escalabilidade, de maneira que funcione bem para um número cada vez maior de usuários.

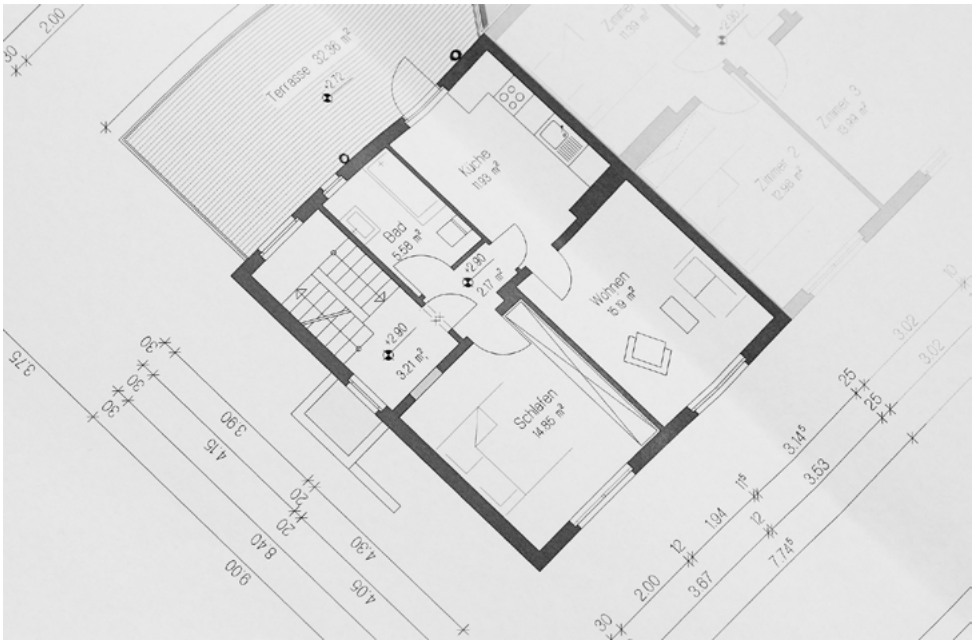
As ações mencionadas são importantes para o desenvolvimento de qualquer projeto. Essa definição, no entanto, é subjetiva, podendo variar, dependendo do cenário em que a empresa está inserida. "Utilizar padrões de projeto é uma maneira de aumentar a flexibilidade dos componentes do software e torná-los de mais fácil reutilização. Contudo, isso, às vezes, vem com um preço de tornar os componentes mais complicados" (Shvets, 2021, p. 37).

O QUE SÃO OS PADRÕES DE PROJETO?

Padrões de projetos (*design patterns*) são soluções já planejadas e implementadas para problemas conhecidos no desenvolvimento de software, as quais foram bem documentadas e podem ser seguidas pelos demais desenvolvedores.

> Um padrão de projeto sistematicamente nomeia, motiva e explica uma solução de projeto geral, que trata um problema recorrente de projeto em sistemas orientados a objetos. Ele descreve o problema, a solução, quando aplicar a solução e suas consequências. Também dá sugestões e exemplos de implementação. A solução é um arranjo genérico de objetos e classes que resolve o problema. A solução é customizada e implementada para resolver o problema em um contexto particular (Gamma, 1995 *apud* Valente, 2004, p. 4).

Segundo Shvets (2021), eles são como plantas de obra pré-fabricadas, que podem ser customizadas para solucionar um problema de projeto recorrente em seu código. Assim, faz-se necessário diferenciar algoritmo de padrões, pois muitas pessoas confundem esses conceitos. O algoritmo é uma sequência bem definida para se atingir o fim desejado e o padrão de projetos instrui para esse objetivo; a sequência, no entanto, é definida por você, como na analogia acima.



Os indivíduos envolvidos na produção de software perceberam que muitos dos problemas de projetos de sistemas eram recorrentes. Esse foi um dos fatores que motivou o surgimento dos *design patterns*, ou padrões de projetos, os quais são aplicados a problemas clássicos na produção de softwares (Valentim; Souza Neto, 2005).

Grupos de Padrões

Os padrões de projeto podem ser classificados em quatro grupos principais. São eles:

PADRÕES DE CRIAÇÃO
Os padrões desse grupo se referem à criação de objetos para maior flexibilidade e reaproveitamento de código.
PADRÕES DE ESTRUTURA
Definem a montagem de objetos em estruturas maiores, inserindo os padrões.

PADRÕES DE COMPORTAMENTO
Focam na comunicação e definições de responsabilidades dos objetos.
PADRÕES DE ARQUITETURA
São padrões para problemas abrangentes, de alto nível, em situações que envolvem a forma com que a aplicação será organizada de acordo com a infraestrutura disponível e necessária.

UM POUCO DE HISTÓRIA DOS PADRÕES DE PROJETO

Não é muito preciso afirmarmos quem criou os padrões de projeto, pois esses parâmetros são soluções adotadas para problemas conhecidos, de forma que elas começaram com o desenvolvimento em si, no entanto, as raízes dos padrões são inspiradas na arquitetura e engenharia civil, uma vez que as alternativas empregadas nessas áreas eram frequentemente aplicadas para resolver problemas recorrentes nas estruturas físicas.

> O conceito de padrões foi primeiramente descrito por Christopher Alexander em Uma Linguagem de Padrões. O livro descreve uma 'linguagem' para o projeto de um ambiente urbano. As unidades dessa linguagem são os padrões. Eles podem descrever quão alto as janelas devem estar, quantos andares um prédio deve ter, quão largas as áreas verdes de um bairro devem ser, e assim em diante (Shvets, 2021, p. 32).

Depois desse livro, foram lançadas outras publicações, sendo que, em 1994, *Design Patterns: elements of reusable object-oriented software*, de Erich Gamma e outros autores, foi apelidado de "o livro da Gangue dos Quatro (Gang of Four)" (Shvets, 2021, p. 33).

Esse livro foi muito bem aceito na comunidade de desenvolvimento de software, impactando de maneira significativa, já que estabeleceu uma linguagem comum aos desenvolvedores no que dizia respeito a estruturas de desenvolvimento. Desde o lançamento dessa publicação, padrões em inúmeros projetos foram aplicados e muitos outros são criados a cada dia, levando a comunidade de desenvolvedores a otimizar e tornar seus códigos mais claros e flexíveis.

Primeira Geração de Padrões de Projeto

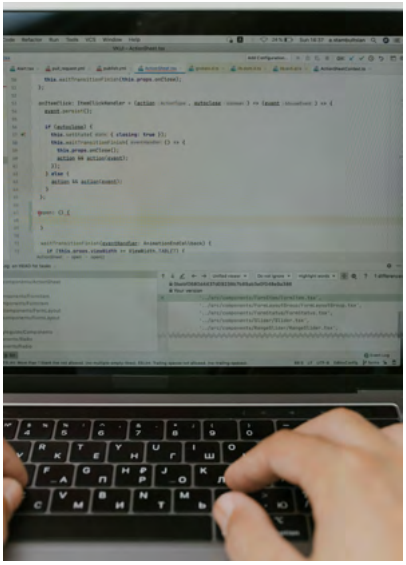
Os parâmetros incluídos no livro *Gang of Four* podem ser considerados de primeira geração. Nesse livro, os autores identificaram e documentaram 23 padrões de design, agrupando-os nas três principais categorias a seguir: padrões de criação, padrões estruturais e padrões comportamentais. Eles estabeleceram um marco e fixaram as bases para o uso e evolução dos *design patterns*.

INDICAÇÃO DE LIVRO

Padrões de Projeto: soluções reutilizáveis de software orientado a objetos

O livro é um clássico do desenvolvimento da orientação a objetos, pois conta com muitos exemplos relacionados à composição, podendo aumentar a reutilização do seu código.

Os Padrões de Projetos Adotados para Novas Tecnologias



Desde a primeira geração, os padrões de projeto evoluíram e atualmente abrangem diversas tecnologias e novos paradigmas de programação, como gestão de fluxos assíncronos, computação em nuvem ou aplicativos móveis, que seguem uma estrutura um pouco diferenciada.

Podemos observar, no que diz respeito à aplicação dos padrões de projeto em *data science* ou em controles de transformadores, um leque cada vez maior das aplicações dos padrões de projeto.

OS ANTIPADRÕES

Assim como existem padrões de projeto, existem os antipadrões, que são as soluções intuitivas, os quais, porém, levam a problemas estruturais e de manutenção e destacam práticas a serem evitadas.

APROFUNDANDO

O termo foi definido por Andrew Koenig, em 1995, um ano após o lançamento do livro *Gang of Four*. "Um antipadrão sugere outros padrões aplicáveis que podem oferecer soluções melhores" (Freeman; Freeman, 2007, p. 479). Adicionalmente, afirma-se que: "um antipadrão sempre parece ser uma boa solução, mas, ao ser aplicado, acaba produzindo o efeito contrário" (Freeman; Freeman, 2007, p. 479).

Como exemplo, podemos citar:

GOD OBJECT
Uma classe que faz tudo no sistema, de difícil manutenção.
SPAGHETTI CODE
Código que é difícil de entender devido à sua estrutura complexa.

IMPORTÂNCIA DOS PADRÕES DE PROJETO

Podemos ressaltar que, para o sucesso de qualquer projeto, documentação e orientação, as chamadas lições aprendidas acabam por ser fundamentais, pois ações futuras serão embasadas em situações já enfrentadas. Dessa forma, contar com a sabedoria das lições aprendidas reduzirá nosso tempo de desenvolvimento, assim como beneficiara a qualidade do software, uma vez que ele poderá ser embasado em um desenvolvimento já experimentado e aplicado amplamente, conduzindo a um projeto bem elaborado, de forma que podemos afirmar a relevância dos padrões de projeto no desenvolvimento de software.

> Os design patterns desempenham um relevante papel na construção de projetos, pois foram desenvolvidos para serem reutilizados. Assim, subsidiando aos arquitetos de software uma ferramenta que irá evitar que os problemas sejam resolvidos a partir de princípios elementares, ou seja, fazendo tudo desde o início (Valentim; Souza Neto, 2005, p. 69).

EM FOCO

Acesse seu ambiente virtual de aprendizagem e confira a aula referente a este tema. **Recursos de mídia disponíveis no conteúdo digital do ambiente virtual de aprendizagem.**

NOVOS DESAFIOS

Os padrões de projeto e suas aplicações são extremamente úteis no dia a dia das empresas, permitindo que elas reduzam o tempo de desenvolvimento e, consequentemente, diminuindo o tempo de implementação das soluções e otimizando os lucros dos projetos. Além disso, o código ficará extremamente bem estruturado, reduzindo a manutenção, o que levará à redução de custos durante o ciclo de vida do software implementado, itens que são extremamente valiosos para uma empresa que atue no ramo de software.

Os padrões de projeto expandem, cada dia mais, as suas áreas de aplicação. Isso estimulará os profissionais a se especializarem, cada vez mais, nessas aplicações e se valorizem, uma vez que o profissional com esse conhecimento se destacará no competitivo mercado de trabalho.

No início deste tema, questionou-se a respeito da solução que poderia ser adotada, propondo um exercício mental a respeito. Por meio deste conteúdo, a resposta pôde ser desenvolvida. Em se tratando dos padrões de projeto, como verificamos, os padrões de criação poderiam ser aplicados para solucionar as questões relacionadas à criação de objetos, sobretudo os padrões *factory method* e *builder*.

Percebemos que se trata de um ramo muito interessante para que você adquira mais conhecimentos e especialize-se para que se diferencie no disputado mercado de trabalho. Discorremos acerca da importância dos padrões de projeto e do auxílio que esses parâmetros nos fornecem para definir as soluções de desenvolvimento encontradas no dia a dia do desenvolvimento, revelando-se como resposta à nossa questão inicial, que era a reflexão por meio das soluções dos problemas de software.

AUTOATIVIDADE

1. "Utilizar padrões de projeto é uma maneira de aumentar a flexibilidade dos componentes do software e torná-los de mais fácil reutilização. Contudo, isso, às vezes, vem com um preço de tornar os componentes mais complicados" (Shvets, 2021, p. 37).

   A respeito dos padrões de projeto é correto afirmar:

   a) Contribuem com a manutenção, facilitando ao longo do tempo.
   b) Contribuem com a manutenção, dificultando ao longo do tempo.
   c) Dificultam a manutenção e degradam a performance.
   d) Dificultam a manutenção e aumentam performance.
   e) Diminuem a escalabilidade.

2. "Eles aplicaram o conceito de padrões de projeto para programação. O livro mostrava 23 padrões que resolviam vários problemas de projeto orientado a objetos e se tornou um best-seller rapidamente" (Shvets, 2021, p. 33).

   A respeito do livro *Design Patterns: elements of reusable object-oriented software*, analise as afirmativas a seguir:

   I - Os padrões incluídos no livro *Gang of Four* podem ser considerados os padrões da primeira geração.
   II - No livro, os autores identificaram e documentaram 23 padrões de design.
   III - No livro, os padrões foram agrupados em três principais categorias: padrões de criação, padrões estruturais e padrões comportamentais.

   É correto o que se afirma em:

   a) I, apenas.
   b) III, apenas.
   c) I e II, apenas.
   d) II e III, apenas.
   e) I, II e III.

3. "Um antipadrão sempre parece ser uma boa solução, mas, ao ser aplicado, acaba produzindo o efeito contrário" (Freeman; Freeman, 2007, p. 479).

   A respeito dos antipadrões, assinale a alternativa correta:

   a) Padrões de projeto e antipadrões dificultam a manutenção.
   b) O padrão *Spaghetti Code* se refere a uma classe que faz tudo no sistema, de difícil manutenção.
   c) São soluções intuitivas, porém, levarão a problemas de manutenção.
   d) São soluções planejadas, que não levarão a problemas.
   e) São soluções definitivas, que não levarão a problemas.

REFERÊNCIAS

FREEMAN, E.; FREEMAN, E. **Use a cabeça**: padrões de projetos. 2. ed. Rio de Janeiro: Alta Books, 2007.

GAMMA, E. *et al*. **Design Patterns**: elements of reusable object-oriented software. Reading: Addison-Wesley, 1995.

SHVETS, A. **Mergulho nos padrões de projeto**. [*S. l.*: *s. n.*], 2021.

VALENTE, E. C. **Padrões de interação e usabilidade**. 2004. Dissertação (Mestrado em Computação) – Instituto de Computação, Universidade Estadual de Campinas, Campinas, 2004.

VALENTIM, R. A. de M.; SOUZA NETO, P. A. de. O impacto da utilização de *design patterns* nas métricas e estimativas de projetos de software: a utilização de padrões tem alguma influência nas estimativas? **Revista da FARN**, Natal, v. 4, n. 1, p. 63-74, 2005.

GABARITO

1. Alternativa A.

   A opção A está correta, pois faz parte dos benefícios estudados no tema acerca dos padrões de projeto.
   A opção B está errada, pois a manutenção só é facilitada ao longo do tempo.
   As opções C e D estão erradas, pois a manutenção é facilitada.
   A opção E está errada, pois a escalabilidade fica maior.

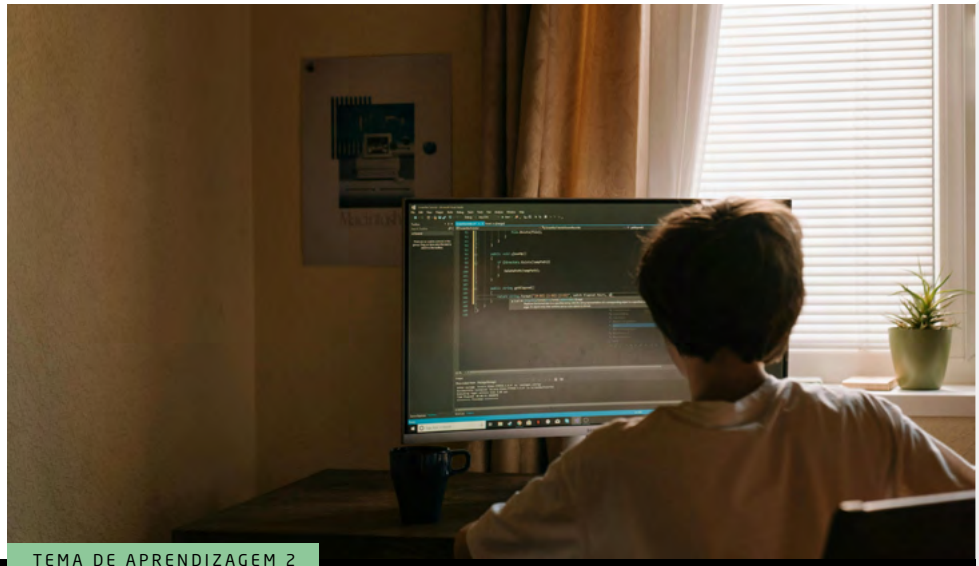
2. Alternativa E.

   Todas as afirmativas estão corretas e são evidenciadas no presente tema de aprendizagem.

3. Alternativa C.

   A opção C está correta, de acordo com a definição citada no tema.
   A alternativa A está errada, pois padrões de projeto facilitam a manutenção.
   As alternativas B, D e E são incorretas, pois antipadrões sempre são causadores de problemas.

MINHAS ANOTAÇÕES



TEMA DE APRENDIZAGEM 2

PADRÕES DE CRIAÇÃO

MINHAS METAS

- Estabelecer o uso dos padrões de criação e sua função nas soluções dos problemas de desenvolvimento.
- Definir o padrão de criação Singleton e sua disposição de classes.
- Definir o padrão de criação Factory e sua disposição de classes.
- Definir o padrão de projeto Abstract Factory e sua disposição de classes.
- Definir o padrão de projeto Builder e sua disposição de classes.
- Definir o padrão Prototype e sua disposição de classes.
- Conscientizar o estudante acerca do alinhamento da teoria dos padrões de projeto com a prática de mercado.

INICIE SUA JORNADA

No processo de desenvolvimento de software, um dos aspectos cruciais é a **criação de seu código**. Diante dessa realidade, podemos nos questionar se um padrão de criação é importante para o desenvolvimento de sistemas operacionais – por que esses tipos de padrões são essenciais? Tais modelos são, de fato, princípios fundamentais adotados, que moldam a arquitetura do aplicativo e mostram os desafios enfrentados pelos engenheiros de software em décadas de programação. Não são, portanto, somente ferramentas técnicas, que fornecem soluções prontas, mas também podem ser considerados princípios essenciais para definir a estrutura de um projeto de programa de computador e sua codificação.

A aplicação dos **padrões de criação** em seus projetos trará inúmeros benefícios. À medida que você mergulha no conhecimento a respeito dos padrões de criação, tais benefícios se tornam, cada vez mais, claros. O pleno entendimento, no entanto, se dará por meio da experimentação e implementação dos padrões em seu próprio projeto.

À medida que você avança em seus estudos, pergunte-se sempre para que os padrões, de fato, servem e procure identificar que não se tratam apenas de soluções técnicas. Observe que elas acabam contribuindo em diversas áreas, como no design, na comunicação ou no próprio desenvolvimento. Dessa forma, o conhecimento adquirido pode se converter em sabedoria, sendo extremamente benéfico para sua trajetória profissional no desenvolvimento de software.

Portanto, cada conceito, padrão desenvolvido ou boas práticas deve ser encarado com entusiasmo e determinação, pois constituem o alicerce do desenvolvimento de software.

Considere, estudante, que você foi contratado como desenvolvedor para uma grande empresa de software, a qual passa por muitas questões relativas à criação de objetos e soluções, uma vez que, durante sua definição e criação, tais soluções não são bem definidas e não são flexibilizadas, trazendo inúmeros problemas futuros, como a manutenção do software.

Como você abordaria o planejamento e desenvolvimento das soluções? Será que os padrões de criação poderiam contribuir com as atividades?

A resposta para esse questionamento poderá ser observada e respondida ao longo do tema, sendo que ao término da leitura você terá condições de responder esse questionamento.

Os padrões de projeto são soluções que devem ser bem analisadas e antes de serem soluções que possam ser aplicadas mecanicamente, você deverá refletir a respeito do seu uso, situações em que podem ser aplicadas as mesmas soluções propostas ou com algumas alterações.

PLAY NO CONHECIMENTO

Neste podcast, você compreenderá os antipadrões e como evitar a sua implementação, as quais, muitas vezes, são um tanto intuitivas, mas causam problemas de manutenção ou performance ao software, que está sendo desenvolvido. Acesse e entenda! **Recursos de mídia disponíveis no conteúdo digital do ambiente virtual de aprendizagem.**

VAMOS RECORDAR?

Os padrões de criação ajudam a definir a estrutura do software e constituem a fundação do software, dentre eles, podemos citar o padrão Singleton, amplamente utilizado. Este vídeo ilustra esse parâmetro para sua compreensão: **https://www.youtube.com/watch?v=Q4APeeSpClo**.

DESENVOLVA SEU POTENCIAL

Os padrões de projeto podem ser divididos em:

PADRÕES DE CRIAÇÃO

São *design patterns* que se concentram na maneira de criar objetos ou classes, gerenciando a sua instância e dependências. Exemplos incluem: Singleton, Factory Method e Abstract Factory, que ajudam a centralizar e organizar a criação de objetos em um sistema, promovendo flexibilidade e manutenibilidade do código.

PADRÕES DE ESTRUTURA

Definem como classes e objetos são compostos para formar estruturas maiores. Eles facilitam a organização e a comunicação entre os componentes do sistema. Exemplos incluem: Adapter, Decorator e Composite, que permitem que objetos trabalhem juntos de maneira eficiente e flexível, garantindo a coesão e a baixo acoplamento.

COMPORTAMENTO

Lidam com a interação entre objetos e responsabilidades distribuídas entre eles. Eles definem como os objetos se comunicam e como o comportamento é delegado entre eles. Exemplos incluem: Observer, Strategy e Command, que permitem definir algoritmos, comportamentos e responsabilidades de forma independente, promovendo a reutilização e a extensibilidade do código.

ARQUITETURA

São padrões de alto nível que definem a estrutura geral e a organização de sistemas de software. Eles fornecem diretrizes abstratas para projetar sistemas escaláveis, robustos e flexíveis. Exemplos incluem: MVC (Model-View-Controller), MVVM (Model-View-View-Model) e Microservices, que orientam a divisão de responsabilidades, a separação de preocupações e a escalabilidade de sistemas complexos.

Cada um deles é adequado para uma situação específica, abordaremos a respeito dos projetos de criação.

> Efetivamente, os designs patterns podem contribuir para suavizar a transição do modelo de análise para o modelo de implementação, haja vista, balizam-se em tipos recorrentes de problemas de implementação em projetos de software. Assim, podendo ser vistos como um modelo de 'microarquitetura' aplicado a um dado problema, o que estabelece uma linguagem comum entre a construção de aplicações que permeiam um mesmo contexto (Valentim; Souza Neto, 2005, p. 70).

PENSANDO JUNTOS

Um aspecto muito importante para entendermos – e uma boa prática – é a compreensão de que se deve programar para uma interface e não para uma implementação. O que se pretende dizer com isso?

Para que o projeto seja flexível, devemos inserir novas chamadas e novos trechos de código, sem fazer com que o existente deixe de funcionar. Uma boa forma de se fazer isso é usar uma interface intermediária, ou seja, não realizar chamadas diretamente para uma classe, mas para uma interface que se comunica com ela. Dessa forma, você pode implementar várias chamadas, ou alterar a classe, sem causar problemas a todas as chamadas.

PADRÕES DE CRIAÇÃO

Os padrões de criação são utilizados com a finalidade de estabelecer os moldes para que uma determinada funcionalidade seja instituída, servindo como a estrutura do desenvolvimento.

Padrão Singleton

Esse padrão é relacionado à criação de instância de classes. Ele garante que teremos apenas uma instância da classe criada.

Muitos dos objetos que instanciamos precisam de apenas uma instância. Um erro de execução poderia ser criado se instanciássemos mais de uma vez. Como exemplo, podemos citar o objeto que referencia um arquivo de configuração ou uma placa gráfica no seu computador. Por esse motivo, o padrão faz-se necessário.

Você pode se perguntar por que precisaríamos de um padrão, ao passo que, poderíamos utilizar variáveis globais e convencionar o uso. Nesse caso, porém, recursos desnecessários, como memória, poderiam ser desperdiçados.

O padrão Singleton também se trata de uma convenção. Para iniciar o seu desenvolvimento, devemos observar os passos em comum a seguir:

1. Quando criamos métodos para essa classe, devemos confeccionar o construtor padrão privado. Assim, impedimos que outras classes acessem esse bloco de códigos.

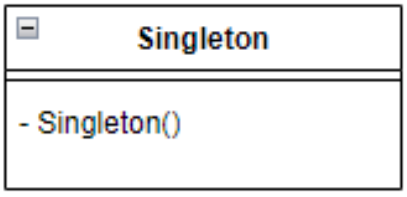


Figura 1 – Classe padrão Singleton / Fonte: Shvets (2021, p. 154).

Descrição da Imagem: a figura apresenta um quadrado com uma linha horizontal, dividido em duas partes. No quadro de baixo, temos o método construtor Singleton(), com o sinal de menos do lado esquerdo. Fim da descrição.

2. Ao definir um expediente de criação como estático (getInstance()), como o método construtor privado criado acima (Singleton()), tal fórmula cria o objeto e o salva no campo estático (instancia). Dessa forma, todas as chamadas ao método estático retornarão o mesmo objeto já criado.

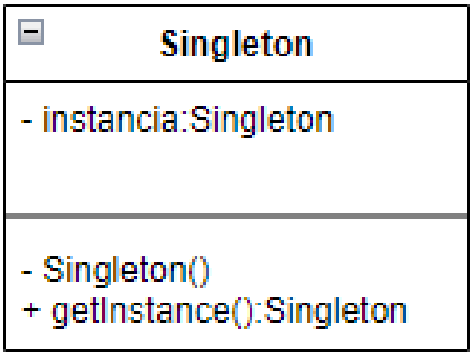


Figura 2 - Classe padrão Singleton e método estático / Fonte: Shvets (2021, p. 154).

Descrição da Imagem: a figura apresenta um quadrado com duas linhas horizontais, que o dividem em três partes. Na primeira, temos a inscrição Singleton; na segunda, o sinal de menos, ao lado direito, e a palavra "instância" seguida por dois pontos; e na terceira, a inscrição Singleton, seguida por um abre e fecha de parênteses, assim como, na linha de baixo, o termo GetInstance, um abre e fecha de parênteses, o sinal de dois pontos e a palavra Singleton. Fim da descrição.

> O padrão Singleton é uma estrutura que define classes que serão instanciadas uma única vez. Classes, como spooler, windowmanager, filesystem, entre outras, são implementadas dessa forma. Tais classes devem prover um mecanismo de controle para a criação de uma instância única, oferecendo acesso global ao único objeto criado, bem como permitir uma fácil especialização em subclasses (Albuquerque; Rojas; Ribeiro, 2010, p. 16).

Analogia: o governo é um excelente exemplo de um padrão Singleton. Um país pode ter apenas um executivo oficial. Independentemente das identidades pessoais dos indivíduos que formam governos, o título "O Governo de X" é um ponto de acesso global que identifica o grupo de pessoas no comando (Shvets, 2021, p. 154).

Padrão Factory

Esse padrão também é conhecido como construtor virtual. Ele consiste em um método que fornece uma interface para criação de objetos de uma classe superior (superclasse) e permite que as classes-filhas (subclasses) façam alterações nos tipos de objetos que estão sendo criados.

Por exemplo: você tem um sistema de delivery que controla restaurantes, mas quer passar a administrar, também, lanchonetes. Se o código é acoplado à classe restaurante, você terá que adicionar alterações para utilizar lanchonete; e se surgir outro meio, terá que fazer alterações novamente, de forma que seu código fique frágil e deselegante.

Segundo o método Factory, você deve criar uma classe-fábrica, que define o método de cada uma das subclasses (restaurante, lanchonete), sendo os objetos retornados pelo método fábrica chamados de produtos.

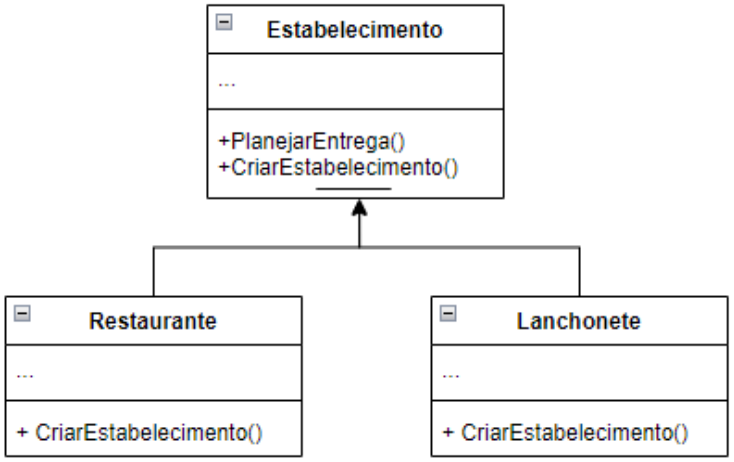


Figura 3 - Diagrama de classes do padrão Factory / Fonte: o autor.

Descrição da Imagem: representação de três quadros de classes: estabelecimento, restaurante e lanchonete. Cada um deles é dividido por duas linhas horizontais, sendo que seu nome fica na primeira delas. O quadro chamado Estabelecimento contém as expressões: PlanejarEntrega() e CriarEstabelecimento(). Ele se liga a dois outros quadros por uma seta, com os nomes Restaurante e Lanchonete. Fim da descrição.

Segundo Shvets (2021, p. 83), "as subclasses só podem retornar tipos diferentes de produtos se esses produtos tiverem uma classe ou interface base em comum".

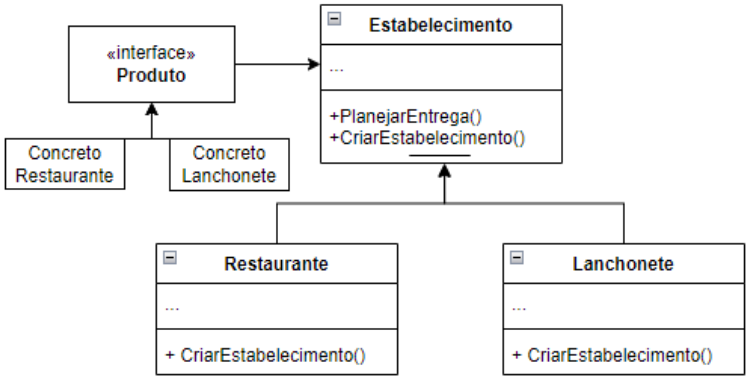


Figura 4 - Diagrama de classes do padrão Factory / Fonte: o autor.

Descrição da Imagem: em um quadrado, linhas delimitam o espaço com a descrição "interface Produto". À direita, há também um quadrado, representando a superclasse, identificada como "Estabelecimento". Dentro dela, há o método construtor: CriarEstabelecimento(), delineado por linhas mais espessas. Abaixo, dividindo o restante do espaço, estão dois quadrados: um rotulado como "Restaurante" e outro como "Lanchonete". Abaixo do quadrado "Interface", temos duas implementações de objeto concreto, identificadas como "Concreto Restaurante" e "Concreto Lanchonete". Fim da descrição.

Um outro exemplo com diálogo de plataforma cruzada:

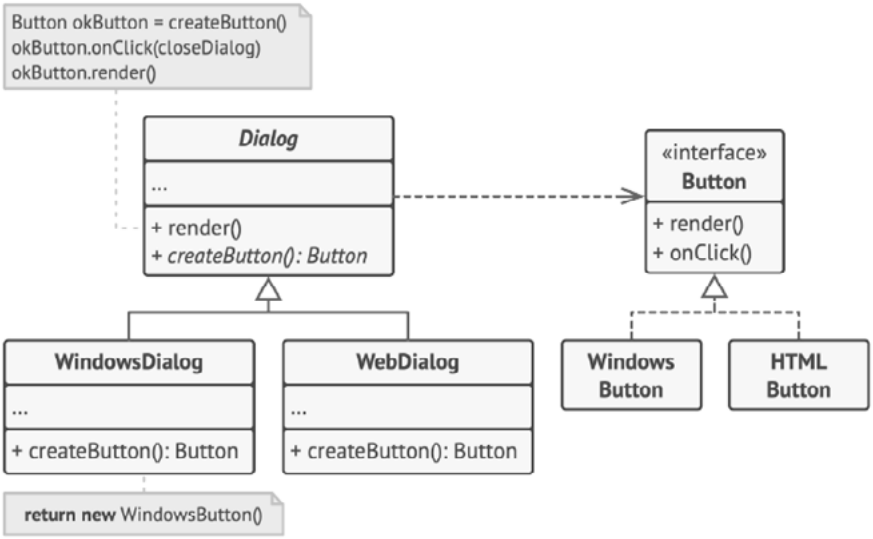
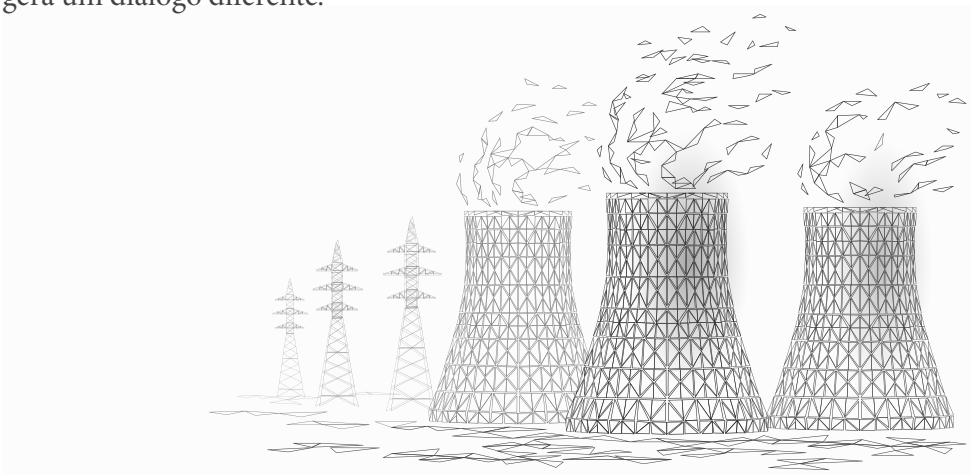


Figura 5 – Diagrama de classes do padrão Factory – exemplo de plataforma cruzada / Fonte: Shvets (2021, p. 87).

Descrição da Imagem: em um quadrado com o nome Dialog, há o texto: render(), ligado por uma seta pontilhada à interface Button (outro quadrado) e dois outros quadrados abaixo: um com o rótulo WindowsDialog e outro com o nome WebDialog, seguidos pelos textos: "create button". Também temos os respectivos objetos (quadrados) Windows Button e HTML Button. Fim da descrição.

No exemplo anterior, verificamos que, por meio da interface implementada, é definido o tipo do botão, como Web ou Windows, o qual, consequentemente, gera um diálogo diferente.



Padrão Abstract Factory

Cria vários objetos relacionados sem a necessidade de especificar as classes concretas. Você pode ter a necessidade de criar objetos diferentes com uma característica semelhante, por exemplo, mesa e cadeira do mesmo estilo. Em primeiro lugar, você cria uma interface fábrica, com as classes de cada estilo.

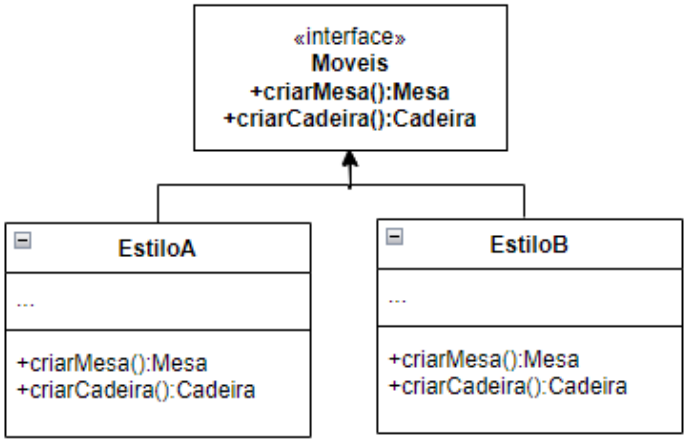


Figura 6 – Diagrama de classes Abstract Factory / Fonte: adaptada de Shvets (2021).

Descrição da Imagem: a figura apresenta um quadro com o texto "Interface" e nome "Móveis", o qual possui duas classes: EstiloA e EstiloB, logo abaixo, também representadas por quadrados. Cada uma delas, por sua vez, contém as expressões: criarMesa e criarCadeira, ambos ligados por uma seta à interface móveis. Fim da descrição.

Segundo os autores Freeman e Freeman (2007, p. 135), "uma fábrica abstrata nos dá uma interface para criar uma família de produtos. Ao escrever o código que usa essa interface, desvinculamos nosso código da fábrica real que cria os produtos". Com isso, você garante que exista a criação de mesa nos dois estilos e a composição de cadeira nos dois estilos. Para cada variante dos produtos, criamos uma classe factory.

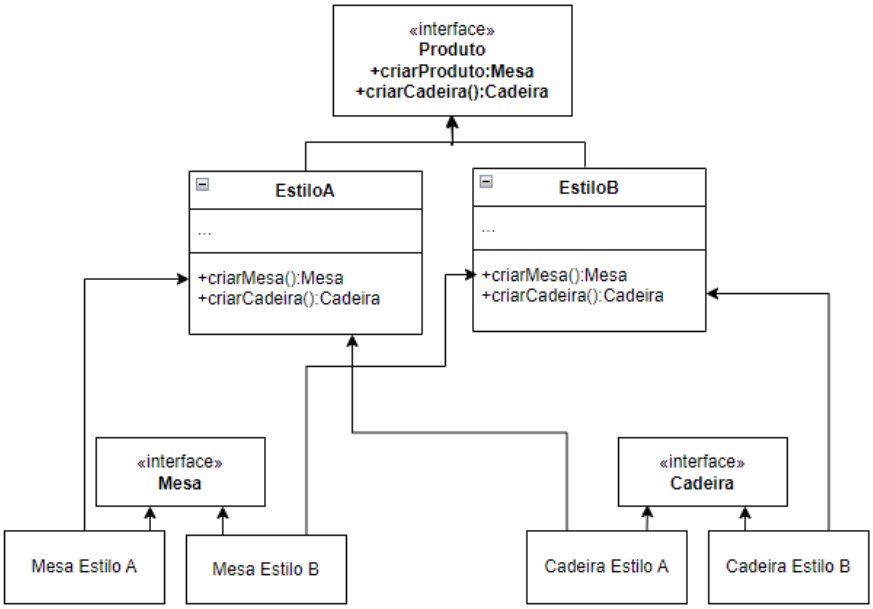


Figura 7 - Exemplo do padrão Abstract Factory / Fonte: adaptada de Shvets (2021).

Descrição da Imagem: em um quadro, há um texto com o nome da interface "Produto", com os textos: "criarProduto" e "criarCadeira". Abaixo, outros dois quadros, que são as classes: "EstiloA" e "EstiloB", também com os textos: "criarMesa" e "criarCadeira". Essa interface possui dois métodos, representados pelos textos: "criarMesa" e "criarCadeira", ambos implementados pelas classes mencionadas. Abaixo do quadrado, encontra-se a representação de duas cadeiras, representadas pelos textos "Cadeira Estilo A" e "Cadeira Estilo B", respectivamente, com linhas contínuas, indicando sua estrutura. À esquerda, estão dois quadrados, representando os objetos "Mesa Estilo A" e "Mesa Estilo B", respectivamente, também representada por linhas contínuas. Fim da descrição.

> O código cliente tem que funcionar com ambas as fábricas e produtos via suas respectivas interfaces abstratas. Isso permite que você mude o tipo de uma fábrica que passou para o código cliente, bem como a variante do produto que o código cliente recebeu, sem quebrar o código cliente atual (Shvets, 2021, p. 101).

Padrão Builder

Permite produzir diferentes tipos de um objeto, usando o mesmo código. Em um código, em que você deve instanciar um objeto complexo, com muitos parâmetros, o construtor fica muito grande. Ao criar várias subclasses para instanciar os vários objetos, o código pode ficar muito complexo, pois qualquer novo parâmetro o forçará a aumentar essa hierarquia cada vez mais.

Você pode, também, ter uma classe diretor, a qual sabe de todos os passos que precisam ser executados para compor o objeto.

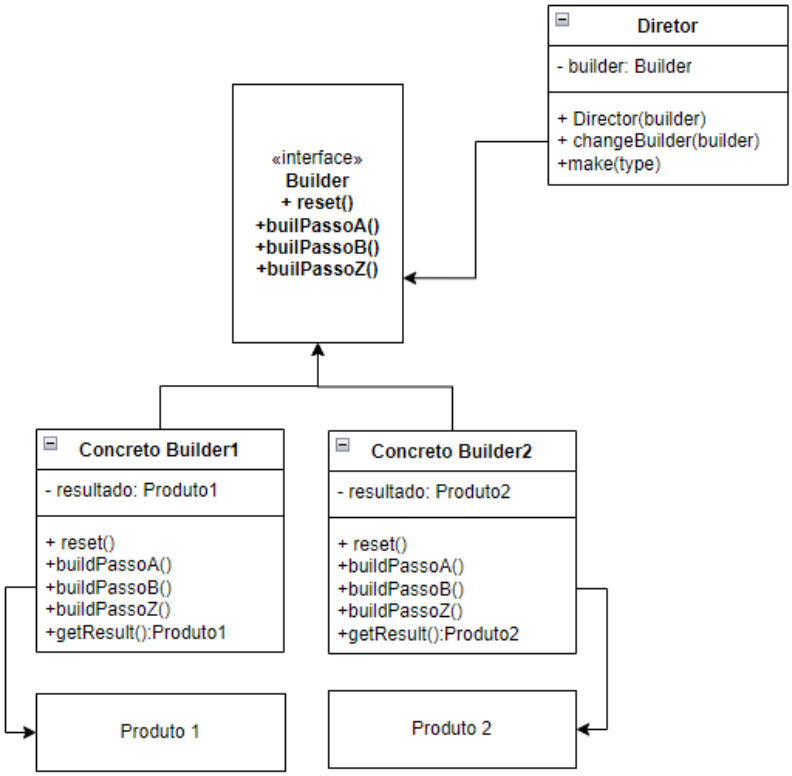


Figura 8 – Diagrama de classes do padrão Builder / Fonte: adaptada de Shvets (2021).

Descrição da Imagem: em um quadrado, há uma representação textual do diagrama de classes do padrão Builder com os textos Director, Change Builder e Make, que são os métodos. Na parte à esquerda, encontra-se outro quadrado com o texto interface Builder e os textos: reset, buildPassoA, buildPassoB, buildPassoZ. Abaixo, há dois quadrados com os textos Concreto Builder1 e Concreto Builder2, respectivamente, representando as diferentes classes de produtos, cada uma implementando a interface mencionada, ou seja, com os textos: reset, buildPassoA, buildPassoB, buildPassoZ e getResult do produto. Abaixo, outros quadrados com os textos: Produto 1 e Produto 2 representam os objetos criados. Fim da descrição.

Padrão Prototype

Com esse padrão, torna-se possível copiar objetos sem que sua codificação fique dependente deles. Fazer a cópia do objeto, por si, pode acarretar alguns problemas. Por exemplo, se você copiar, não verá métodos privados que a classe possa utilizar. Outro entrave seria a classe ficar com uma dependência da classe que irá copiar.

O padrão Prototype permite a criação do clone do objeto sem acoplar seu código à respectiva classe. Esse parâmetro delega a tarefa de clone ao próprio objeto a ser clonado.

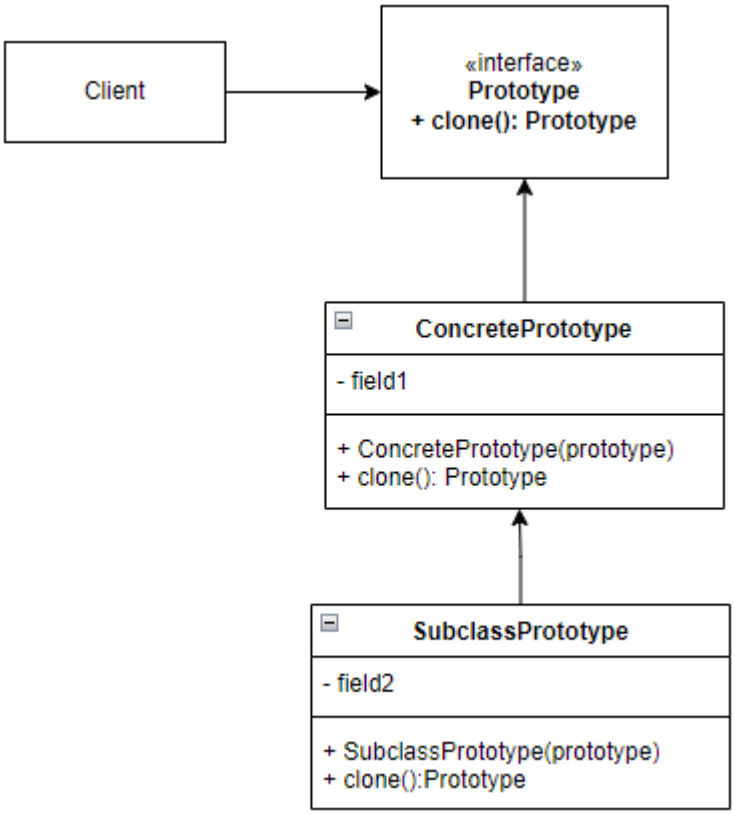


Figura 9 – Diagrama de classes do padrão Prototype / Fonte: adaptada de Shvets (2021).

Descrição da Imagem: na esquerda, temos um quadrado com a inscrição Client. Mais à direita, há um quadrado com a inscrição "Interface", sendo os dois interligados por uma linha contínua. Em outra linha, temos a palavra "Prototype" e na seguinte, "clone", indicando os métodos. Abaixo, há outro quadrado com o texto "ConcretePrototype", o texto "field1" e os métodos "ConcretePrototype" e "clone". Logo abaixo, temos outro quadrado com o nome "SubclassPrototype", indicando o atributo "field2" e os textos "SubclassPrototype" e "Clone". Fim da descrição.

INDICAÇÃO DE FILME

O Jogo da Imitação

O filme retrata a história de Alan Turing, um matemático e pioneiro da computação que desempenhou um papel fundamental na quebra do código Enigma, usado pelos nazistas durante a Segunda Guerra Mundial.

Refletindo sobre a história: ao longo do longa-metragem, você pode observar paralelos entre os desafios enfrentados por Turing e sua equipe na decodificação do Enigma e os conceitos de *design patterns* na engenharia de software. Eles precisavam identificar padrões nos códigos criptografados para desenvolver estratégias eficazes de decodificação, destacando a importância de reconhecer e aplicar padrões para resolver problemas complexos, seja na computação, seja em outros campos.

EM FOCO

Acesse seu ambiente virtual de aprendizagem e confira a aula referente a este tema. **Recursos de mídia disponíveis no conteúdo digital do ambiente virtual de aprendizagem.**

NOVOS DESAFIOS

Neste tema de aprendizagem, abordamos padrões de projeto de criação que são reutilizáveis para a solução dos problemas comuns no desenvolvimento de software. Verificamos, também, como os padrões Singleton, Factory, Abstract Factory, Builder e Prototype podem nos ajudar.

A aplicação desses padrões é essencial ao desenvolvimento de software. Para os estudantes em início de carreira na área de programação, um ponto que os destaca é o conhecimento de soluções para problemas conhecidos. Portanto, o desenvolvimento de padrões de criação é um diferencial no mercado de trabalho, uma vez que as empresas valorizam candidatos que apresentem habilidades sólidas em design de software.

Tenha sempre em mente, estudante, que a prática contínua faz parte do aperfeiçoamento. Isso não apenas lhe ajudará como desenvolvedor, mas também lhe auxiliará a avançar na carreira.

Como você abordaria o planejamento e desenvolvimento das soluções? Será que os padrões de criação poderiam contribuir com as atividades?

No início do tema, propusemos um questionamento acerca da forma com que você abordaria o planejamento e o desenvolvimento das soluções e desafios impostos, se de fato os padrões de criação poderiam contribuir com as atividades necessárias.

Podemos entender que todos os padrões são soluções que foram amplamente experimentadas e aplicadas, de forma que, entendendo o objetivo e tendo em mente os benefícios da aplicação, conseguimos aplicar esses temas em nossas soluções e desafios impostos pelo mercado, usufruindo os benefícios proporcionados pela aplicação de padrões, como flexibilidade e escalabilidade do sistema, mantendo uma estrutura bem fundada.

Durante o tema, evidenciamos vários padrões, que apresentam soluções válidas e documentadas para diversos problemas que se apresentam no dia a dia. Diante desses problemas e soluções apresentadas, podemos responder que o padrão de projetos é realmente muito importante, pois apresenta soluções para problemas do cotidiano.

1. "Efetivamente, os designs patterns podem contribuir para suavizar a transição do modelo de análise para o modelo de implementação, haja vista, balizam-se em tipos recorrentes de problemas de implementação em projetos de software. Assim, podendo ser vistos como um modelo de 'microarquitetura' aplicado a um dado problema, o que estabelece uma linguagem comum entre a construção de aplicações que permeiam um mesmo contexto" (Valentim; Souza Neto, 2005, p. 70).

   A respeito das divisões de padrões de projeto, é correto afirmar:

   a) Singleton é um padrão de criação.
   b) Factory Method é um padrão de estrutura.
   c) Decorator é um padrão de criação.
   d) State é um padrão de interface.
   e) Visitor é um padrão estrutural.

2. "Imagine que você criou um objeto, mas depois de um tempo, decidiu criar um novo. Ao invés de receber um objeto fresco, você obterá um que já foi criado" (Shvets, 2021, p. 151).

   Diante do exposto, analise as afirmativas a seguir:

   I - O padrão que se encaixa no cenário descrito é o Singleton.
   II - O padrão Singleton possui um construtor privado, assim, previne que outros objetos o instanciem com new.
   III - O governo é um excelente padrão de Singleton, pois um país pode ter apenas uma administração oficial.

   É correto o que se afirma em:

   a) I, apenas.
   b) III, apenas.
   c) I e II, apenas.
   d) II e III, apenas.
   e) I, II e III.

3. "Você está criando uma aplicação de gerenciamento de logística. A primeira versão da sua aplicação pode lidar apenas com o transporte de caminhões, portanto, a maior parte do seu código fica dentro da classe Caminhão. Depois de um tempo, sua aplicação se torna bastante popular. Todos os dias, você recebe dezenas de solicitações de empresas de transporte marítimo para incorporar a logística marítima na aplicação" (Shvets, 2021, p. 81).

   Diante do exposto, assinale a alternativa que representa o padrão que deve ser aplicado ao cenário descrito:

   a) Singleton.
   b) Factory Method.
   c) Adapter.
   d) Decorator.
   e) Template Method.

REFERÊNCIAS

ALBUQUERQUE, M.; ROJAS, A.; RIBEIRO, P. C. M. Utilizando *design patterns* GoF no apoio ao desenvolvimento de um framework Java. **Cadernos do IME**, Rio de Janeiro, v. 30, n. 1, p. 13-27, 2010.

FREEMAN, E.; FREEMAN, E. **Use a cabeça: padrões de projetos**. 2. ed. Rio de Janeiro: Alta Books, 2007.

SHVETS, A. **Mergulho nos padrões de projeto**. [*S. l.*: *s. n.*], 2021.

VALENTIM, R. A. de M.; SOUZA NETO, P. A. de. O impacto da utilização de design patterns nas métricas e estimativas de projetos de software: a utilização de padrões tem alguma influência nas estimativas? **Revista da FARN**, Natal, v. 4, n. 1, p. 63-74, 2005.

GABARITO

1. Alternativa A.

   A opção A está correta, pois o Singleton faz parte dos padrões de criação.
   As demais opções estão incorretas, pois: B) Factory Method é um padrão comportamental; C) Decorator é um padrão de extensão; D) State é um padrão de operação; E) Visitor é um padrão de extensão.

2. Alternativa E.

   Todas as afirmativas são corretas.
   A descrição corresponde ao padrão Singleton, que garante que uma classe tenha apenas uma instância e forneça um ponto global de acesso a essa instância (I). Uma característica do padrão Singleton é ter um construtor privado, impedindo a instanciação direta da classe por outros objetos e garantindo que a mesma instância seja sempre retornada (II). A comparação com o governo, como um exemplo de Singleton, é válida, já que, em muitos casos, como em um país, existe apenas uma instância de um governo oficial, que gerencia os assuntos da nação. Isso reflete a ideia do padrão Singleton, em que uma classe tem apenas uma instância (III).

3. Alternativa B.

   O padrão que se adequa ao cenário definido é o padrão Factory Method.
   As demais alternativas estão incorretas, pois: A) o Singleton é usado quando é necessário garantir que uma classe tenha apenas uma instância e fornecer um ponto global de acesso a essa instância, como em gerenciadores de configuração, *pools* de conexão e caches; C) o Adapter é empregado quando é necessário permitir que interfaces incompatíveis trabalhem juntas. Ele converte a interface de uma classe em outra que o cliente espera, permitindo que classes com interfaces diferentes possam colaborar entre si; D) o Decorator é usado quando é necessário adicionar funcionalidades a objetos de forma dinâmica e flexível, sem alterar sua estrutura básica. Esse padrão permite que novas funcionalidades sejam adicionadas a um objeto existente sem modificar sua estrutura, seguindo o princípio aberto/fechado; E) o Template Method é usado quando se deseja definir o esqueleto de um algoritmo em uma classe base, deixando alguns passos do algoritmo para serem implementados nas subclasses. Isso permite que diferentes subclasses possam redefinir partes do algoritmo sem alterar sua estrutura global.

MINHAS ANOTAÇÕES

MINHAS ANOTAÇÕES

TEMA DE APRENDIZAGEM 3

PADRÕES DE ESTRUTURA

MINHAS METAS

- Apresentar uma visão prática do uso dos padrões na empresa.
- Conscientizar o aluno para a importância da simplicidade e flexibilidade do código.
- Introduzir os padrões estruturais.
- Definir padrões consolidados, como: Adapter, Facade, Decorator e outros modelos estruturais.
- Ilustrar, por meio de diagramas de classes, como o aluno identifica as conexões e chamadas de métodos.
- Citar abordagens práticas para o uso dos padrões de projeto.
- Despertar no aluno a curiosidade para a utilização dos padrões em grandes sistemas.

INICIE SUA JORNADA

Imagine-se como um **desenvolvedor** de software em uma grande empresa, ansioso por ajudar a criar um sistema inovador e que faça a diferença na vida das pessoas. Você, porém, se depara com um programa de código confuso, difícil de alterar e dar manutenção, como resolver esse desafio? Como transformar o código complexo em algo simples?

O mundo profissional está cada vez mais competitivo, sendo que a necessidade por resultados é crescente nas empresas e aumenta a pressão sobretudo nos desenvolvedores, que contribuem para a criação de soluções de sistemas nas organizações. Dessa forma, estudante, em sua trajetória do mundo do desenvolvimento e necessidade de sistemas robustos e flexíveis, a **estrutura** de uma aplicação ocupa lugar de destaque, pois, sem essa estrutura sólida, os projetos podem se tornar confusos, de difícil manutenção ou ineficientes. Esse lugar de destaque e crescente busca pelos *design patterns* pode ser verificada na internet. Sendo assim, pesquise e elabore uma listagem de cinco livros que abordam o design de projetos e seus respectivos autores.

Os padrões de design de estrutura nos apresentam respostas a essa questão. Eles nos mostram soluções para problemas comuns, que são encontradas no dia a dia e comprovadas para entraves em sistemas de software, ampliando nossos horizontes e ajudando-nos a ter uma visão mais sistêmica do programa.

A compreensão dos padrões de design vem da **experimentação** prática, por meio da qual você poderá, de fato, vivenciar os cenários que se apresentam. Portanto, à medida que você, estudante, experimenta essas situações e casos de uso, será possível reconhecer os problemas e saber como aplicar tais padrões nos seus projetos.

Os padrões de projeto, apesar de serem soluções padronizadas para algumas situações, não se tratam de uma aplicação mecânica. Você, estudante, será capaz de avaliar as situações de aplicação e as razões para a aplicabilidade do processo, sendo que a reflexão sobre essas situações é fundamental. Qual a importância do padrão de projetos no desenvolvimento? Ou ainda, como os padrões de estrutura podem ser adequar ao meu projeto?

PLAY NO CONHECIMENTO

Ouça o podcast acerca do tema Integrando Design Patterns à Metodologia Ágil, assunto atual e que se aplica ao desenvolvimento de softwares. Acesse e atualize seus conhecimentos. **Recursos de mídia disponíveis no conteúdo digital do ambiente virtual de aprendizagem.**

VAMOS RECORDAR?

As situações que enfrentamos no dia a dia envolvem resolução de problemas, as quais devem ser trabalhadas de forma que apresentem implementações sólidas e estruturadas. Acerca dos padrões, sobretudo o Strategy, e de sua codificação, assista ao vídeo sugerido.

Disponível em: **https://www.youtube.com/watch?v=WPdrnuSHAQs**.

DESENVOLVA SEU POTENCIAL

A padronização é fundamental ao desenvolvimento de software. A estruturação e simplicidade, na criação, é fundamental para garantir um programa mais robusto e escalável. Isso faz dos padrões de estrutura uma categoria crucial para auxiliar a composição de classes e objetos, de forma que estruturas mais complexas sejam formadas, simplificando a criação e ajudando a garantir que os sistemas sejam mais flexíveis:

> [...] efetivamente, os designs patterns podem contribuir para suavizar a transição do modelo de análise para o modelo de implementação, haja vista, balizam-se em tipos recorrentes de problemas de implementação em projetos de software. Assim, podendo ser vistos como um modelo de 'microarquitetura' aplicado a um dado problema, o que estabelece uma linguagem comum entre a construção de aplicações que permeiam um mesmo contexto (Valentim; Souza Neto, 2005, p. 70).

ADAPTER PATTERN

Esse padrão de projeto permite que objetos com interfaces incompatíveis trabalhem juntos, convertendo classes e fazendo com que elas trabalhem entre si.

EU INDICO

Este vídeo ressalta o padrão Adapter e seu uso explicado com a codificação. Disponível em: **https://www.youtube.com/watch?v=tNbaQpg0k_E**.

Devemos considerar duas abordagens principais para implementar o Adapter: adaptação por herança e por composição.

Adaptação por herança

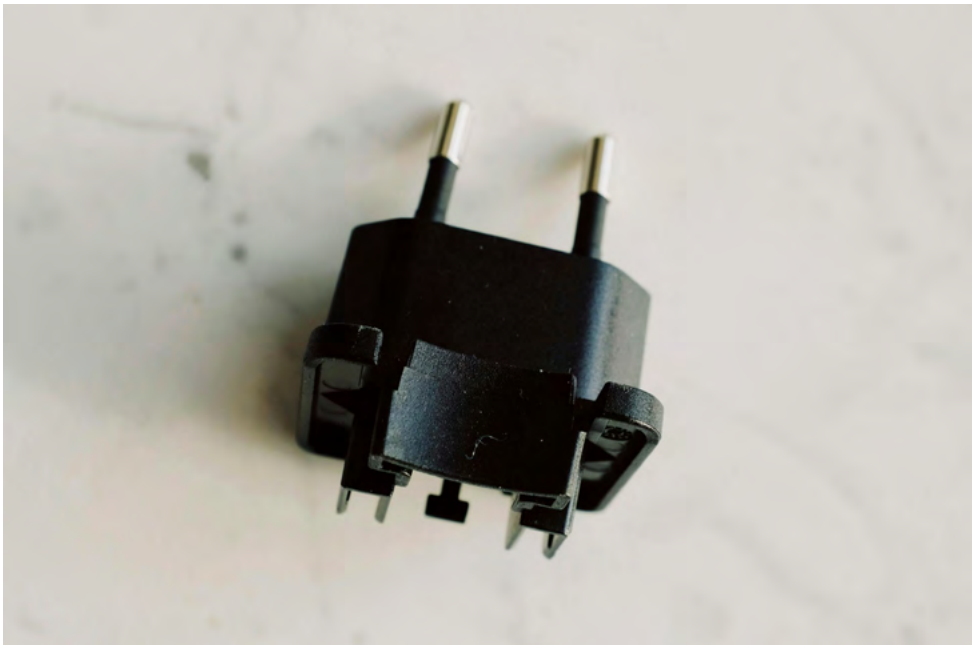
A classe responsável por fazer a adaptação herda a interface da classe que precisa ser adaptada. Essa classe, então, implementa os métodos necessários para suprir o esperado pela interface, adaptando as chamadas de forma transparente.

Para a classe Adapter conseguir adaptar as comunicações entre os objetos, ela herda interfaces dos dois padrões a serem utilizados. Como no exemplo da tomada, ela herda uma interface da tomada alemã e outra da brasileira, mantendo características dos dois padrões.

ADAPTAÇÃO POR COMPOSIÇÃO

A classe que pode ser chamada de adaptadora mantém uma instância da que precisa ser adaptada, delegando a chamada dos métodos para a solicitação criada, quando não desejamos herdá-la da classe que deve ser adaptada, como a abordagem anteriormente mencionada.

A seguir, podemos identificar a representação das classes na adaptação por composição.



BRIDGE PATTERN

O padrão Bridge atua, separando uma abstração da implementação. Segundo Shvets (2021, p. 177),

> [...] [ele] permite que você divida uma classe grande ou um conjunto de classes intimamente ligadas em duas hierarquias separadas – abstração e implementação – que podem ser desenvolvidas independentemente umas das outras.

PENSANDO JUNTOS

Imagine que você tenha uma classe chamada forma, a qual possui duas subclasses: uma chamada círculo e outra, quadrado. Cada uma delas tem duas classes, por exemplo, círculo azul, círculo vermelho, quadrado azul, quadrado vermelho. Em outras palavras, se criarmos mais uma classe de tipo forma, teremos mais duas subclasses. Assim, a criação da hierarquia segue uma progressão geométrica. Compreendeu o problema enfrentado?

> O padrão Bridge tenta resolver esse problema ao trocar de herança para composição do objeto. Isso significa que você extrai uma das dimensões em uma hierarquia de classe separada, para que as classes originais referenciem um objeto da nova hierarquia, ao invés de ter todos os seus estados e comportamentos dentro de uma classe (Shvets, 2021, p. 179).

Como exemplo, podemos citar uma API que pode ter várias versões: uma para Windows; uma para Linux, MacOS e várias GUIs; uma para clientes; e uma para administradores. Uma nova GUI ou sistema operacional fará a hierarquia crescer exponencialmente.

EU INDICO

Este vídeo ressalta o padrão Bridge e seu uso explicado com a codificação. Disponível em: **https://www.youtube.com/watch?v=rCyg-8eiLZI**.

O padrão Bridge trabalha, dividindo as classes em duas hierarquias:

ABSTRAÇÃO
Camada GUI
IMPLEMENTAÇÃO
API do sistema operacional

COMPOSITE PATTERN

> O Composite é um padrão de projeto estrutural que permite que você componha objetos em estruturas de árvores e, então, trabalhe com essas estruturas como se elas fossem objetos individuais. Por exemplo, imagine que você tem dois tipos de objetos: produtos e caixas. Uma Caixa pode conter diversos Produtos, bem como um número de Caixas menores. Essas Caixas menores também podem ter alguns Produtos ou até mesmo Caixas menores que elas, e assim por diante (Shvets, 2021, p. 195).

O padrão Composite sugere trabalhar com a ideia de produtos e caixas por meio de uma interface comum. De acordo com Freeman e Freeman (2007, p. 291), ele "permite que você componha objetos em estruturas de árvore para representar hierarquias parte-todo. Com esse padrão, os clientes podem tratar objetos individuais ou composições de objetos de maneira uniforme".

Um exemplo de aplicação poderiam ser menus e submenus, os quais compõem uma árvore. A Figura 1 ilustra o diagrama de classes do padrão.

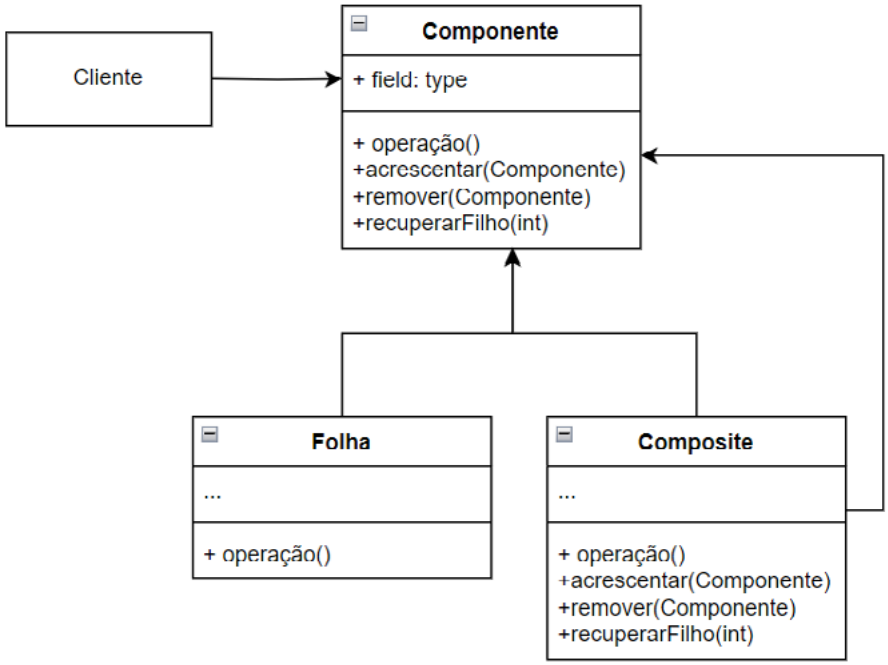


Figura 1 – Diagrama de classes do padrão Composite / Fonte: Freeman e Freeman (2007, p. 293).

Descrição da Imagem: na parte superior esquerda, há um quadrado, representando o objeto Cliente. Logo à direita, há outro quadrado com o nome Componente, dividido por duas linhas horizontais, que representam as divisões para o nome da classe "Componente", atributos "field" e métodos da classe, representados pelos textos: "operação", "acrescentar", "remover" e "recuperarFilho". Abaixo desse segundo quadrado, há dois outros, um ao lado do outro, ambos com linhas contínuas ao redor, indicando a subclasse "Folha" e "Composite", respectivamente. A "Folha" contém o texto "operação" e a classe "Composite" contém os textos: "operação", "acrescentar", "remover" e "recuperarFilho". Fim da descrição.

DECORATOR PATTERN

Segundo Shvets (2021, p. 208), "o Decorator é um padrão de projeto estrutural que permite que você acople novos comportamentos para objetos ao colocá-los dentro de invólucros de objetos que contém os comportamentos". Por exemplo, em um objeto notificação, é possível obter inúmeros tipos de notificação, assim, você precisaria herdar as classes de forma a atender a todas as chamadas de notificação.

A agregação é o princípio que aplicaremos no padrão Decorator, o qual será utilizado em vez da herança.

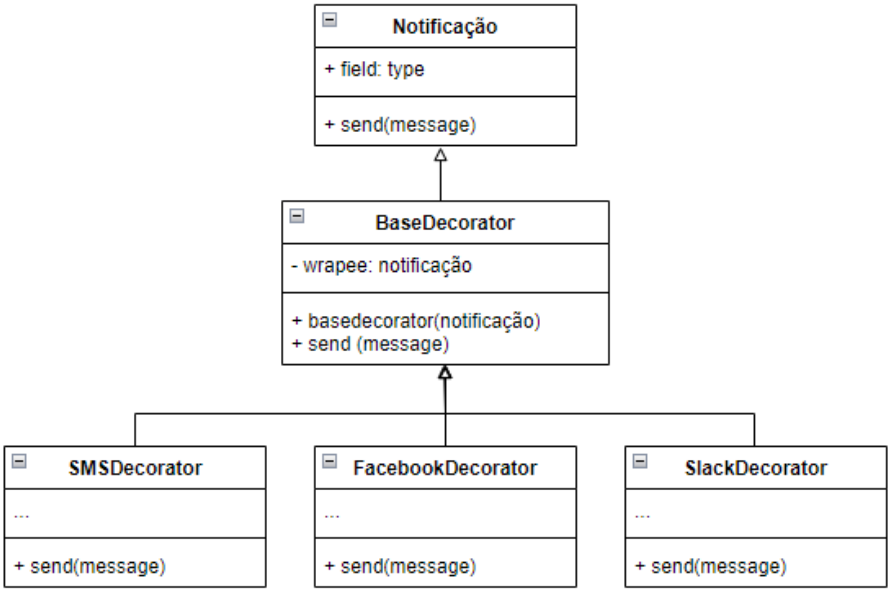


Figura 2 – Diagrama de classes do padrão Decorator / Fonte: Shvets (2021, p. 214).

Descrição da Imagem: no diagrama, há um grande quadrado que representa a "superclasse Notificação". Dentro dele, há um texto que diz "superclasse Notificação". Saindo desse quadrado, há uma linha contínua, conectando-o a um segundo quadrado, que representa a "classe agregada chamada BaseDecorator". Dentro desse segundo quadrado, há um texto que diz "classe agregada BaseDecorator". Além disso, partindo do segundo quadrado, há três linhas contínuas, conectando-o a três quadrados menores. O primeiro quadrado menor representa a classe para envio de mensagens SMS, o segundo representa a classe para envio de mensagens Facebook e o terceiro representa a classe para envio de mensagens Slack. Cada um desses quadrados menores contém um texto que indica qual tipo de mensagem é utilizado (SMS, Facebook, Slack). Fim da descrição.

Esse diagrama descreve a estrutura do padrão Decorator, em que a superclasse Notificação é a base para as diferentes formas de envio de mensagens, como SMS, Facebook e Slack, implementadas por meio da classe agregada base Decorator.

FACADE PATTERN

Segundo Shvets (2021, p. 228), "o Facade é um padrão de projeto estrutural que fornece uma interface simplificada para uma biblioteca, um framework, ou qualquer conjunto complexo de classes". Ele fornece interface simples para um sistema em que muitas classes interagem.

EU INDICO

Este vídeo o padrão Bridge seu uso explicado com a codificação.
Disponível em: **https://www.youtube.com/watch?v=5bltlFCGIts**.

FLYWEIGHT PATTERN

Esse padrão visa minimizar o uso de recursos do computador, como memória, prezando pela otimização, por exemplo, quando uma aplicação precisa compartilhar informações com várias classes semelhantes, consumindo, dessa forma, muitos recursos computacionais.

> A maioria das implementações de editores de documentos possuem formatação e edição de texto, instalações que são modularizadas até certo ponto. Editores de documentos orientados a objetos normalmente usam objetos para representar elementos incorporados, como tabelas e figuras. No entanto, eles geralmente não usam um objeto para cada personagem no documento, embora isso promovesse flexibilidade nos melhores níveis da aplicação. Caracteres e elementos incorporados poderiam, então, ser tratados uniformemente em relação à forma como são desenhados e formatados. A aplicação poderia ser estendida para suportar novos conjuntos de caracteres sem perturbar outras funcionalidades. A estrutura do objeto do aplicativo pode imitar a estrutura física do documento (Gamma *et al.*, 1995, p. 218, tradução nossa).

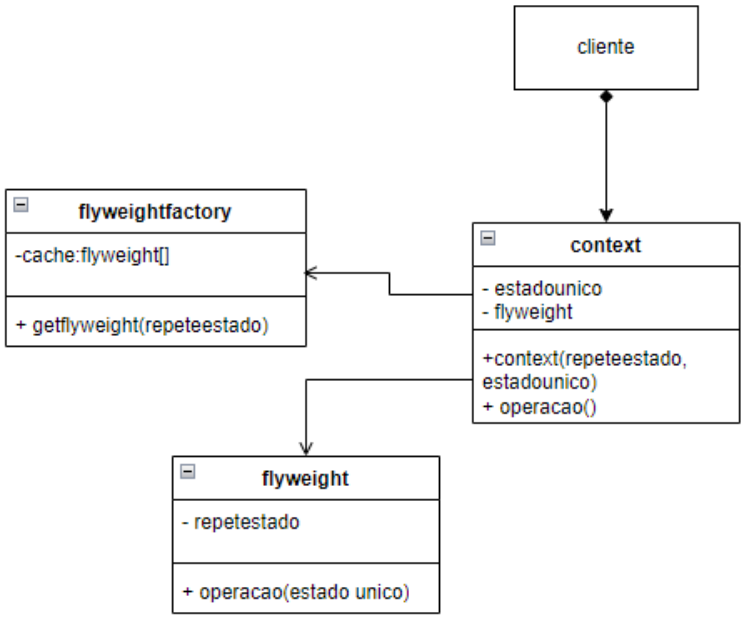


Figura 3 - Diagrama de classes do padrão Flyweight / Fonte: Shvets (2021, p. 246).

Descrição da Imagem: no centro da representação visual, há um quadrado grande que representa o objeto "cliente". Abaixo do objeto "cliente", há outro quadrado que representa a classe agregada "contexto". Dentro desse segundo quadrado, há um texto que diz "context". Partindo da classe agregada "contexto", há uma linha contínua, conectando-a a um terceiro quadrado, que representa a classe "flyweight". Dentro desse terceiro quadrado, há um texto que diz "flyweight". Além disso, dentro da classe "flyweight", há dois textos: um que diz "operacao" e outro que diz "estado único". Ao lado da classe "flyweight", há outro quadrado que representa a classe "flyweight-factory". Dentro desse quadrado, há um texto que diz classe "flyweightfactory". Além disso, dentro da classe "flyweightfactory", há um texto que diz "getfliweight". Fim da descrição.

Esse conjunto de elementos descreve a estrutura de um padrão "flyweight", em que um objeto "cliente" interage com uma classe agregada "contexto", que, por sua vez, utiliza as classes "flyweight" para compartilhar recursos e uma classe "flyweightfactory" para gerenciar esses objetos "flyweight" e sua reutilização.

PROXY PATTERN

Outro modelo de design estrutural conhecido é o Proxy. Ele permite a substituição de um objeto por outro. Esse substituto regula o acesso ao objeto original,

permitindo a execução de ações antes ou depois que uma solicitação seja encaminhada ao objeto original.

Podemos comparar um cartão de crédito, por exemplo, a um intermediário para uma conta bancária, que, por sua vez, representa uma reserva financeira. Ambos aderem à mesma funcionalidade, evitando a necessidade de transportar grandes somas em dinheiro físico. Isso traz comodidade ao cliente, que não precisa lidar com os riscos associados ao transporte de dinheiro. Além disso, o comerciante também se beneficia, já que a receita da transação é adicionada eletronicamente à sua conta, eliminando preocupações, como perdas ou roubos, durante o transporte ao banco.

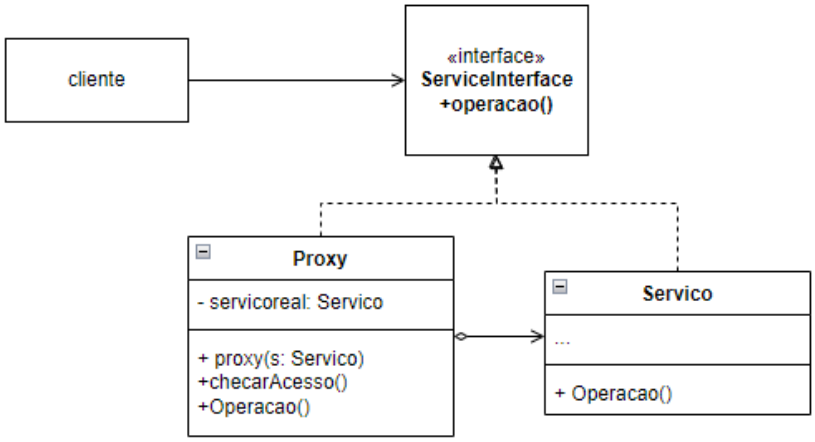


Figura 4 - Diagrama de classes do padrão Proxy / Fonte: Shvets (2021, p. 258).

Descrição da Imagem: no centro da representação visual, há um quadrado que representa o objeto "Cliente". Dentro desse quadrado, há um texto que diz "Cliente". Saindo desse quadrado, há uma linha contínua, conectando-o a outro quadrado (à direita), que representa a interface "ServiceInterface". Dentro desse segundo quadrado, há um texto que diz: "interface, ServiceInterface+operação()". Uma linha pontilhada conecta-o a um terceiro quadrado, que representa a subclasse "Proxy". Ele contém os termos: "proxy", "checarAcesso" e "Operacao". Por fim, ao lado direito da classe "Proxy", há um quadrado que representa "Serviço". Estas duas últimas classes citadas são interligadas por uma seta contínua, com um losango na origem. Fim da descrição.

Esse conjunto de elementos descreve um diagrama de classes em que o objeto "Cliente" implementa a interface "ServiceInterface" e possui uma subclasse "Proxy", que contém uma classe serviço agregada.

EM FOCO

Acesse seu ambiente virtual de aprendizagem e confira a aula referente a este tema. **Recursos de mídia disponíveis no conteúdo digital do ambiente virtual de aprendizagem.**

NOVOS DESAFIOS

Ao se desenvolver um software para uma grande empresa, você se deparará com inúmeras situações de códigos frágeis, extremamente acoplados e que necessitem refatoração.

Podemos citar, por exemplo, um sistema de gerenciamento de conteúdo, em que você criará várias páginas, com artigos e imagens, os quais, por sua vez, podem conter outros elementos. Nessa abordagem, você poderia aplicar o padrão Composite.

Nesse momento, você deverá considerar os padrões de projeto que irão definir, com maior robustez e flexibilidade, seu software. Por exemplo, se você se deparar com um software que apresenta muitas classes com responsabilidades sobrepostas ou mal definidas, poderá aplicar o padrão Facade para simplificar a interface, ou se o código estiver consumindo muitos recursos computacionais devido ao excesso de objetos em memória, poderá usar o padrão Flyweight.

Neste tema de aprendizagem, relacionamos vários padrões que poderão lhe ajudar a simplificar o seu código-fonte de uma forma concreta, sendo que, com todo o conteúdo visto, conseguimos chegar a uma resposta a respeito da pergunta feita inicialmente que é o desafio de um código complexo e ajudar a torná-lo simples, ou seja, com o auxílio dos padrões de projeto de estrutura, poderemos simplificar nossas atividades diárias, tornando os sistemas mais robustos.

1. "O Flyweight é um padrão de projeto estrutural que permite a você colocar mais objetos na quantidade de RAM disponível ao compartilhar partes comuns de estado entre os múltiplos objetos ao invés de manter todos os dados em cada objeto" (Shvets, 2021, p. 239).

   Com base nas informações apresentadas, avalie as asserções, a seguir, e a relação proposta entre elas:

   I - O padrão Flyweight é utilizado para reduzir a utilização de memória, compartilhando eficientemente objetos que possuem estados idênticos entre várias instâncias.

   PORQUE

   II - O padrão Flyweight extrai o estado compartilhado dos objetos e o armazena externamente, permitindo que várias instâncias compartilhem o mesmo estado, reduzindo, assim, a necessidade de armazenamento de múltiplas cópias desse estado.

   A respeito dessas asserções, assinale a opção correta:

   a) As asserções I e II são verdadeiras, e a II é uma justificativa correta da I.
   b) As asserções I e II são verdadeiras, mas a II não é uma justificativa correta da I.
   c) A asserção I é uma proposição verdadeira, e a II é uma proposição falsa.
   d) A asserção I é uma proposição falsa, e a II é uma proposição verdadeira.
   e) As asserções I e II são falsas.

2. "O Bridge é um padrão de projeto estrutural que permite que você divida uma classe grande ou um conjunto de classes, intimamente ligadas, em duas hierarquias separadas – abstração e implementação – que podem ser desenvolvidas independentemente umas das outras" (Shvets, 2021, p. 177).

   A respeito do padrão Bridge, analise as afirmativas a seguir:

   I - Desacoplar uma abstração de sua implementação para que as duas possam variar independentemente é uma intenção do padrão Bridge.
   II - Termos, como abstração e implementação, fazem parte da definição do padrão Bridge.
   III - A abstração é a camada de controle de alto nível, a qual não faz nenhum trabalho por conta própria e deve delegar para a camada de implementação.
   IV - No padrão Bridge, a abstração contém uma referência para uma implementação, de modo que pode delegar parte de seu comportamento para a implementação.

É correto o que se afirma em:

a) I, apenas.
b) II e IV, apenas.
c) III e IV, apenas.
d) I, II e III, apenas.
e) I, II, III e IV.

3. "Um padrão adaptador faz uma interface estar conforme com outra, fornecendo, assim, uma abstração de diferentes interfaces; uma classe faz isso, herdando privadamente de uma classe adaptada" (Gamma, 1995, p. 155, tradução nossa).

A respeito do padrão adaptador, analise as afirmativas a seguir:

I - É um padrão de projeto estrutural que permite que objetos com interfaces incompatíveis colaborarem entre si.
II - Ele é útil quando você deseja usar uma classe existente, mas sua interface não atende às necessidades do seu aplicativo, e não é possível ou desejável modificar a classe existente diretamente.
III - É um padrão de projeto criacional, que permite copiar objetos existentes sem fazer seu código ficar dependente de suas classes.
IV - O Adapter atua como uma camada intermediária entre o cliente e a classe adaptada, traduzindo solicitações do cliente para chamadas compatíveis com a classe adaptada e vice-versa.

É correto o que se afirma em:

a) I, apenas.
b) II e IV, apenas.
c) III e IV, apenas.
d) I, II e IV, apenas.
e) I, II, III e IV.

REFERÊNCIAS

FREEMAN, E.; FREEMAN, E. **Use a cabeça: padrões de projetos**. 2. ed. Rio de Janeiro: Alta Books, 2007.

GAMMA, E. *et al*. **Design Patterns**: elements of reusable object-oriented software. Reading: Addison-Wesley, 1995.

SHVETS, A. **Mergulho nos padrões de projeto**. [*S. l.*: *s. n.*], 2021.

VALENTIM, R. A. de M.; SOUZA NETO, P. A. de. O impacto da utilização de *design patterns* nas métricas e estimativas de projetos de software: a utilização de padrões tem alguma influência nas estimativas? **Revista da FARN**, Natal, v. 4, n. 1, p. 63-74, 2005.

GABARITO

1. Alternativa A.

   A asserção I é o objetivo principal do padrão Flyweight. A asserção II explica como o padrão Flyweight permite a redução de memória ao compartilhar o estado do objeto entre várias instâncias. As duas afirmações são verdadeiras, e a segunda é justificativa da primeira.

2. Alternativa E.

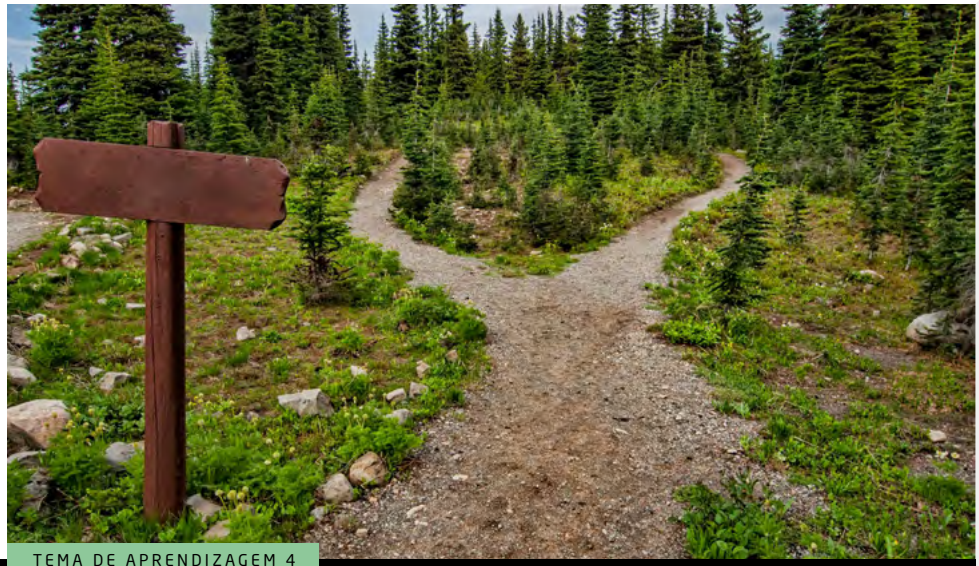
   Todas as afirmativas estão corretas e são características do padrão Bridge.

3. Alternativa D.

   A afirmativa III se refere ao padrão Prototype, as demais afirmativas são características do padrão Adapter.

MINHAS ANOTAÇÕES

Unicesumar



TEMA DE APRENDIZAGEM 4

PADRÕES DE COMPORTAMENTO

MINHAS METAS

- Identificar os desafios comuns, que se apresentam no desenvolvimento de softwares.
- Compreender o papel dos padrões de projeto, como guias para a solução dos problemas.
- Identificar a necessidade da aplicação dos padrões de projeto.
- Entender o que são padrões comportamentais e como atuam.
- Compreender os diferentes padrões comportamentais que se apresentam.
- Refletir acerca dos desafios enfrentados e da aplicação dos padrões de projeto, sobretudo os comportamentais.
- Estabelecer conexões entre teoria e prática na solução dos problemas, usando os padrões de projeto.

INICIE SUA JORNADA

Os **padrões de projeto**, como um todo, nos auxiliam a desenvolver um software, mantendo sua flexibilidade e melhorando sua manutenção. Esse aplicativo, por sua vez, possui vários grupos de padrões. Você saberia dizer qual a área de atuação dos padrões comportamentais?

Cada grupo dos padrões de projeto possui uma área de atuação, como os arquiteturais, que respondem pela organização geral da estrutura, de forma que os parâmetros comportamentais possuem um direcionamento.

Quando você, estudante, ingressa no mundo profissional, pode se sentir um pouco perdido diante da quantidade de informações acerca das formas distintas de se desenvolver um software, da maneira como as implementações ocorrem no mercado e dos modos de manutenção dos **algoritmos**.

Assim, torna-se fundamental refletirmos a respeito de cada solução implementada e como um padrão poderia ser utilizado para resolver o problema que se apresenta. Pergunte-se, então, acerca da aplicação desse padrão e de como ele contribui para essa solução.

Elabore uma pesquisa a respeito dos **padrões comportamentais**. Encontre três artigos acerca do assunto, relacionados às suas áreas de atuação dentro da informática. Em seguida, defina exemplos de aplicação dos principais padrões de projetos comportamentais. Eles são capazes de lhe auxiliar nas tarefas diárias?

PLAY NO CONHECIMENTO

Ouça o nosso podcast, acerca do tema A Influência dos Padrões de Projeto na Cultura DevOps, e atualize seus conhecimentos. **Recursos de mídia disponíveis no conteúdo digital do ambiente virtual de aprendizagem.**

VAMOS RECORDAR?

No vídeo sugerido, Macoratti fala brevemente a respeito dos padrões de projeto e dos padrões comportamentais.

Disponível em: **https://www.youtube.com/watch?v=1KdbOJ7EwKs**.

DESENVOLVA SEU POTENCIAL

Os padrões de projeto constituem uma ferramenta de apoio essencial para a construção de softwares robustos e flexíveis. Dentro desse cenário, os padrões de comportamento se destacam por definirem a interação entre objetos. Tais parâmetros não apenas promovem a interação, mas também a reutilização de código, contribuindo para a manutenção do sistema.

> Efetivamente, os design patterns podem contribuir para suavizar a transição do modelo de análise para o modelo de implementação, haja vista, balizam-se em tipos recorrentes de problemas de implementação em projetos de software. Assim, podendo ser vistos como um modelo de 'microarquitetura' aplicado a um dado problema, o que estabelece uma linguagem comum entre a construção de aplicações que permeiam um mesmo contexto (Valentim; Souza Neto, 2005, p. 70).

CHAIN OF RESPONSIBILITY PATTERN

Esse padrão permite que vários objetos recebam uma solicitação e possam lidar com elas. Ele é usado quando há múltiplos dispositivos para tratar uma chamada e o sistema não sabe qual deles lidará com ela. A estrutura é composta por três elementos:

ELEMENTO 1
Handler
ELEMENTO 2
ConcretHandler
ELEMENTO 3
Client

Quando existem várias chamadas, por exemplo, muitas validações em um sistema de autenticação, cada uma delas é transformada em objetos chamados *Handlers*. O padrão nos diz para ligar esses *handlers* em uma corrente, cada um deles referencia o próximo, formando uma corrente. O pedido de autenticação viaja por meio da corrente para que todas as validações sejam feitas e a pessoa, autenticada.

COMMAND PATTERN

Esse padrão comportamental encapsula uma solicitação como um objeto. Por exemplo, se você tem uma barra de botões, os quais são responsáveis pela execução de uma operação cada, poderia criar subclasses. Essa solução, porém, não é escalável e não é independente.

> O padrão Command sugere que os objetos GUI (como botões) não enviem esses pedidos diretamente à classe de negócios. Ao invés disso, você deve extrair todos os detalhes do pedido, tais como o objeto a ser chamado, o nome do método e a lista de argumentos em uma classe comando separada, que tem apenas um método que aciona esse pedido (Shvets, 2021, p. 296).

INTERPRETER PATTERN

Usado para interpretar uma expressão ou gramática definida em uma linguagem específica, esse padrão lhe permite definir uma representação gramática e implementar um interpretador. Esse parâmetro define uma estrutura de árvore em que o interpretador verifica as estruturas gramaticais, percorrendo cada nó da árvore.

EU INDICO

Neste vídeo acerca do padrão de projetos Interpreter, Macoratti explica o parâmetro, suas regras e aplicações.
Disponível em: **https://www.youtube.com/watch?v=1G5N0nfrH4g**.

ITERATOR PATTERN

Permite acessar os elementos de uma coleção de objetos de forma sequencial, ou seja, o padrão Iterator acessa os objetos, percorrendo cada item. Ele é composto por dois elementos principais: Iterator e Collection. O Iterator é a interface que define o posicionamento e movimentação na Collection. Geralmente, o Iterator possui os métodos next(), current(), hasnext().

MEDIATOR PATTERN

Promove a comunicação entre dois objetos, encapsulando a forma como eles interagem e reduzindo o acoplamento.

> O padrão Mediator sugere que você deveria cessar toda comunicação direta entre componentes que quer tornar independentes um do outro. Ao invés disso, esses componentes devem colaborar indiretamente, chamando um objeto mediador especial que redireciona as chamadas para os componentes apropriados. Como resultado, os componentes dependem apenas de uma única classe mediadora, ao invés de serem acoplados a dúzias de outros colegas (Shvets, 2021, p. 330).

Cada classe, que representa um componente, possui uma propriedade do tipo Mediator. Tal classe comunica-se com a classe mediador, chamando o método notify(). O Mediator possui uma classe relacionada chamada MediadorConcreto, que mantém a referência de todas as demais.

MEMENTO PATTERN

Esse padrão permite capturar e restaurar o estado de um objeto criado, pois:

> [...] algumas vezes, é necessário armazenar o estado interno de um objeto, por exemplo, em pontos de checagem ou mecanismo de desfazer, mas objetos normalmente encapsulam o estado, tornando inacessível para outros objetos externamente (Gamma *et al.*, 1995, p. 316).

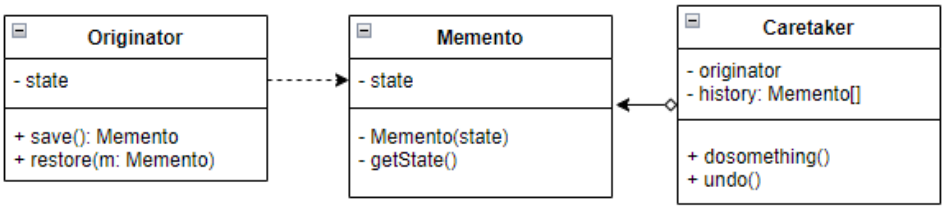


Figura 1 – Diagrama de classes do padrão Memento / Fonte: Shvets (2021, p. 351).

Descrição da Imagem: quadrado com o nome "Originator", representando a classe com os textos "state", "save" e "restore", correspondendo aos seus métodos. Logo à direita, outra classe é representada pelo texto "Memento" e os termos "Memento", correspondendo ao o método, e "getState". Ao lado dela, a classe chamada "Caretaker", com os textos "originator" e "history", indicando se tratar de um atributo do tipo "Memento", com os textos "dosomething" e "undo". Fim da descrição.

OBSERVER PATTERN

Define uma relação de um-para-muitos entre objetos, de modo que, quando um objeto muda de estado, todos os relacionados são notificados e atualizados automaticamente.

> O padrão Observer sugere que você adicione um mecanismo de assinatura, para a classe publicadora, para que objetos individuais possam assinar ou desassinar uma corrente de eventos vindos daquela publicadora. Nada tema! Nada é complicado como parece. Na verdade, esse mecanismo consiste em um vetor para armazenar uma lista de referências aos objetos do assinante e alguns métodos públicos que permitem adicionar assinantes e removê-los da lista (Shvets, 2021, p. 364).

STATE PATTERN

Esse padrão de projeto permite a um objeto alterar seu comportamento quando o estado interno é alterado. Seria como se o objeto mudasse de classe.

> O padrão State sugere que você crie novas classes para todos os estados possíveis de um objeto e extraia todos os comportamentos específicos de estados para dentro dessas classes.
>
> Ao invés de implementar todos os comportamentos por conta própria, o objeto original, chamado contexto, armazena uma referência para um dos objetos de estado que representa seu estado atual, e delega todo o trabalho relacionado aos estados para aquele objeto (Shvets, 2021, p. 382).

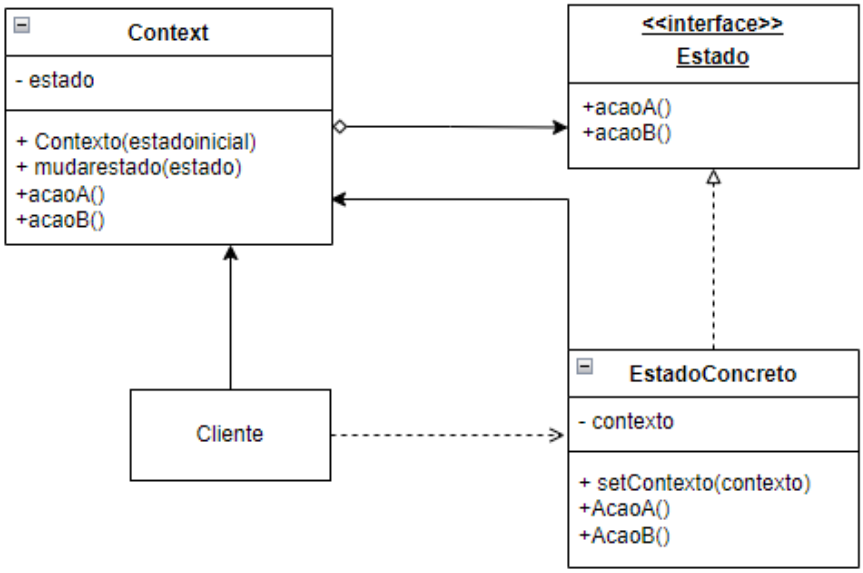


Figura 2 - Diagrama de classes do padrão State / Fonte: Shvets (2021, p. 384).

Descrição da Imagem: quadrado com o texto "Context", representando a classe com esse nome. Há um texto dentro do mesmo quadrado, representando o atributo "estado". Abaixo e dentro do mesmo quadrado, há os textos: "Contexto", com o termo, entre parênteses, "estadoinicial"; "mudar estado", com "estado" entre parênteses; e "acaoA" e "acaoB" em cada linha. Abaixo desse quadrado, há outra classe chamada "EstadoConcreto", com os textos: "contexto", "setContexto", "AcaoA e AcaoB. Mais para a esquerda, um quadrado pequeno representa o objeto "Cliente". Fim da descrição.

STRATEGY PATTERN

É um padrão que permite definir uma família de algoritmos, os quais são encapsulados em classes separadas, chamadas de estratégia, implementando uma mesma interface ou herdando de uma classe abstrata.

APROFUNDANDO

Utilizamos o Strategy Pattern quando o objeto cliente precisa executar algoritmos distintos de acordo com alguma situação, por exemplo, em um pagamento, por cartão de crédito ou carteira digital.

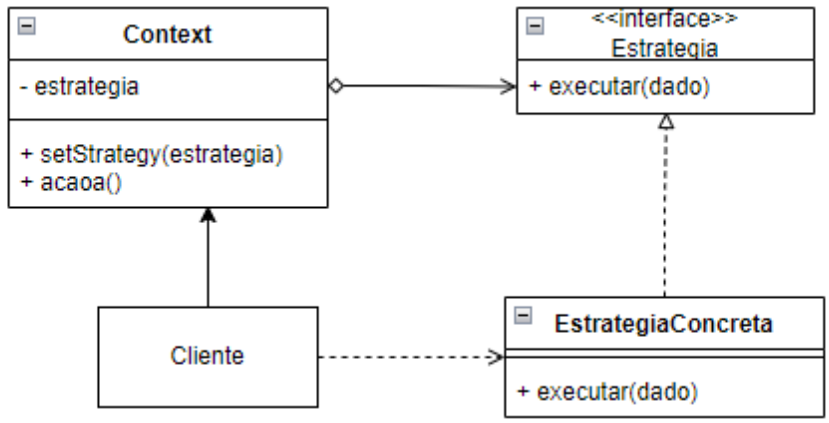


Figura 3 - Diagrama de classes do padrão Strategy / Fonte: Shvets (2021, p. 401).

Descrição da Imagem: quadrado com o texto "Context", representando a classe com esse nome. Um texto, dentro do mesmo quadrado, representa o atributo "estrategia". Abaixo dele, há as expressões: "setStrategy", com "estratégia" entre parênteses, e "acaoa". No lado direito, interligado por uma linha contínua e um diamante na origem, temos um outro quadrado, representando a interface "Estrategia", com o texto "executar" e a palavra "dado" entre parênteses. Abaixo, temos outro quadrado com o nome "Estrategia Concreta" e o texto "executar", com a palavra "dado" entre parênteses. Da classe "EstrategiaConcreta", parte uma linha pontilhada, que se liga à interface "Estrategia". À esquerda e abaixo do quadrado "Context", temos um quadrado menor com o texto "Cliente", representando o objeto "Cliente" e ligando-se por uma linha pontilhada à classe "EstrategiaConcreta". Fim da descrição.

Essa implementação traz mais flexibilidade ao seu código e permite um maior acoplamento, facilitando a manutenção e os testes.

TEMPLATE METHOD PATTERN

O Template Method é um dos parâmetros mais utilizados no universo de padrões de projeto, o qual se concentra na definição de um esqueleto de algoritmo de uma classe base, permitindo que suas subclasses forneçam implementações específicas.

> O padrão Template Method define o esqueleto de um algoritmo dentro de um método, transferindo alguns de seus passos para as subclasses. O Template Method permite que as subclasses redefinam certos passos de um algoritmo sem alterar a estrutura do próprio algoritmo (Freeman; Freeman, 2007, p. 238).

EU INDICO

Neste vídeo, Fabio Kon explica o padrão Template Method.
Disponível em: **https://www.youtube.com/watch?v=SpWtH03rBfE**.

Esse método é muito utilizado em *web frameworks*. Isso, porque existem etapas que seguem um padrão ao lidar com o HTTP, mas algumas partes podem variar, dependendo da necessidade.

VISITOR PATTERN

Esse padrão permite adicionar novas operações a uma estrutura de objetos, sem modificá-los, ou seja, colocando o novo comportamento em uma classe criada separadamente e de nome visitante, sendo que o objeto original é parametrizado.

A Figura 4, seguir, apresenta o diagrama do padrão para identificação da relação entre as classes.

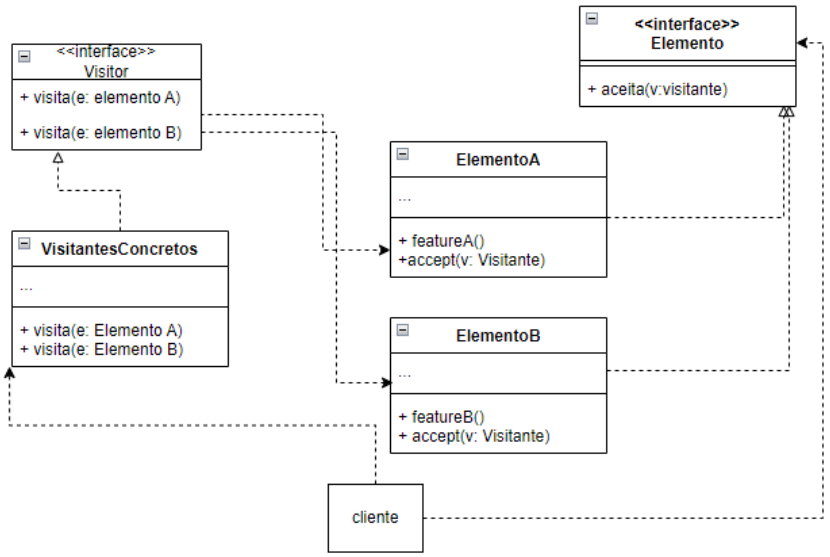


Figura 4 – Diagrama de classes do padrão Visitor / Fonte: Shvets (2021, p. 431).

Descrição da Imagem: um quadrado com o texto "Interface" e o texto "Visitor", representando a interface com esse nome. Dentro do mesmo quadrado, temos o texto "visita" e, entre parênteses, os termos "elemento A. Na linha seguinte, temos o texto "visita" novamente, com os termos, entre parênteses, "elemento B". No lado direito, interligado por duas linhas pontilhadas, temos dois quadrados: um com o texto "Elemento A" e outro com o texto "Elemento B", os quais representam o nome de cada classe. Um deles possui o texto "featureA" e o outro, "featureB". Nos dois quadrados, há também o termo "accept" com o texto "Visitante" entre parênteses. No canto superior direito, temos um outro quadrado com o texto "interface" e o termo "Elemento", indicando o nome da interface. Também, temos o texto "aceita", com os termos "visitante" entre parênteses. As linhas que saem de "Elemento A" e "Elemento B" são pontilhadas. Abaixo, à esquerda, temos um quadrado com o texto "Visitantes Concretos", com os textos "visita" e "elemento A" entre parênteses. Na linha seguinte, temos "visita" e "Elemento B" entre parênteses. Abaixo, ao centro, temos um quadrado menor com o texto "Cliente", representando o objeto cliente interligado com linhas pontilhadas a "VisitantesConcretos" e interface "Elemento". Fim da descrição.

Shvets (2021, p. 436) recomenda: "Utilize o Visitor quando você precisa fazer uma operação em todos os elementos de uma estrutura de objetos complexa, por exemplo, uma árvore de objetos".

EM FOCO

Acesse seu ambiente virtual de aprendizagem e confira a aula referente a este tema. **Recursos de mídia disponíveis no conteúdo digital do ambiente virtual de aprendizagem.**

NOVOS DESAFIOS

Como os padrões de projeto são soluções já aplicadas para problemas conhecidos, é fundamental o seu conhecimento e aplicação prática. Na teoria, você está aprendendo e consolidando seus conhecimentos nos *design patterns*.

No mercado de trabalho, esses padrões comportamentais são aplicados frequentemente, representando um papel crucial na criação e manutenção de sistemas, de forma que é fundamental seu entendimento e aplicação dos conceitos aqui tratados.

Quando você iniciou sua jornada neste tema, deparou-se com uma indagação acerca da área de atuação dos padrões comportamentais. A resposta para essa questão segue na linha de que os padrões comportamentais se concentram na interação entre os objetos. Em um sistema de software, eles definem como os objetos se comunicam e como eles colaboram para realizar uma tarefa ou responder a um evento em tempo de execução, subindo um pouco a granularidade e avaliando em um contexto mais amplo. Sob o aspecto da área de atuação, os padrões comportamentais se aplicariam a áreas como:

- **Arquitetura de software**: na interação entre os diversos módulos.
- **Desenvolvimento de software**: na definição da interação entre os componentes e garantia de flexibilidade.
- **Sistemas distribuídos**: na definição de como os diferentes nós do sistema interagem entre si.
- **Inteligência artificial e sistemas autônomos**: na definição de como os agentes inteligentes interagem entre si.

Na pesquisa solicitada a respeito dos padrões comportamentais e áreas de atuação, pode-se seguir a linha de que os padrões comportamentais de projeto se concentram na definição de como objetos interagem entre si e se comportam durante a execução do software. Esses padrões têm diversas áreas de atuação, incluindo a definição de comportamentos variáveis e a gestão de comunicação entre elementos. Por exemplo, o padrão Observer é frequentemente utilizado em sistemas de interface gráfica para notificar os objetos interessados acerca das mudanças de estado, enquanto o padrão Strategy é aplicado em contextos em que diferentes algoritmos podem ser intercambiáveis para realizar uma mesma tarefa, permitindo uma maior flexibilidade e extensibilidade do sistema. Já o padrão State é útil quando um objeto precisa alterar seu comportamento conforme muda o seu estado interno, proporcionando uma organização mais clara e manutenível do código.

AUTOATIVIDADE

1. "Os padrões comportamentais estão preocupados com algoritmos e a atribuição de responsabilidades entre objetos" (Gamma *et al*., 1995, p. 249, tradução nossa).

   A respeito dos padrões de projeto comportamentais, marque a alternativa correta:

   a) Descrevem a criação dos objetos.
   b) Descrevem a estrutura dos objetos.
   c) Descrevem os padrões de comunicação entre os objetos.
   d) Não permitem uma flexibilidade, pois o código fica acoplado.
   e) Não são escaláveis.

2. "Os padrões comportamentais estão preocupados com algoritmos e a atribuição de responsabilidades entre objetos. Padrões comportamentais descrevem não apenas padrões de objetos ou classes, mas também os padrões de comunicação entre eles. Esses padrões caracterizam um fluxo de controle complexo que é difícil de seguir em tempo de execução. Eles desviam seu foco do fluxo de controle para permitir que você se concentre apenas na maneira como os objetos estão interconectados" (Gamma *et al*., 1995, p. 249, tradução nossa).

   A respeito dos padrões comportamentais, analise as afirmativas a seguir:

   I - Eles descrevem como os objetos interagem e se comunicam entre si.
   II - Eles promovem a flexibilidade ao separar algoritmos das classes que os utilizam.
   III - Eles encapsulam comportamentos em objetos, permitindo que eles variem independentemente das classes dos clientes.

   É correto o que se afirma em:

   a) I, apenas.
   b) III, apenas.
   c) I e II, apenas.
   d) II e III, apenas.
   e) I, II e III.

3. "O padrão Observer define e mantém a dependência entre objetos. O exemplo clássico do Observer é Smalltalk Model/View/Controller Smalltalk, em que todas as visualizações do modelo são notificadas sempre que o seu estado muda" (Gamma *et al*., 1995, p. 249, tradução nossa).

   Assinale a alternativa correta a respeito do padrão Observer:

   a) Permite que você passe pedidos por uma corrente de *handlers*.
   b) Transforma o pedido em um objeto independente, que contém toda a informação sobre ele.
   c) Permite que você percorra elementos de uma coleção sem expor as suas representações estruturais (lista, pilha, árvore etc.).
   d) Permite que você defina um mecanismo de assinatura para notificar múltiplos objetos a respeito de quaisquer eventos que ocorram com o objeto que eles estão observando.
   e) Permite que um objeto altere seu comportamento quando mudar seu estado interno.

REFERÊNCIAS

FREEMAN, E.; FREEMAN, E. **Use a cabeça: padrões de projetos**. 2. ed. Rio de Janeiro: Alta Books, 2007.

GAMMA, E. *et al*. **Design Patterns**: elements of reusable object-oriented software. Reading: Addison-Wesley, 1995.

SHVETS, A. **Mergulho nos padrões de projeto**. [*S. l.*: *s. n.*], 2021.

VALENTIM, R. A. de M.; SOUZA NETO, P. A. de. O impacto da utilização de *design patterns* nas métricas e estimativas de projetos de software: a utilização de padrões tem alguma influência nas estimativas? **Revista da FARN**, Natal, v. 4, n. 1, p. 63-74, 2005.

GABARITO

1. Alternativa C.

   A alternativa A está incorreta, pois os padrões de projeto comportamentais não se concentram na criação dos objetos, mas sim no comportamento e interação entre eles durante a execução do software.
   A alternativa B está incorreta, pois os padrões de projeto comportamentais não descrevem a estrutura dos objetos, mas sim como eles interagem e se comportam.
   A alternativa C está correta, pois os padrões de projeto comportamentais também abordam os padrões de comunicação entre os objetos.
   A alternativa D está incorreta, pois os padrões de projeto comportamentais são projetados para aumentar a flexibilidade do código, permitindo que os objetos mudem de comportamento de forma dinâmica, sem necessidade de modificação no código cliente.
   A alternativa E está incorreta, pois a escalabilidade não é uma característica inerente aos padrões de projeto comportamentais.

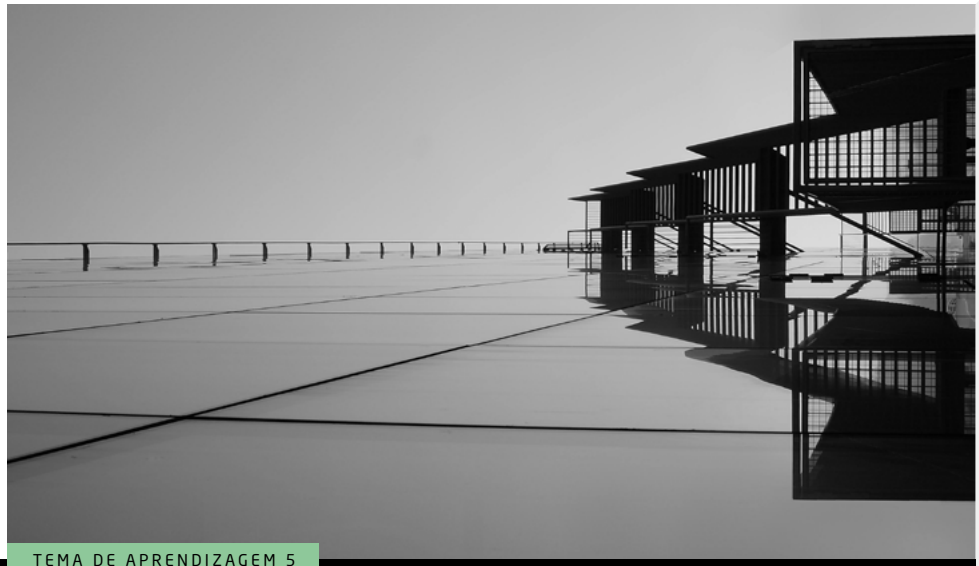
2. Alternativa E.

   A afirmativa I está correta, pois os padrões comportamentais, de fato, descrevem como os objetos interagem e se comunicam.
   A afirmativa II está correta, pois esses padrões promovem a flexibilidade ao separar algoritmos das classes que os utilizam.
   A afirmativa III está correta, pois os padrões comportamentais encapsulam comportamentos em objetos, permitindo que eles variem independentemente das classes dos clientes.

3. Alternativa D.

   A opção D está correta, pois se refere ao padrão Observer.
   As demais opções estão incorretas, pois referem-se ao padrão: Chain of Responsibility (A); Command (B); Iterator (C); State (E).

MINHAS ANOTAÇÕES



TEMA DE APRENDIZAGEM 5

PADRÕES DE ARQUITETURA

MINHAS METAS

- Aumentar o engajamento por padrões.
- Despertar o interesse pelos padrões de arquitetura no estudante.
- Refletir a respeito dos padrões e da carreira de desenvolvedor.
- Demonstrar a importância dos padrões de arquitetura.
- Apresentar os principais padrões e seu funcionamento.
- Compreender a melhor alternativa a ser aplicada.
- Incentivar o aprofundamento dos estudos e pesquisas relacionadas ao tema de arquitetura.

INICIE SUA JORNADA

Você já parou para pensar como os padrões de arquitetura são análogos ao esqueleto humano? Podemos fazer essa analogia, pois essa estrutura nos dá a sustentação comprovada para a solução de problemas encontrados no desenvolvimento.

Imagine-se, estudante, deparando-se com um grande problema que possa ser solucionado com uma **arquitetura bem definida**, em que você precise lidar com problemas de súbito crescimento do processamento e os serviços da arquitetura sejam independentes entre si. Como você garantiria que o sistema fosse flexível o suficiente para se adaptar a mudanças futuras? Qual arquitetura você utilizaria?

Os **padrões de arquitetura** nos fornecem saída para problemas como esse, assim como oferecem soluções testadas e comprovadas, as quais podemos aplicar em nosso dia a dia.

Com sua evolução no desenvolvimento de seu projeto, você pode aplicar e testar novas arquiteturas que se apliquem melhor a seu ambiente de desenvolvimento, desde modelos como a MVC (*Model-View-Controller*) até estruturas como os microsserviços.

Tendo essas questões a respeito dos padrões de arquitetura em mente, elabore uma **pesquisa** acerca de como podemos garantir que nosso código seja extensível e adaptável. Qual o padrão de arquitetura ajudaria a garantir a criação de objetos com flexibilidade? Devemos refletir a respeito desse problema comum em desenvolvimento. Isso torna-se possível, criando objetos sem especificar suas classes diretamente.

PLAY NO CONHECIMENTO

Ouça nosso podcast acerca do tema *Arquitetura de Software para Aplicações Mobile*. Acesse e atualize seus conhecimentos. **Recursos de mídia disponíveis no conteúdo digital do ambiente virtual de aprendizagem.**

VAMOS RECORDAR?

O vídeo do canal Eu TI Ensino fala brevemente acerca dos padrões de projeto arquiteturais. Acesse-o e entenda!

Disponível em: https://www.youtube.com/watch?v=fJafcfgloOw.

DESENVOLVA SEU POTENCIAL

Assim como os demais, os padrões de arquitetura são soluções comprovadas para problemas que se apresentam em nosso desenvolvimento. Eles ocupam um lugar fundamental na concepção de sistemas operacionais, pois auxiliam a identificação da forma de estruturação, assim como fornecem diretrizes para o desenvolvimento de softwares.

INDICAÇÃO DE FILME

Matrix (1999)

O filme conta a história de um futuro distópico em que as máquinas dominam os humanos, os quais vivem em uma realidade criada por computador. Os personagens vivem em um mundo simulado e, nesse ambiente, encontra-se o experiente programador Neo, que surge como uma espécie de salvador.

Refletindo sobre a história: no longa-metragem, os personagens, que vivem em um mundo simulado, identificam padrões no código e podem interagir com ele. Dessa forma, você pode traçar um paralelo entre os parâmetros identificados, os quais são necessários em projetos para que não ocorram anormalidades.

MVC PATTERN

Esse padrão é uma arquitetura de software que divide a aplicação em três camadas: *Model* (modelo), *View* (visualização) e *Controller* (controlador).

A camada *Model* é responsável pelos dados e regra de negócios da aplicação; a camada *View* é responsável por exibir ao usuário telas, mensagens, resultados; e a camada *Controller* faz a intermediação entre a *Model* e a *View*.

Para fazer a divisão em seu ambiente de software, você pode usar a divisão em pacotes ou *namespaces.* As camadas, porém, são agrupadas de acordo com a funcionalidade. Essa divisão clara de responsabilidades facilita a manutenção, flexibilizando a aplicação que está sendo desenvolvida.

> Imagine que você esteja usando o seu aparelho de MP3 favorito, como o Itunes. Você pode usar a interface para adicionar novas músicas, com o nome das trilhas. O aparelho cuida de uma pequena base de dados, contendo todas as suas músicas, com os nomes associados a elas e os respectivos dados. Além disso, ele cuida da reprodução das músicas e, enquanto faz isso, atualiza constantemente a sua interface de usuário para exibir o nome da música atual, o tempo de execução e outras informações. Bem, o que faz tudo isso funcionar é o modelo-visualização-controlador (Freeman; Freeman, 2007, p. 421).

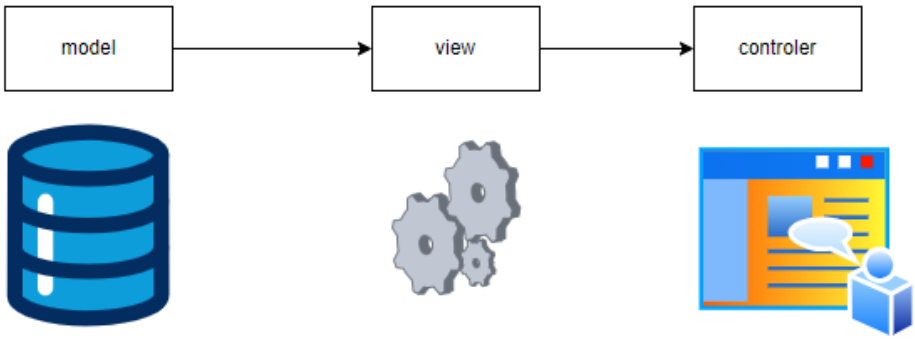


Figura 1 – Representação gráfica do MVC com separação das camadas / Fonte: o autor.

Descrição da Imagem: quadrado com nome Model, ligado por uma linha com um quadrado com o nome View, por sua vez, ligado por uma linha com o nome Controller. Abaixo de cada quadrado, uma ilustração representa cada objeto: um disco (um cilindro), algumas engrenagens e um página de internet, figuras que representam o Model, View e Controller respectivamente. Fim da descrição.

MVVM PATTERN

Esse modelo demonstra, também, uma divisão dos componentes, criada por meio das camadas *Model, View e ViewModel* (MVVM).

A camada *Model* é responsável por representar os dados e a lógica de negócios da aplicação, contendo classes que definem a estrutura dos dados e respectivas operações.

A camada de *View* diz respeito às telas, botões e demais interfaces gráficas, ou seja, a parte que interage com o usuário.

A camada *ViewModel* age como um intermediário entre o modelo e a visão. Ele inclui a lógica de apresentação, manipulação de dados e interação do usuário. O *ViewModel* prepara os dados do modelo para exibição na visão, assim como recebe entradas do usuário, atualizando o modelo conforme necessário. Ele geralmente implementa comandos e propriedades observáveis para facilitar a comunicação entre a visão e o modelo.

Uma diferença considerável entre o MVC e o MVVM é que o primeiro controla o fluxo de exibição, escolhendo qual controle será exibido de acordo com a lógica criada na *Model*, ao passo que o segundo passa o valor a ser exibido diretamente para a *user interface*, sem definir o que será exibido.

EVENT-DRIVEN ARCHITECTURE

É um paradigma arquitetônico que se baseia no uso de eventos do sistema. Essas ocorrências alteram o estado dos objetos, assim, o sistema responde a tais ações.

O princípio da arquitetura orientada a eventos é o de que os componentes reajam a episódios que ocorram dentro do sistema ou em sistemas externos. Podem significar qualquer situação, desde a solicitação de um usuário até a atualização em um banco de dados.

Um exemplo simples, porém, fundamental, é o de um botão em uma interface, que, ao ser clicado, dispara um evento *onclick* e executa a chamada em alguma classe relacionada.

Como mencionado, trata-se de um exemplo simples e, talvez, não reflita a complexidade de uma arquitetura orientada a eventos. Para tanto, é necessário entender que a essência dessa estrutura é a arquitetura de sistemas complexos, que reagem, de forma dinâmica, a uma variedade de eventos que ocorrem em tempo real.

Agora, imagine que esse mesmo botão:

1. Chame um código Javascript, que adicione um produto em um carrinho, fazendo uma requisição assíncrona.

2. Realize uma série de operações, como atualização a banco, envio de e-mail etc.

3. Envie a resposta ao cliente com mensagem.

4. Atualize a interface do cliente.

O funcionamento do botão estaria mais alinhado aos princípios da arquitetura orientada a eventos.

Existem muitos outros padrões mais avançados com esse mesmo princípio, por exemplo, o *Publish Subscribe*, em que os componentes se inscrevem e o sistema de barramento de eventos encaminha a ocorrência a todos os participantes. Outro padrão que pode ser citado é o *event-driven messaging*, em que os componentes se comunicam por meio de mensagens, enviando os eventos a serem executados.

MICROSERVICES ARCHITECTURE

Essa arquitetura preconiza que cada serviço seja independente um do outro, sendo que todos se comunicam por meio de protocolos bem definidos, como http ou mensagem assíncrona. A forma de desacoplamento faz com que o sistema seja muito mais flexível, podendo ser escalável e mais modularizado.

Os microsserviços devem seguir alguns requisitos na sua implementação, quais sejam:

REQUISITO 1
Devem ser independentes, ou seja, devem ser implementados e removidos independentemente.
REQUISITO 2
A implantação deve ser rápida.
REQUISITO 3
Como um microsserviço poderá ter uma demanda maior que outro em um certo momento, torna-se necessário que este possa escalar de maneira independente.
REQUISITO 4
Uma falha ocorrida em um microsserviço não deve afetar outros.

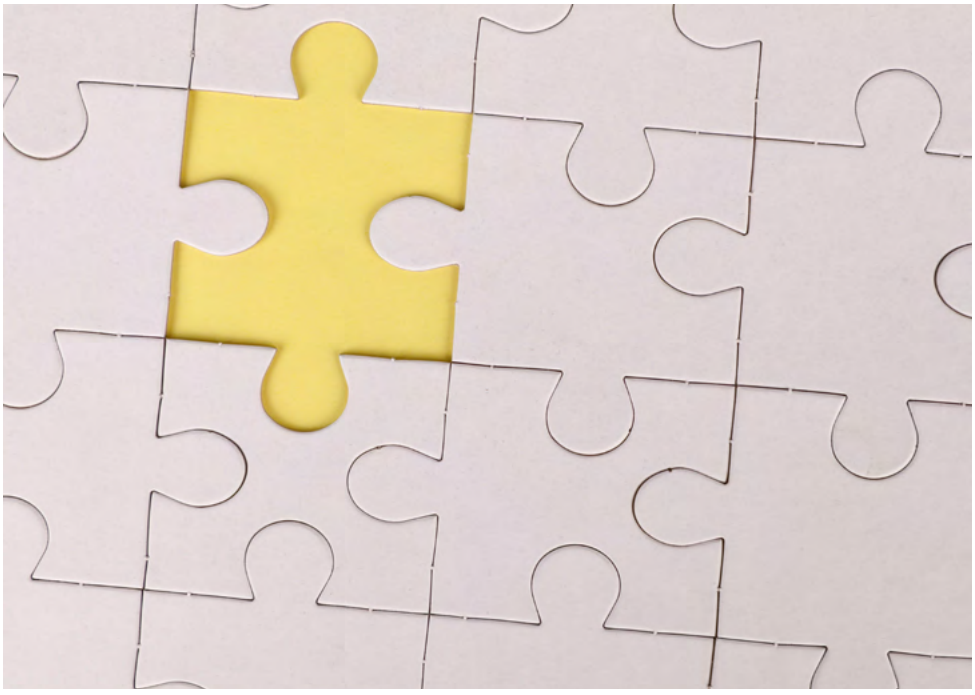
Atualmente, quando se fala em arquitetura/design, desenvolvimento e entrega de software, a mais nova tendência são os microsserviços, os quais apareceram, ultimamente, como um novo paradigma, pois compõem vários pequenos serviços, cada um rodando seu próprio processo e se comunicando por um meio leve de comunicação (Pivetta, 2023).

> Para melhor compreensão desse estilo arquitetural, bem como as vantagens e desvantagens inerentes, é relevante explicar um estilo arquitetural mais tradicional, o monolítico. Uma aplicação monolítica é desenvolvida como uma unidade única e é organizada em módulos dependentes da aplicação. Esses módulos não podem, assim, serem reutilizados por outra aplicação (Carrusca, 2018, p. 10).

Considerando um sistema que possui funcionalidades, como autenticação, gerenciamento de catálogo, carrinho, transação financeira, dentre outras, cada uma delas poderia ser atribuída a um microsserviço.

Embora possamos identificar que se trata de um sistema de comércio eletrônico, a arquitetura de microsserviços permite tais ações, individualmente, possam ser aplicadas a outros propósitos.

Segundo Gonçalves (2023), para nos assegurarmos de que estamos diante de uma arquitetura de microsserviços, é necessário que tenhamos em atenção certos pontos centrais, tais como: produzir de maneira independente, desenvolver em função de um modelo de negócio, possuir próprio estado, manter o alinhamento da arquitetura e organização.



SERVICE-ORIENTED ARCHITECTURE

Service-Oriented Architecture (SOA) é um paradigma de design de software, cujo foco é a construção de sistemas modulares, projetados para serem reutilizáveis, interagindo entre si por meio de serviços.

> Essa solução promove a reutilização de código e permite uma maior facilidade de manutenção, ao mesmo tempo que possibilita substituir um serviço por outro, sem haver perceção disso. No entanto, isso só é possível se a semântica for a mesma. SOA é uma arquitetura que oferece níveis de abstração, conectividade heterogênea e orquestração de serviços, assim como a possibilidade de alinhar os objetivos do negócio com as capacidades dos programadores. Apesar de haver grande esforço no sentido da utilização dessa arquitetura, a verdade é que ainda existe uma falta de consenso em como utilizá-la de forma correta (Gonçalves, 2023, p. 11).

Para exemplificar o uso da arquitetura SOA, em uma aplicação desse parâmetro no comércio eletrônico, teríamos os serviços:

AUTENTICAÇÃO
Com as funcionalidades de registro, login e geração de tokens.
CATÁLOGO DE PRODUTOS
Com as funcionalidades de listar e adicionar produtos.
CARRINHO
Com as funcionalidades de adicionar ou excluir produtos do carrinho.
PAGAMENTO
Com as funcionalidades de pagar por cartão ou boleto bancário.

Comparação com microsserviços

Ambos padrões são estilos de arquitetura que utilizam componentes independentes, os quais, no entanto, diferenciam-se em alguns pontos, como:

ZOOM NO CONHECIMENTO

- **Escopo e granularidade**: microsserviços são focados em uma funcionalidade específica, ao passo que o SOA abrange múltiplas funcionalidades.
- **Comunicação**: nos microsserviços, usualmente, usa-se *APIs Restful*, ao passo que, em SOA, usa-se o protocolo SOAP.
- **Dados**: em SOA, os dados podem ser compartilhados entre serviços; em microsserviços, os dados são centralizados, possuindo uma base independente.

Segundo Gonçalves (2023), alguns críticos afirmam que a estrutura dos microsserviços não é algo novo e que se trataria de uma arquitetura orientada a serviços (SOA). De acordo com o autor:

> Ambas surgem com o mesmo objetivo: o de transformar arquiteturas inflexíveis (legacy) em arquiteturas orientadas a serviços, as quais são mais flexíveis e ágeis para desenvolver soluções digitais inovadoras. Em um nível elevado de abstração, ambas as arquiteturas são um estilo de arquitetura que estrutura um sistema como um conjunto de serviços (Gonçalves, 2023, p. 10).

EM FOCO

Acesse seu ambiente virtual de aprendizagem e confira a aula referente a este tema.. **Recursos de mídia disponíveis no conteúdo digital do ambiente virtual de aprendizagem.**

NOVOS DESAFIOS

Nas atividades diárias, você se deparará com momentos em que conhecimentos adquiridos, muitas vezes teóricos, se aplicarão a situações reais para as quais você poderá propor alternativas para resolução.

No início do nosso estudo, você foi questionado a respeito de uma situação específica: como resolveria o desafio imposto e que tipo de arquitetura usaria?

Vimos, neste tema de aprendizagem, arquiteturas, como SOA ou microsserviços, poderiam se adaptar bem. Assim, sendo a arquitetura de microsserviços mais independente, ela se adaptaria melhor ao cenário proposto.

O mercado de trabalho busca profissionais que tenham um conhecimento da arquitetura e do desenvolvimento, pois uma aplicação deve ser projetada com padrões desde o início, permitindo um software de melhor qualidade e maior reconhecimento no mercado.

No início da jornada, foi solicitada uma pesquisa a respeito do padrão de arquitetura que poderia auxiliar na flexibilidade do código, usando a criação de objetos sem especificar as classes exatas. A resposta a esse questionamento é o padrão Factory Method, pois ele permite a criação de objetos sem especificar suas classes exatas, delegando responsabilidades de instanciar objetos para subclasses específicas.

O Factory Method é útil quando uma classe não conhece antecipadamente as classes de objetos que deve criar, ou quando queremos que as subclasses possam modificar a lógica de criação de objetos. O Factory Method promove o princípio do encapsulamento, permitindo a criação de objetos de maneira flexível e extensível.

1. "Imagine que você esteja usando o seu aparelho de MP3 favorito, como o Itunes. Você pode usar a interface para adicionar novas músicas, com o nome das trilhas. O aparelho cuida de uma pequena base de dados, contendo todas as suas músicas, com os nomes associados a elas e os respectivos dados. Além disso, ele cuida da reprodução das músicas e, enquanto faz isso, atualiza constantemente a sua interface de usuário para exibir o nome da música atual, o tempo de execução e outras informações. Bem, o que faz tudo isso funcionar é o modelo-visualização-controlador" (Freeman; Freeman, 2007, p. 421).

   A respeito do padrão MVC, assinale a alternativa correta:

   a) O padrão MVC separa a aplicação em três componentes principais: *Model*, *View* e *Controller.*
   b) No padrão MVC, o *Model* é responsável por gerenciar a interação do usuário com a interface gráfica.
   c) No padrão MVC, a *View* é responsável pela manipulação dos dados e pela lógica de negócios da aplicação.
   d) O padrão MVC não permite a separação clara de responsabilidades entre os componentes da aplicação.
   e) No padrão MVC, a comunicação entre o *Model* e a *View* ocorre diretamente, sem a intervenção do *Controller.*

2. "Atualmente, quando se fala em arquitetura/design, desenvolvimento e entrega de software, a mais nova tendência são os microsserviços. Apareceram ultimamente como um novo paradigma, pelo fato de compor vários pequenos serviços, cada um rodando seu próprio processo e se comunicando por um meio leve de comunicação" (Pivetta, 2023, p. 13).

   A respeito dos microsserviços considere as afirmativas a seguir:

   I - Devem ser independentes, ou seja, devem ser implementados e removidos independentemente.
   II - Como um microsserviço poderá ter uma demanda maior que outro em um certo momento, torna-se necessário que este possa escalar de maneira independente.
   III - Uma falha ocorrida em um microsserviço não deve afetar outros.

   É correto o que se afirma em:

   a) I, apenas.
   b) III, apenas.
   c) I e II, apenas.
   d) II e III, apenas.
   e) I, II e III.

3. "Para melhor compreensão deste estilo arquitetural, bem como as vantagens e desvantagens inerentes, é relevante explicar um estilo arquitetural mais tradicional, o monolítico. Uma aplicação monolítica é desenvolvida como uma unidade única e é organizada em módulos dependentes da aplicação. Estes módulos não podem assim, serem reutilizados por outra aplicação" (Carrusca, 2018, p. 36).

   A respeito do estilo arquitetural monolítico, é correto afirmar:

   a) No padrão de arquitetura monolítica, a aplicação é desenvolvida como uma unidade única e é organizada em módulos dependentes da aplicação.
   b) O padrão de arquitetura monolítica promove a modularização e a reutilização de componentes para facilitar a escalabilidade e a manutenção da aplicação.
   c) Na arquitetura monolítica, os diferentes módulos da aplicação podem ser facilmente implantados e escalados de forma independente.
   d) A arquitetura monolítica é altamente adaptável a mudanças nos requisitos de negócio, devido à sua estrutura flexível e modular.
   e) Uma aplicação monolítica é geralmente composta por múltiplas unidades de código independentes, cada uma responsável por uma função específica da aplicação.

REFERÊNCIAS

CARRUSCA, A. V. **Gestão de micro-serviços na Cloud e Edge**. 2018. Dissertação (Mestrado em Engenharia Informática) – Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa, Lisboa, 2018.

FREEMAN, E.; FREEMAN, E. **Use a cabeça**: padrões de projetos. 2. ed. Rio de Janeiro: Alta Books, 2007.

GONÇALVES, F. D. de S. **Arquitetura de microsserviços e o caso da Framework Lithium da Primavera BSS**. 2023. Dissertação (Mestrado em Engenharia Informática) - Escola de Engenharia, Universidade do Minho, Lisboa, 2023.

PIVETTA, L. **Arquitetura de software**: microsserviços. 2023. Trabalho de Conclusão de Curso (Graduação Tecnológica em Sistemas para Internet) – Colégio Politécnico, Universidade Federal de Santa Maria, Santa Maria, 2023.

GABARITO

1. Alternativa A.

   A alternativa A está correta, pois trata-se da divisão entre os componentes. A alternativa B está incorreta, pois o *Model* representa a parte dos dados e a lógica. A alternativa C está incorreta, pois a *View* é a camada de apresentação. A alternativa D está incorreta, pois a separação de responsabilidades é clara nesse modelo. A alternativa E está incorreta, pois a camada *Controller* intermedeia a comunicação.

2. Alternativa E.

   Todas as afirmativas estão corretas.
   A afirmativa I está correta, pois o microsserviço deve ser independente. A afirmativa II está correta, pois um microsserviço pode ter uma demanda maior que outro. A afirmativa III está correta, pois uma falha ocorrida em um microsserviço não deve afetar outros.

3. Alternativa A.

   A alternativa A está correta, pois descreve o conceito central de uma arquitetura monolítica. A alternativa B está incorreta, pois a arquitetura monolítica não enfatiza a modularização. A alternativa C está incorreta, pois a aplicação é implantada e escalada como uma única unidade. A alternativa D está incorreta, pois a estrutura monolítica tende a ser menos flexível do que abordagens mais modernas. A alternativa E está incorreta, pois não é possível afirmar que uma aplicação monolítica é, geralmente, composta por múltiplas unidades de código independentes, cada uma responsável por uma função específica da aplicação.

MINHAS ANOTAÇÕES

TEMA DE APRENDIZAGEM 6

ANTIPADRÕES

MINHAS METAS

- Definir um antipadrão de projetos, com sua abrangência.
- Determinar as características que podem levar a maus hábitos de programação.
- Estabelecer os principais antipadrões.
- Despertar, no estudante, o interesse pela compreensão dos impactos que maus hábitos de programação podem causar.
- Despertar, no estudante, o interesse pelo entendimento de possíveis abordagens para evitar os padrões.
- Reconhecer os antipadrões.
- Entender boas práticas de programação.

INICIE SUA JORNADA

A existência dos padrões e antipadrões nos levam a considerar um problema que ocorre em nossas atividades diárias na jornada de desenvolvimento de softwares corporativos. Como podemos **evitar armadilhas comuns** e conhecidas em um ambiente de desenvolvimento?

A pergunta nos leva a pensar a respeito dos **antipadrões de design**, que são os parâmetros que resultam em práticas inadequadas e consequências negativas no desenvolvimento dos projetos de software e como essas práticas que, muitas vezes, são consideradas atalhos para prazos apertados e acabam resultando em problemas futuros.

Assim, estudante, elabore uma pesquisa a respeito dessas práticas inadequadas e consequências negativas, que resultam dos padrões de projeto. Você deve citar três práticas inadequadas, descrever três consequências e sua pesquisa deve conter de 10 a 20 linhas.

Reflita a respeito das práticas negativas que possam ocorrer em seu desenvolvimento, perguntando-se o que elas podem causar em seu ambiente de software. Essas consequências podem ser evitadas?

São situações que ocorrerão no seu ambiente de trabalho, as quais devem ser analisadas, e você deve refletir a respeito de como evitar caminhos considerados fáceis em um primeiro momento e que possam dificultar seus projetos em um médio prazo.

PLAY NO CONHECIMENTO

Ouça nosso podcast acerca do tema *Evitando Armadilhas de Design*. Acesse e atualize seus conhecimentos! **Recursos de mídia disponíveis no conteúdo digital do ambiente virtual de aprendizagem.**

VAMOS RECORDAR?

Este vídeo do canal de Rony Marcolino aborda os antipadrões de projetos. Acesse-o e atualize-se!

Disponível em: **https://www.youtube.com/watch?v=YY4mLXOm6Ms**

DESENVOLVA SEU POTENCIAL

Os antipadrões também possuem um local muito importante na engenharia de software, pois a sua observação oferece insights para o desenvolvimento de programas, fazendo com que erros comuns e armadilhas possam ser evitadas, muitas vezes, por pressão de tempo ou até encontrar caminhos que, à primeira vista, pareçam mais diretos e mais fáceis.

Compreender os antipadrões é fundamental para os desenvolvedores de software, pois ajuda-os a reconhecer os padrões mal aplicados e que poderão gerar problemas futuros no desenvolvimento de programas.

ANTIPADRÃO OBJETO DEUS

Uma técnica comum é separar o código em diversas classes e camadas, de forma a distribuir a lógica em classes específicas, na melhor atribuição de responsabilidades, sendo que cada uma delas fique responsável por uma divisão. O que acaba ocorrendo, algumas vezes, é que o desenvolvedor aloca, em uma única classe, todo o trabalho da aplicação, ou acaba fazendo com que uma classe contenha todos os dados do aplicativo.

Essa abordagem resulta em excessivo tráfego de mensagens, causando um gargalo na aplicação e degradando a performance. Conforme Smith e Williams (2012, p. 12, tradução nossa): "Uma possível solução para essa abordagem seria refatorar o design para distribuir a inteligência de forma uniforme entre as classes superiores e manter o relacionamento de dados juntos".

> Um God Object contém código desordenado que é difícil de manter, estender, usar, testar e integrar com outras partes do aplicativo. Eles violam o princípio da responsabilidade única (SRP) e a Lei de Deméter ou princípio do conhecimento mínimo, que reduz as dependências entre classes e ajuda a construir componentes que são fracamente acoplados (Macoratti, 2023, on-line).

APROFUNDANDO

Lei de Deméter aborda um problema específico do acoplamento (nível de dependência entre as classes). Ela foi abordada pela primeira vez na Universidade de Northeastern, em Boston, em 1987. Essa lei define uma série de regras que, se seguidas, reduzem o acoplamento.

> A lei especifica que os métodos e propriedades de uma classe só devem invocar os membros dos seguintes objetos:
>
> - O objeto no qual o método ou propriedade é declarado.
> - Os objetos que são passados para o membro, utilizando os seus parâmetros.
> - Os objetos que são declarados dentro da mesma classe como o método de chamada ou propriedade. Esses são os objetos componentes diretos da classe que contém o membro.
> - Os objetos globais, embora geralmente devam ser minimizados (Macoratti, 2013, on-line).

ANTIPADRÃO CÓDIGO ESPAGUETE

É muito comum encontrarmos no desenvolvimento um código enorme, reunindo, no mesmo bloco de códigos, várias regras de negócio e múltiplas definições, que deveriam estar separadas, com vários redirecionamentos de fluxo do programa.

Código espaguete é uma gíria usada para se referir a uma teia emaranhada de código-fonte de programação, em que o controle dentro de um programa salta para todos os lados e é difícil de seguir. O código espaguete normalmente possui muitas instruções GOTO, e é comum em programas antigos, que usavam tais instruções extensivamente (Rouse, 2017).

Não se trata de uma boa prática, pois:

PONTO 1
Torna difícil a manutenção do software.
PONTO 2
Tem baixa legibilidade devido à grande quantidade de código e regras diferentes no mesmo bloco.

PONTO 3

É frágil, ou seja, se alterar uma parte tem grandes chances de afetar o sistema todo.

PONTO 4

Dificulta os testes.

PONTO 5

Tem uma escalabilidade limitada; à medida que cresce, o software se torna difícil de manter e se adaptar a novos requisitos.

ANTIPADRÃO GRANDE BOLA DE LODO

Muitos desenvolvedores iniciam a codificação da aplicação sem o devido planejamento da arquitetura. Eles acabam por iniciar a arquitetura padrão *n-tyer*, porém sem uma definição clara das camadas e do desenvolvimento do aplicativo, criando camadas implícitas e separando o código em módulos. Infelizmente, essa prática pode acarretar código desorganizado, resultando em falta de clareza nas responsabilidades.

Um exemplo poderia ser encontrado em um sistema legado sem uma arquitetura claramente definida ao longo do tempo. Com códigos adicionais e correções de bugs, o código se tornou um emaranhado difícil de se entender e de se manter. Considere um sistema de gerenciamento de pedidos. Nele, você pode encontrar:

- Falta de estruturação.
- Acoplamento excessivo.
- Duplicação de código.
- Complexidade desnecessária.
- Dificuldade de manutenção.

ANTIPADRÃO NÚMERO MÁGICO

São números digitados diretamente no seu código-fonte, sem um aparente significado, ou seja, se um outro desenvolvedor realizasse a manutenção no seu código, seria um número sem um significado claro para ele. Por esse motivo, a denominação número mágico.

Uma boa prática seria armazenar valores em arquivos de configuração, constantes ou tabelas do banco de dados.

ANTIPADRÃO DEPENDÊNCIA CIRCULAR

Ocorre quando um objeto A depende de outro B. Este, por sua vez, necessita do C, o qual, por seu turno, precisa do objeto inicial A, gerando uma dependência cíclica. Sendo assim, consideremos um sistema de gestão de pedidos com três módulos:

ZOOM NO CONHECIMENTO

- **Módulo gestão de pedidos**: gerencia o ciclo de vida dos pedidos.
- **Módulo gestão de estoque**: gerencia o estoque, os produtos e os itens.
- **Módulo gestão de clientes**: gerencia informações dos clientes.

A gestão de pedidos depende da gestão de estoque. Para criar um novo pedido, é necessário consultar o estoque. Ao mesmo tempo, a gestão de pedidos depende da gestão de clientes para associar o cliente com o pedido. O módulo de clientes depende do módulo de gestão de pedidos para atualizar o histórico de pedidos. Por vezes, a dependência circular pode ser a lógica da aplicação, porém, é importante reconhecer que modificações em um módulo podem ter efeitos em outro. Sendo assim, é uma boa prática projetar o sistema, minimizando as dependências circulares.

INDICAÇÃO DE FILME

Hackers: piratas de computador

Um adolescente conhecido como Zero Cool (Jonny Lee Miller) é uma lenda entre os hackers, pois, com apenas 11 anos, ele inutilizou 1.507 computadores em Wall Street, provocando um caos total no mundo das finanças.

Refletindo sobre a história: *Hackers* retrata um grupo de jovens talentosos que invadem sistemas de computador. Embora o filme seja mais voltado para a cultura geek, ele ilustra o antipadrão *God Class* ou *Monolithic Class*, em que uma classe de software se torna excessivamente grande, complexa e difícil de entender ou modificar.

COMO RECONHECER ANTIPADRÕES

Ao identificar um código que não está muito claro, muitas vezes, conseguimos identificar alguns padrões básicos da codificação.

Alguns fatores são primordiais para se pensar durante o desenvolvimento. Por exemplo: "performance é um atributo emergente do software, já que resulta de interações entre componentes de software, plataformas, usuários e contextos" (Arcelli; Cortelessa; Trubiani, 2012, p. 2, tradução nossa). Acoplamento é outra característica a ser observada nos códigos-fonte de nossas aplicações.

Portanto, estudante, observando codificação, performance e acoplamento, você conseguirá cobrir uma boa parte da sua aplicação e reconhecerá a existência de antipadrões que devem ser analisados e corrigidos.

COMO EVITAR ANTIPADRÕES

É fundamental para garantir um desenvolvimento sem maiores contratempos e uma evolução bem estruturada. Nesse sentido, alguns pontos devem ser considerados:

PONTO 1 – CONHECIMENTO E CONSCIENTIZAÇÃO

Antes de tudo, é essencial que a equipe tenha um sólido conhecimento acerca dos antipadrões existentes e esteja consciente dos impactos negativos que podem causar. Isso envolve compreender os conceitos por trás dos antipadrões, reconhecer sinais de sua presença e estar aberto a discutir e abordar esses problemas proativamente.

PONTO 2 – REVISÃO DE CÓDIGO

A revisão de código é uma prática fundamental para garantir a qualidade do software. Antes de integrar o código ao repositório principal, os membros da equipe devem revisar o código uns dos outros em busca de possíveis antipadrões, bugs e oportunidades de otimização. Isso ajuda a identificar e corrigir problemas precocemente, reduzindo o impacto negativo no desenvolvimento futuro.

PONTO 3 – PADRÕES DE PROJETO

Utilizar padrões de projeto reconhecidos e estabelecidos pode ajudar a prevenir antipadrões. Os padrões de projeto oferecem soluções comprovadas para problemas comuns de desenvolvimento de software, promovendo uma arquitetura mais robusta, modular e de fácil manutenção.

PONTO 4 – REFATORAÇÃO

A refatoração é o processo de reestruturar o código sem alterar seu comportamento externo. Ela é essencial para eliminar antipadrões que possam surgir durante o desenvolvimento. A refatoração contínua ajuda a manter o código limpo, legível e eficiente, reduzindo a probabilidade de antipadrões surgirem e se propagarem.

PONTO 5 – TESTES AUTOMATIZADOS

Testes automatizados são cruciais para garantir que o software funcione conforme o esperado e para identificar regressões após mudanças no código. A falta de testes adequados pode resultar em antipadrões, como código de difícil manutenção e baixa qualidade.

PONTO 6 – BOAS PRÁTICAS DE CODIFICAÇÃO

Adotar e seguir boas práticas de codificação ajuda a evitar a introdução de antipadrões. Isso inclui escrever código claro, modular e bem documentado, além de seguir convenções de nomenclatura e estilo de código consistentes.

PONTO 7 – REVISÃO DE DESIGN

Assim como a revisão de código, a revisão de design é importante para identificar e corrigir antipadrões arquiteturais e de design. Isso envolve analisar a estrutura do software, a organização dos componentes e a escalabilidade da solução proposta.

PONTO 8 – FEEDBACK E MELHORIA CONTÍNUA

Por fim, é crucial estabelecer um ciclo de feedback e melhoria contínua. Isso inclui solicitar retornos regularmente, tanto de colegas quanto de clientes, e estar aberto a aprender com os erros e aprimorar constantemente os processos de desenvolvimento para evitar a reincidência de antipadrões.

Levando em consideração todos esses pontos ressaltados, seu código será bem estruturado e você evitará os antipadrões.

EM FOCO

Acesse seu ambiente virtual de aprendizagem e confira a aula referente a este tema. **Recursos de mídia disponíveis no conteúdo digital do ambiente virtual de aprendizagem.**

NOVOS DESAFIOS

Ao desempenhar seu papel como desenvolvedor no mercado de trabalho, a identificação de problemas no código-fonte será muito importante, pois isso permitirá que você identifique problemas comuns no código real do mundo profissional, ao reconhecer antipadrões, como o *Spaghetti Code*, em um projeto real poderá ajudar a priorizar pontos como refatoração constante, tendo como objetivo a melhoria do seu código.

No início de sua jornada, você foi incitado a elaborar uma pesquisa a respeito das práticas inadequadas e consequências negativas, que resultam dos padrões de projeto. Três práticas inadequadas e três consequências foram solicitadas para reflexão.

Podem ser práticas inadequadas:

- **Acoplamento**: quanto mais acoplados estiverem os objetos, mais difícil será modificá-los de forma independente. Isso pode levar a uma série de problemas no código, incluindo dificuldade de manutenção, dificuldade no teste, baixa reutilização, dificuldade de escalonamento, entre outros.
- **Números mágicos**: valores numéricos usados de forma direta (*hard coded*) no código, sem uma explicação clara a respeito do seu significado.
- **Código sem estrutura clara**: dificuldade de manutenção e ocasionando perda de performance.

E podemos citar como consequências:

- **Baixa manutenibilidade**: com alto acoplamento, código confuso ou outras práticas ruins a manutenção será prejudicada.
- **Complexidade excessiva**: muitas vezes, um projeto relativamente simples pode se tornar complexo devido a práticas adotadas.
- **Risco de erros**: quanto mais confuso e com exceções, aumenta-se o risco de erros.

Assim, com o entendimento de todas as práticas mencionadas, você se destacará no mercado de trabalho, tornando-se um profissional muito procurado pelas empresas de software.

1. "Um God Object contém código desordenado que é difícil de manter, estender, usar, testar e integrar com outras partes do aplicativo. Eles violam o princípio da responsabilidade única (SRP) e a Lei de Deméter ou princípio do conhecimento mínimo que reduz as dependências entre classes e ajuda a construir componentes que são fracamente acoplados" (Macoratti, 2023, on-line).

   Referente a esse assunto, assinale a alternativa correta:

   a) O chamado objeto deus é um objeto que divide o código por responsabilidades.
   b) O objeto deus centraliza o código, mas divide as responsabilidades entre as classes.
   c) O chamado objeto deus é um objeto que concentra o código com um único bloco, sem divisão de responsabilidades.
   d) Por ter todo o código centralizado em um único objeto a manutenção se torna mais fácil.
   e) O objeto deus é uma prática recomendada de design de software que ajuda a simplificar a arquitetura do sistema.

2. "Código espaguete é uma gíria usada para se referir a uma teia emaranhada de código-fonte de programação em que o controle, dentro de um programa, salta para todos os lados, e é difícil de seguir. O código espaguete normalmente possui muitas instruções GOTO e é comum em programas antigos, que usavam tais instruções extensivamente" (Rouse, 2017, on-line, tradução nossa).

   A respeito do código espaguete, analise as afirmativas a seguir:

   I - Não é uma boa prática, pois é frágil, ou seja, se alterar uma parte, há grandes chances de afetar o sistema todo.
   II - Tem uma escalabilidade limitada; à medida que cresce, o software se torna difícil de manter e se adaptar a novos requisitos.
   III - Tem baixa legibilidade devido à grande quantidade de códigos e regras diferentes no mesmo bloco.

   Assinale a alternativa correta:

   a) I, apenas.
   b) III, apenas.
   c) I e II, apenas.
   d) II e III, apenas.
   e) I, II e III.

3. "A Grande Bola de Lodo é um antipadrão arquitetônico. Refere-se a uma arquitetura que carece de qualquer design modular e, portanto, torna-se apenas uma massa de código desorganizado sem qualquer estrutura real. Geralmente é o resultado do acréscimo de novos recursos ao longo do tempo, sem o esforço necessário para projetar como esses recursos devem interagir de maneira modular e sustentável" (Big..., 2021, on-line, tradução nossa).

   Referente a esse assunto, assinale a alternativa correta:

   a) A Grande Bola de Lama é uma arquitetura altamente modular e bem estruturada, facilitando a adição de novos recursos ao sistema de forma organizada.
   b) É um modelo arquitetônico ideal que promove a flexibilidade e a adaptabilidade do sistema às mudanças de requisitos.
   c) Abordagem de design bem-sucedida que prioriza a simplicidade e a eficiência na implementação de sistemas de software.
   d) É um antipadrão arquitetônico que surge quando os desenvolvedores dedicam tempo e esforço suficientes para planejar e projetar a interação entre os diferentes componentes do sistema.
   e) É um antipadrão arquitetônico que surge quando novos recursos são adicionados ao sistema sem considerar como eles devem interagir de maneira modular e sustentável.

REFERÊNCIAS

ARCELLI, D.; CORTELESSA, V.; TRUBIANI, C. Antipattern-based model refactoring for software performance improvement. *In*: QOSA – INTERNATIONAL ACM SIGSOFT CONFERENCE ON QUALITY OF SOFTWARE ARCHITECTURES, 8., 2012, Bertinoro. **Proceedings** [...]. New York: Association for Computing Machinery, 2012.

BIG ball of mud. **DevIQ**, [*s. l.*], 9 maio 2021. Disponível em: https://deviq.com/antipatterns/big-ball-of-mud. Acesso em: 25 jun. 2024.

MACORATTI, J. C. A Lei de Demeter (LoD). **Macoratti.net**, [*s. l.*], 20 jan. 2013. Disponível em: https://www.macoratti.net/13/01/net_dem1.htm. Acesso em: 25 jun. 2024.

MACORATTI, J. C. Apresentando o antipadrão: *god object*. **Macoratti.net**, [*s. l.*], 6 jun. 2023. Disponível em: https://macoratti.net/21/01/c_godobj1.htm. Acesso em: 25 jun. 2024.

ROUSE, M. What Does Spaghetti Code Mean? **Techopedia**, Panama City, 1 feb. 2017. Disponível em: https://www.techopedia.com/definition/9476/spaghetti-code. Acesso em: 25 jun. 2024.

SMITH, C.; WILLIAMS, L. Software performance antipatterns for identifying and correcting performance problems. *In*: WOSP – INTERNATIONAL WORKSHOP ON SOFTWARE AND PERFORMANCE, 2., 2000, Ottawa. **Proceedings** [...]. New York: Association for Computing Machinery, 2000.

GABARITO

1. Alternativa C.

   A alternativa A está incorreta, pois o objeto deus centraliza o código.
   A alternativa B está incorreta, pois as responsabilidades não são divididas.
   A alternativa C está correta, pois o objeto deus concentra o código, sem dividir as responsabilidades.
   A alternativa D está incorreta, pois não é o que ocorre; a manutenção é mais fácil com classes específicas.
   A alternativa E é incorreta, pois o uso do objeto deus não é uma prática recomendada.

2. Alternativa E.

   Todas as afirmativas estão corretas.
   A afirmativa I está correta, pois, quando alteramos uma parte do código e não a modularizamos, há grandes chances de que afete outras partes.
   A afirmativa II está correta, pois a falta de estruturação e organização torna o código menos flexível e mais difícil de estender ou modificar, limitando sua escalabilidade.
   A afirmativa III está correta, pois a presença de muitas regras e funcionalidades em um único bloco de código dificulta a compreensão e manutenção do software.

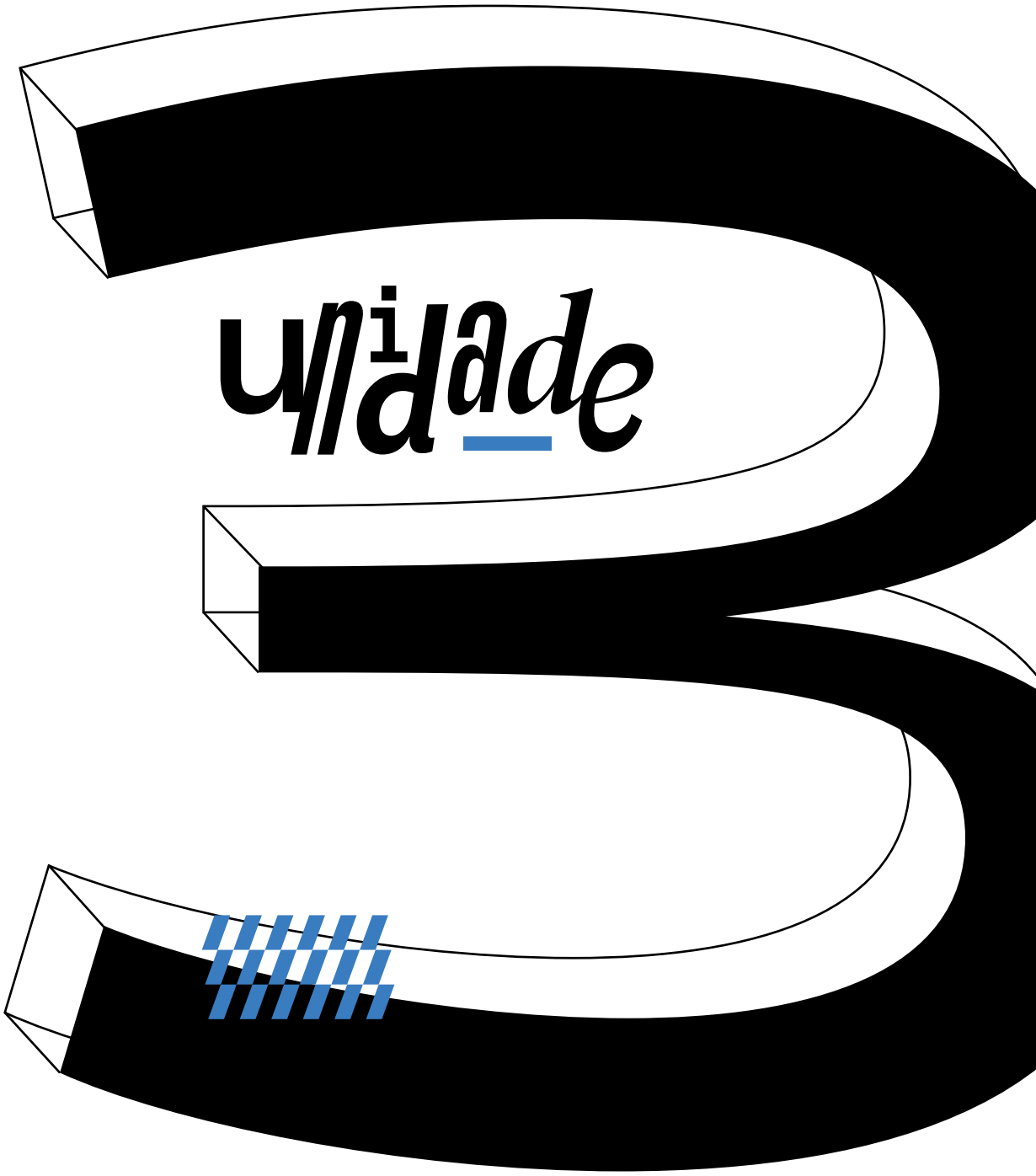
3. Alternativa E.

   A alternativa A está incorreta, pois a descrição é o oposto e destaca a falta de design modular.
   A alternativa B está incorreta, pois esse antipadrão não é nem modular nem sustentável, além de ser de difícil manutenção.
   A alternativa C está incorreta, pois a falta de estrutura desse antipadrão leva à ineficiência.
   A alternativa D está incorreta, pois é o resultado da falta de esforço na criação de um design modular.
   A alternativa E está correta, pois destaca a falta de design modular e desorganização, refletindo esse antipadrão.

MINHAS ANOTAÇÕES

MINHAS ANOTAÇÕES

Unicesumar



TEMA DE APRENDIZAGEM 7

TENDÊNCIAS E FUTURO

MINHAS METAS

- Abordar novas tecnologias e arquiteturas.
- Demonstrar que os padrões podem evoluir com as tecnologias aplicadas.
- Apresentar soluções para novos cenários que se apresentam no cotidiano.
- Discutir os design patterns de Cloud e internet das coisas (IoT).
- Relembrar princípios do design patterns.
- Vislumbrar aplicações e novas soluções no mundo da IoT.
- Conscientizar os estudantes acerca das perspectivas de carreira nessas áreas.

INICIE SUA JORNADA

Quando abordamos o tema dos *design patterns*, muitas vezes, nos deparamos com cenários em que há um conjunto de situações já estabelecidas para problemas comuns no desenvolvimento de softwares. Contudo, o mundo está em constante evolução. Novas tecnologias, linguagens de programação ou arquiteturas surgem a cada dia, consequentemente, estabelecendo novos desafios e novas práticas a serem adotadas, de forma que é inevitável que nos questionemos: como podemos garantir que estamos adotando as melhores práticas em um cenário de constante evolução?

Os *design patterns* apresentam um papel fundamental no desenvolvimento de software e aplicação de boas práticas, porém, é importante termos o entendimento de que esses padrões vão além de soluções predefinidas, eles representam um conjunto de princípios e contextos, que nos ajudam a criar sistemas mais robustos, flexíveis e fáceis de manter.

Para que você compreenda melhor a importância desse assunto e se aprofunde no tema, elabore uma pesquisa de 10 a 20 linhas acerca dos **três princípios** dos *design patterns*.

É fundamental que reflitamos a respeito do uso dos *design patterns*. Ponderar acerca do tema nos ajuda a identificar padrões emergentes, que possam contribuir para uma prática mais sólida e atualizada, de forma que devemos nos questionar: qual a aplicação deles e a sua eficácia nas diversas situações que se apresentam no dia a dia?

PLAY NO CONHECIMENTO

Ouça nosso podcast acerca do tema Design Patterns e a Inteligência Artificial. Acesse e atualize seus conhecimentos. **Recursos de mídia disponíveis no conteúdo digital do ambiente virtual de aprendizagem.**

VAMOS RECORDAR?

O vídeo do canal de Elder Moraes discute a aplicação dos *design patterns* nos microsserviços. Acesse e se atualize!

Disponível em: **https://www.youtube.com/watch?v=S7oYU8qBKIE**.

DESENVOLVA SEU POTENCIAL

Em um momento em que vemos a tecnologia evoluir cada vez mais, com o aparecimento de novas arquiteturas o tempo todo, faz-se necessário que padrões de projeto acompanhem as tendências e novas estruturas para que também evoluam e apresentem abordagens para a prevenção as práticas erradas.

INDICAÇÃO DE FILME

O Círculo (The Circle)

The Circle é uma das empresas mais poderosas do planeta no ramo da internet. Ela é responsável por conectar os e-mails dos usuários com suas atividades diárias, suas compras e outros detalhes de suas vidas privadas.

Refletindo sobre a história: o filme mostra o uso constante de dispositivos distribuídos, sensores de internet das coisas, inteligência artificial ou outras tecnologias, que são tendências e exigem uma arquitetura muito bem trabalhada para a sua execução, sendo que os padrões de projeto são fundamentais para o correto desenvolvimento.

PADRÕES DE PROJETO NA NUVEM

Computação na nuvem é uma realidade cada vez mais presente em nosso cotidiano. Todos nós acessamos nossos e-mails em um webmail alocado na nuvem ou por meio de aplicativos de edição de texto na nuvem, sistemas ERP na nuvem, dentre outras aplicações.

Com todos esses recursos, podemos imaginar que exista uma programação extensa nos bastidores, seguindo uma arquitetura largamente adaptada, motivo de preocupação constante no que diz respeito a questões, como performance, otimização, manutenção, entre outras, de forma que alguns padrões de projetos foram definidos para essa nova abordagem de arquitetura.

Embaixador

Aplicativos baseados em nuvem resilientes exigem recursos, como interrupção de circuito, roteamento, medição e monitoramento, além da capacidade de fazer atualizações de configuração relacionadas à rede. Pode ser difícil ou impossível atualizar aplicativos herdados ou bibliotecas de código existentes para adicionar tais ferramentas, pois o código não será mais mantido ou não poderá ser facilmente modificado pela equipe de desenvolvimento.

> Chamadas de rede também podem exigir configuração significativa para conexão, autenticação e autorização. Se essas chamadas forem usadas em vários aplicativos, criados, usando vários idiomas e estruturas, as chamadas deverão ser configuradas para cada uma dessas instâncias. Além disso, funcionalidade de segurança de rede poderá precisar ser gerenciada por uma equipe central na sua organização. Com uma base de código grande, pode ser arriscado para essa equipe atualizar código de aplicativo com a qual não esteja familiarizada (Microsoft Learn, 2024, on-line).

O conceito principal do padrão de projeto Embaixador é isolar os clientes do serviço de comunicação, abstraindo a complexidade da comunicação de rede e oferecendo uma interface mais simples e confiável para os clientes.

O Embaixador pode ser implementado como uma classe em uma linguagem de programação específica ou como um componente em um sistema distribuído.

Nesse caso, o Embaixador seria responsável por gerenciar todas as iterações de rede em nome dos clientes, encapsulando a lógica relacionada à comunicação de rede.

Uma outra forma de aplicação, no contexto de serviços, por exemplo, seria por meio do uso do Amazon Web Services (AWS). O Embaixador pode ser um serviço gerenciado que fornece as funcionalidades de intermediação de comunicação. No AWS, ele pode ser implementado como uma função *lambda* ou serviço *gateway*.

APROFUNDANDO

Independentemente de como seja implementado, o papel do Embaixador permanece o mesmo: ele atua como um intermediário entre os clientes e o serviço de comunicação subjacente, simplificando a comunicação de rede e melhorando a eficiência e a confiabilidade do sistema.

PADRÕES DE PROJETO PARA MICROSSERVIÇOS

O padrão de arquitetura de microsserviços é definido para que a aplicação seja cada vez mais escalável e cada parte consiga operar de forma independente.

Cada microsserviço é uma parte do todo e atua de forma independente, tornando o processo de manutenção mais ágil e inteligente.

Os padrões de microsserviços incluem:

SINGLE RESPONSIBILITY PRINCIPLE

Cada microsserviço deve ter uma única responsabilidade e não deve ser misturado com outras atribuições (Principais..., 2022).

SERVICE DISCOVERY

Os microsserviços precisam ser capazes de se encontrar e se comunicar uns com os outros de forma dinâmica (Principais..., 2022).

API GATEWAY

Um ponto único de entrada para todas as chamadas de serviço, o que facilita a gestão de autenticação, autorização e roteamento (Principais..., 2022).

MICROSERVICE CHASSIS

Uso de uma camada comum de serviços, como registro, monitoramento e segurança, para diminuir a complexidade e aumentar a reutilização (Principais..., 2022).

CQRS (COMMAND QUERY RESPONSIBILITY SEGREGATION)

Cada microsserviço deve ser projetado para lidar com comandos e consultas de forma diferente, o que ajuda a evitar conflitos de concorrência (Principais..., 2022).

EVENT-DRIVEN ARCHITECTURE

Uso de eventos para comunicar mudanças de estado entre microsserviços, o que permite uma maior escalabilidade e flexibilidade (Principais..., 2022).

CIRCUIT BREAKER

Uso de um mecanismo para lidar com falhas de serviço, o que ajuda a evitar cascatas de falha (Principais..., 2022).

DATA REPLICATION

Replicação de dados entre microsserviços para garantir a disponibilidade e a tolerância a falhas (Principais..., 2022).

Esses são alguns dos principais padrões de microsserviços. Há outros que podem lhe auxiliar, estudante, na trajetória do desenvolvimento com os microsserviços.

PADRÕES DE PROJETO PARA ARQUITETURA *SERVLESS*

O conceito de *servless* tem ganhado força substancial. Com a promessa de escalabilidade, flexibilidade e economia, a arquitetura sem servidor representa uma mudança de paradigma no desenvolvimento e implantação de aplicativos. Embora a tradução literal seja "sem servidor", o fato é que um servidor continua existindo nos bastidores para hospedar o serviço na AWS, por exemplo. Esse termo se tornou amplamente aceito no mundo do desenvolvimento para descrever a abordagem de aplicativos da nuvem. A "ausência de servidor" quer dizer que o desenvolvedor não se preocupa com o gerenciamento da infraestrutura da nuvem.

A necessidade de padrões de design sem servidor eficazes aumenta à medida que mais usuários adotam esse tipo de arquitetura. Os padrões de projeto são soluções reutilizáveis para problemas comuns. Eles podem lhe orientar na criação da melhor arquitetura para aplicativos sem servidor eficientes, escaláveis e confiáveis. Os designs comprovados lidam com questões como gerenciamento de estados, orquestração de funções, manipulação de dados distribuídos e manutenção da segurança (Guide..., 2023).

Padrão de arquitetura orientada a eventos (EDA)

Trata-se de um padrão ideal para aplicativos *servless,* pois responde aos eventos em tempo real, usando recursos apenas quando ocorre um evento real. Assim, o EDA permite ampla escalabilidade e eficiência.

As aplicações financeiras, por exemplo, podem processar e reagir às alterações nos preços das ações em tempo real ou enviar milhares de notificações em resposta a eventos importantes (Guide..., 2023).

Padrão de Pub-Sub – publicar e assinar

Trata-se de um modelo de mensagens para arquiteturas sem servidor, que separa editores de eventos e assinantes. Nesse sistema, os editores criam eventos e enviam para um corretor de mensagens sem saber quem são os assinantes.

Em uma arquitetura desse modelo, temos entidades Publicador, Tópico e Assinante. Cada uma delas não é uma classe no sentido tradicional da linguagem orientada a objeto. São conceitos abstratos em que os eventos são publicados e nos quais os assinantes se registram para recebê-los. Esses tópicos podem ser serviço de mensagens AWS.

O processo de execução desse padrão poderia ser:

PASSO 1

Configurar o AWS IOT Core, que é um dos serviços projetados para facilitar a comunicação entre dispositivos conectados pela internet das coisas (IOT). Ele atua como um *broker* MQTT, permitindo a publicação e a subscrição de mensagens em tópicos específicos. Por exemplo, ele pode publicar um tópico chamado "sensor temperatura".

PASSO 2

Implementar o componente que atua como publicador de mensagens, enviando comunicações ao AWS IOT Core, o qual apenas publica sem saber quem enviou.

PASSO 3

Implementar outro componente ou dispositivo para atuar como assinante das mensagens. O assinante pode, por exemplo, inscrever-se como "sensor temperatura" para receber atualizações de temperatura.

PADRÕES DE PROJETO PARA IOT

Os padrões abrangem operações amplas, como o ciclo de vida dos dispositivos, ou podem ser mais específicos, como as melhores práticas para conectividade de dispositivos.

Padrão Comando

Espera-se que as alternativas IoT interajam com os dispositivos de tal forma que a solução, ou as pessoas que a utilizam, possam solicitar aos dispositivos, de forma confiável, que executem uma ação. Além disso, essa interação deve ocorrer por meio de redes intermitentes, muitas vezes, utilizando dispositivos com recursos limitados.

O processo executado é:

PASSO 1

O dispositivo configura-se para se comunicar com um terminal de protocolo para que mensagens de comando possam ser enviadas e recebidas.

PASSO 2

Um componente da solução publica uma Mensagem de comando direcionada a um ou mais dispositivos.

PASSO 3

O servidor usa o *endpoint* do protocolo para enviar a mensagem de comando para cada dispositivo configurado anteriormente.

PASSO 4

Após a conclusão da ação solicitada pelo Comando, o dispositivo publica uma mensagem de conclusão do comando para o servidor por meio do terminal do protocolo.

Telemetria

As soluções IoT precisam receber dados medidos de forma confiável e segura em um ambiente remoto por diferentes dispositivos, potencialmente, usando protocolos diferentes. Além disso, uma vez recebidos os dados medidos, a solução precisa processar e encaminhar as informações detectadas para uso pelos seus componentes.

O processo executado é:

PASSO 1

O dispositivo obtém uma medição de um sensor operando em um ambiente remoto da solução IoT (IOT Atlas, 2024).

PASSO 2

O dispositivo publica uma mensagem para o tópico da mensagem telemetry/deviceID contendo a medição. Essa informação é enviada por meio de um protocolo de transporte para um *endpoint* de protocolo disponibilizado pelo servidor (IOT Atlas, 2024).

PASSO 3

O servidor pode, então, aplicar uma ou mais regras às mensagens, a fim de realizar um roteamento refinado em alguns ou todos os dados de medição da mensagem, o qual pode enviar uma mensagem para outro componente da solução (IOT Atlas, 2024).

Esse padrão pode ser combinado com o padrão Pub-Sub mencionado anteriormente.

Para esse universo de IOT, existem outros inúmeros padrões que podem ser explorados e aplicados em suas soluções, por exemplo: inicialização do dispositivo, virtualização, virtualização com *middleware*, dentre outros.

> Não existe uma arquitetura de referência única para uma internet das coisas. Padrões de design são uma forma de construir soluções de arquitetura para casos de uso e classes de casos de uso específicos.

> O trabalho para encontrar soluções de arquitetura de sistema para problemas de internet das coisas levou à conclusão óbvia de que não existe uma arquitetura única apropriada para a maioria dos casos de uso de IoT. O espectro completo da IoT apresenta uma ampla gama de diversos casos de uso e restrições de recursos e, portanto, motiva uma série de soluções de arquitetura. Ainda assim, gostaríamos de basear a discussão num conjunto de referência de conceitos técnicos para ajudar a promover uma compreensão unificada e quebrar os silos de pensamento em torno da arquitetura IoT (Kost, 2014, on-line, tradução nossa).

PADRÕES DE PROJETO *DEVOPS*

O conceito *devops* está cada vez mais presente no dia a dia dos desenvolvedores. Essa arquitetura agiliza o ciclo de *deploy*, padronizando os processos das aplicações e suas instalações. Isso agiliza os processos de negócio, fazendo com que a empresa consiga reduzir custos nessa área.

A plataforma AWS oferece um portfólio de serviços amplo, que deve ser considerado para modernizar a TI.

A justificativa de padrão de design *devops* está embasada em quatro pilares, a saber: princípios, serviços AWS, plataforma AWS Devops e modelo de entrega de aplicativo.

Quatro princípios de *Devops* que geram valor para a AWS são:

PENSE DE FORMA HOLÍSTICA, NÃO APENAS TI

Lembre-se de que a transformação não envolve apenas TI, mas todo o fluxo de valor da organização, desde marketing e vendas até desenvolvimento e operações de TI.

PENSE MULTIFUNCIONALMENTE, NÃO APENAS EM TI

Lembre-se de que o resultado final é configurar ambientes de desenvolvimento de aplicativos multifuncionais para acelerar a priorização de negócios, a resolução de problemas e a tomada de decisões, além de cumprir os objetivos de tempo de obtenção de valor. Serão as principais propostas de valor da plataforma DevOps.

TORNE-O ÁGIL, FLEXÍVEL E RÁPIDO

Lembre-se de que agilidade, flexibilidade e velocidade são os catalisadores da competitividade e do valor comercial. Eles farão parte da plataforma DevOps.

THINK PAAS

Lembre-se de que a AWS fornece serviços variados para implementação de infraestrutura de CD de última geração.

Plataforma Startup Mindset

O objetivo é implementar estratégias DevOps focadas no *time-to-market* e no *time-to-value*. É adaptado para startups de tecnologia. Para implementar a mentalidade de startup, os quatro princípios do DevOps para AWS são implantados da seguinte forma:

PENSE DE FORMA HOLÍSTICA E MULTIFUNCIONAL, NÃO APENAS EM TI

Os negócios relevantes e os recursos de TI são reunidos em uma equipe única e dedicada de desenvolvimento de serviços, apoiada por práticas ágeis apropriadas de Scrum e XP (Abdoulaye, 2016).

TORNE-O ÁGIL, FLEXÍVEL E RÁPIDO

Serviços PaaS, baseados em práticas ágeis e infraestrutura, como o código da AWS, são fornecidos para tornar os processos de entrega de aplicativos ágeis, flexíveis e rápidos (Abdoulaye, 2016).

PENSE EM PAAS

A infraestrutura AWS, como código, e as ferramentas DevOps são combinadas para implementar uma infraestrutura de CD na forma de uma plataforma PaaS (Abdoulaye, 2016).

EM FOCO

Acesse seu ambiente virtual de aprendizagem e confira a aula referente a este tema. **Recursos de mídia disponíveis no conteúdo digital do ambiente virtual de aprendizagem.**

NOVOS DESAFIOS

Como toda grande caminhada começa com um primeiro passo, precisamos entender os princípios dos padrões de projetos. No início da jornada, solicitamos uma pesquisa acerca dos princípios do *design patterns*.

A respeito dessas noções, podemos considerar os seguintes conceitos:

Abstração: princípio fundamental dos *design patterns*, que envolve a identificação e isolamento dos aspectos essenciais de um sistema, ou seja, por meio de interfaces ou classes abstratas, servindo de contrato para classes concretas, permitindo, dessa forma, que os aspectos sejam tratados de forma independente.

Encapsulamento: refere-se à ocultação dos detalhes de implementação de uma classe ou componente, expondo apenas uma interface consistente.

Reutilização: princípio que incentiva a utilização de soluções existentes para problemas comuns.

Modularidade: os padrões de projeto devem promover a modularidade, dividindo o sistema em componentes independentes que podem ser modificados, substituídos ou estendidos sem afetar outros componentes.

Flexibilidade: os padrões de projeto devem permitir que o sistema seja adaptável a mudanças de requisitos ou contexto, facilitando a extensão e manutenção do sistema.

Legibilidade: os padrões de projeto devem ser compreensíveis e legíveis, facilitando a comunicação entre os membros da equipe e a manutenção futura do código.

Eficiência: os padrões de projeto devem ser eficientes em termos de uso de recursos computacionais, garantindo um desempenho adequado do sistema.

Padrões reconhecíveis: os padrões de projeto devem ser reconhecíveis e conhecidos pelos desenvolvedores, facilitando a comunicação e a compreensão do design do sistema.

Tecnologias, como *cloud computing*, IOT, DevOps e microsserviços, estão em alta no mercado e são uma realidade em nossos ambientes de TI, cada vez mais, distribuídos e integrados às nossas vidas. Portanto, faz-se necessário o entendimento dessas tecnologias e respectivos padrões de utilização. Trata-se de temas importantes para o aprofundamento dos seus conhecimentos, pois profissionais especialistas, nessas áreas, são altamente requisitados no mercado de trabalho.

Assim, você pode identificar o quão importante é ter o conhecimento das tecnologias mencionadas e se aprofundar nelas.

1. "Chamadas de rede também podem exigir configuração significativa para conexão, autenticação e autorização. Se essas chamadas forem usadas em vários aplicativos, criados, usando vários idiomas e estruturas, as chamadas deverão ser configuradas para cada uma dessas instâncias. Além disso, funcionalidade de segurança de rede poderá precisar ser gerenciada por uma equipe central na sua organização. Com uma base de código grande, pode ser arriscado para essa equipe atualizar código de aplicativo com a qual não esteja familiarizada" (Microsoft Learn, 2024, on-line).

   Referente a esse assunto, a qual das opções, a seguir, o texto se refere? Assinale a alternativa correta:

   a) *Single responsibility principle*.
   b) *Microservice chassis*.
   c) Padrão embaixador.
   d) *Circuit breaker*.
   e) *Data replication*.

2. O padrão de arquitetura de microsserviços é definido para que a aplicação seja, cada vez mais, escalável e cada parte consiga operar de forma independente. Cada microsserviço é uma parte do todo e atua de forma independente, tornando o processo de manutenção mais ágil e inteligente.

   Acerca dos padrões de microsserviços, analise as afirmativas a seguir:

   I - *Single responsibility principle*: cada microsserviço deve ter uma única responsabilidade e não deve ser misturado com outras responsabilidades.
   II - API *gateway*: um ponto único de entrada para todas as chamadas de serviço, o que facilita a gestão de autenticação, autorização e roteamento.
   III - *Circuit breaker*: usando um mecanismo para lidar com falhas de serviço, o que ajuda a evitar cascatas de falha.

   É correto o que se afirma em:

   a) I, apenas.
   b) III, apenas.
   c) I e II, apenas.
   d) II e III, apenas.
   e) I, II e III.

3. A necessidade de padrões de design sem servidor eficazes aumenta à medida que mais usuários adotam a arquitetura sem servidor. Os padrões de projeto são soluções reutilizáveis para problemas comuns. Eles podem lhe orientar na criação da melhor arquitetura para aplicativos sem servidor eficientes, escaláveis e confiáveis (Guide..., 2023).

   A respeito do conceito *servless*, é correto afirmar:

   a) A adoção da arquitetura sem servidor reduz a necessidade de padrões de design.
   b) Os padrões de design sem servidor são exclusivamente soluções únicas para problemas específicos.
   c) A arquitetura sem servidor não requer consideração de padrões de design.
   d) Os padrões de design sem servidor são uma solução temporária e não reutilizável.
   e) A necessidade de padrões de design sem servidor eficazes aumenta à medida que mais usuários adotam a arquitetura sem servidor.

REFERÊNCIAS

ABDOULAYE, P. 3 AWS Design Patterns to Maximize DevOps Value. **DevOps.com**, [*s. l.*], 11 oct. 2016. Disponível em: https://devops.com/aws-design-patterns-maximize-devops/. Acesso em: 26 jun. 2024.

GUIDE to serverless architecture design patterns. **Trend Micro**, Cork, 8 jun. 2023. Disponível em: https://www.trendmicro.com/en_ie/devops/23/f/serverless-architecture-design-patterns-guide.html. Acesso em: 26 jun. 2024.

IOT ATLAS. **Patterns**. 2024. Patterns. Disponível em: https://iotatlas.net/en/patterns/. Acesso em: 26 jun. 2024.

KOST, M. Design patterns for the internet of things. **Arm Community**, Cambridge, 27 may 2014. Disponível em: https://community.arm.com/arm-community-blogs/b/internet-of-things-blog/posts/design-patterns-for-an-internet-of-things. Acesso em: 26 jun. 2024.

MICROSOFT LEARN. **Padrão embaixador**. c2024. Centro de arquitetura. Disponível em: https://learn.microsoft.com/pt-br/azure/architecture/patterns/ambassador. Acesso em: 26 jun. 2024.

PRINCIPAIS padrões de arquitetura de *microservices*. **Papo.dev**, [*s. l.*], 23 fev. 2022. Disponível em: https://papo.dev/posts/principais-padroes-de-arquitetura-de-microservices. Acesso em: Acesso em: 26 jun. 2024.

GABARITO

1. Alternativa C.

   A opção C está correta, pois o padrão embaixador isola os clientes do serviço de comunicação. As demais opções estão incorretas, pois se tratam de padrões de microsserviços.

2. Alternativa E.

   Todas as afirmações estão corretas.
   A afirmação I está correta, pois *Single Responsibility Principle* é um padrão de microsserviços, que preconiza a responsabilidade única.
   A afirmação II está correta, pois API *Gateway* é um padrão de microsserviços, que preconiza um ponto único de entrada para as chamadas de serviço.
   A afirmação III está correta, pois *Circuit Breaker* é um padrão de microsserviços, que define um mecanismo para lidar com a falha de serviço.

3. Alternativa E.

   A alternativa A está incorreta, pois a introdução da arquitetura sem servidor não elimina a importância dos padrões de design. Pelo contrário, a complexidade dos aplicativos sem servidor pode aumentar a necessidade de padrões para garantir eficiência, escalabilidade e confiabilidade.
   A alternativa B está incorreta, pois os padrões de design são soluções reutilizáveis para problemas comuns. Eles não são exclusivos para arquitetura sem servidor; são aplicáveis em várias outras áreas de design de software.
   A alternativa C está incorreta, pois, mesmo na arquitetura sem servidor, é crucial considerar padrões de design para garantir eficiência, escalabilidade e confiabilidade dos aplicativos.
   A alternativa D está incorreta, pois os padrões de design são concebidos como soluções reutilizáveis para problemas comuns.
   A alternativa E está correta, pois, com a crescente adoção da arquitetura sem servidor, há uma demanda crescente por padrões de design eficazes para garantir que os aplicativos sem servidor sejam eficientes, escaláveis e confiáveis.

MINHAS ANOTAÇÕES

MINHAS ANOTAÇÕES

TEMA DE APRENDIZAGEM 8

MELHORES PRÁTICAS E DICAS

MINHAS METAS

- Conhecer boas práticas de desenvolvimento.
- Entender a contribuição dos padrões de projeto no sucesso de uma proposta de desenvolvimento de software.
- Reconhecer a importância do código simples no desenvolvimento de softwares.
- Compreender a importância da refatoração nos sistemas de software.
- Entender que o desenvolvimento é uma área em que o aprendizado deve ser contínuo.
- Refletir acerca da relevância da comunicação, a qual continua sendo o maior trunfo de um projeto, e da padronização, a qual contribui para que ela ocorra.
- Compreender a importância da automação de testes.

INICIE SUA JORNADA

Ao entrar no mundo profissional, o estudante se depara com uma dificuldade latente para manter um código robusto e escalável que são os códigos confusos, de difícil manutenção ou que não seguem os padrões. Como, então, ele pode enfrentar esse cenário para se destacar como desenvolvedor? Você, em sua carreira, se deparará com o uso de padrões e boas práticas de programação, os quais lhe guiarão em soluções de contorno.

Para lhe auxiliar a visualizar o cenário e respectiva solução, propomos a seguinte situação problema: ao ser contratado por uma empresa de software conhecida no mercado, assim que suas atividades se iniciam, você se depara com um grande desafio, que é o código legado da organização.

Com o início do seu primeiro projeto, você percebe que o código está desorganizado, com poucos comentários, pouca descrição nos nomes das variáveis e falta de padrões de projeto claros. Isso torna a manutenção e a implementação de novas funcionalidades extremamente difíceis, impactando a produtividade da equipe. Pensando nesse contexto, escreva um parágrafo, estabelecendo qual é o problema central a ser enfrentado e como você resolveria essa questão.

Esse experimento lhe estimulará a refletir a respeito das condições de código encontradas nas soluções e a indagar-se acerca de questões, como: por que as boas práticas e padrões não são sempre observados durante o desenvolvimento?

PLAY NO CONHECIMENTO

Ouça nosso podcast acerca do tema *Padrões de Projeto e Testes Automatizados*. Acesse e atualize seus conhecimentos. **Recursos de mídia disponíveis no conteúdo digital do ambiente virtual de aprendizagem.**

VAMOS RECORDAR?

No vídeo do canal Sharpax, Marks Vinicius aborda as boas práticas para programação. Acesse e atualize-se!

Disponível em: **https://www.youtube.com/watch?v=7KuB-mZTdiE**.

DESENVOLVA SEU POTENCIAL

O desenvolvimento de software é uma atividade que pode se tornar muito complexa quando cuidados básicos, padrões reconhecidos ou atenção aos detalhes deixam de ser foco de atenção do desenvolvedor.

INDICAÇÃO DE FILME

Blackberry

Mike Lazaridis e Douglas Fregin estão prestes a criar o primeiro smartphone do mundo e a mudar a forma como as pessoas trabalham e se conectam. Entretanto, negócios duvidosos e, talvez o concorrente mais perigoso, o iPhone, ameaçam o incrível sucesso da empresa.

Refletindo sobre a história: Esse filme mostra a importância da inovação e constante atualização, chamando a atenção para o senso de boas práticas a serem seguidas. Ele também mostra que a não observância de alguns desses pontos, como a compreensão profunda dos requisitos dos usuários, pode ser determinante para o sucesso, ou não, de um projeto.

Alguns pontos, os quais abrangem, além de outros, a compreensão profunda dos requisitos, a qualidade e a comunicação clara e transparente, contribuem para a escolha de um padrão que seja adequado às situações de projeto.

Para a escolha de um padrão de projeto, devem ser considerados os seguintes tópicos, os quais serão detalhados mais adiante:

- entendimento dos requisitos;
- código limpo;
- padrões de projeto adequados para a situação;
- refatoração;
- comentários;
- revisão de código;
- ferramentas de análise estática;
- modularidade;
- aprendizado contínuo.

ENTENDIMENTO DOS REQUISITOS

Antes de iniciar qualquer projeto de software, é fundamental realizarmos um levantamento minucioso dos requisitos, pois é necessária uma compreensão clara a respeito das características e necessidades das pessoas que utilizarão o software, assim como dos requisitos que envolvem a sua solução.

PENSANDO JUNTOS

Muitos projetos desconsideram a importância dos detalhes nesse levantamento, o que acaba suscitando aquela observação por parte do usuário: não era bem isso que eu queria!

> Embora a atividade de comunicação forneça uma boa fundamentação para o entendimento, a análise de requisitos refina esse entendimento por meio de interpretações adicionais. À medida que a estrutura do problema é delineada como parte do modelo de requisitos, invariavelmente, surgem perguntas. São essas perguntas que preenchem as lacunas – ou, em alguns casos, realmente nos ajudam a encontrar as lacunas (Pressman; Maxim, 2016, p. 215).

CÓDIGO LIMPO

A criação de um código bem estruturado e identado, com nomes significativos não só de variáveis, mas também de classes, é uma boa prática na programação, pois simplifica o entendimento, manutenção e extensão de seu código, tornando os sistemas mais robustos e eficientes. Na escolha de um padrão de projetos, esse é um fator que deve ser preponderante. Por suas características, os padrões de projeto já trazem esse benefício, no entanto, trata-se de pontos que devem ser constantemente observados.

Pressman e Maxim (2016) aconselham que você, ao começar a escrever um código:

- Restrinja seus algoritmos, seguindo a prática de programação estruturada.
- Pense na possibilidade de usar programação em pares.
- Selecione estruturas de dados que atendam às necessidades do projeto.
- Domine a arquitetura de software e crie interfaces coerentes com ela.
- Mantenha a lógica condicional tão simples quanto possível.
- Crie *loops* agrupados de tal forma que testes sejam facilmente aplicáveis.
- Escolha denominações de variáveis significativas e obedeça a outros padrões de codificação locais.
- Escreva um código que se documente automaticamente.
- Crie uma disposição (layout) visual (por exemplo: recuos e linhas em branco), que auxilie a compreensão.

PADRÕES DE PROJETO ADEQUADOS PARA A SITUAÇÃO

Os padrões de projeto exercem um papel crucial no sucesso de um projeto de desenvolvimento de software e fornecem soluções comprovadas para problemas conhecidos. "Padrões de projeto são soluções típicas para problemas comuns em projeto de software. Eles são como plantas de obras pré-fabricadas, que você pode customizar para resolver um problema de projeto recorrente em seu código" (Shvets, 2021, p. 29).

Algumas situações ocorrem com frequência no ambiente de desenvolvimento. Por esse motivo, são importantes os padrões. Sendo assim, alguns dos pontos que merecem a atenção do desenvolvedor são:

CONSISTÊNCIA E CLAREZA

A consistência e a clareza do código permitem aos desenvolvedores a criação de um código mais fácil de realizar a manutenção, tornando a equipe menos suscetível a erros de código e facilitando a colaboração.

REUTILIZAÇÃO DE CÓDIGO

Ao permitir que o código seja encapsulado e reutilizado, o desenvolvedor economiza tempo de desenvolvimento e reduz a chance de introduzir erros.

FLEXIBILIDADE E ESCALABILIDADE

Com padrões de arquitetura, como o MVC (*Model-View-Controller*) ou outros, os sistemas se tornam mais flexíveis e escaláveis. Isso, porque os padrões de arquitetura promovem a separação de preocupações e modularização do código, tornando os sistemas mais flexíveis e escaláveis.

FACILIDADE DE MANUTENÇÃO

Projetos que seguem os *design patterns* são mais fáceis de manter a longo prazo, por conta da melhor organização e estruturação do código. Assim, a partir do momento em que novos desenvolvedores analisam o código, eles conseguem dar uma manutenção mais assertiva.

COMUNICAÇÃO

Os padrões de projeto também desempenham um papel importante na comunicação e entre membros da equipe, uma vez que os padrões são soluções documentadas e conhecidas pelos desenvolvedores, bem como usam terminologias comuns. Isso tudo reduz incertezas e mal-entendidos.

Seguindo essas diretrizes e mantendo um entendimento claro dos requisitos, problemas a serem resolvidos e características dos padrões de projeto disponíveis, você poderá escolher um parâmetro de maneira eficaz e construir sistemas de software robustos, flexíveis e de fácil manutenção.

REFATORAR

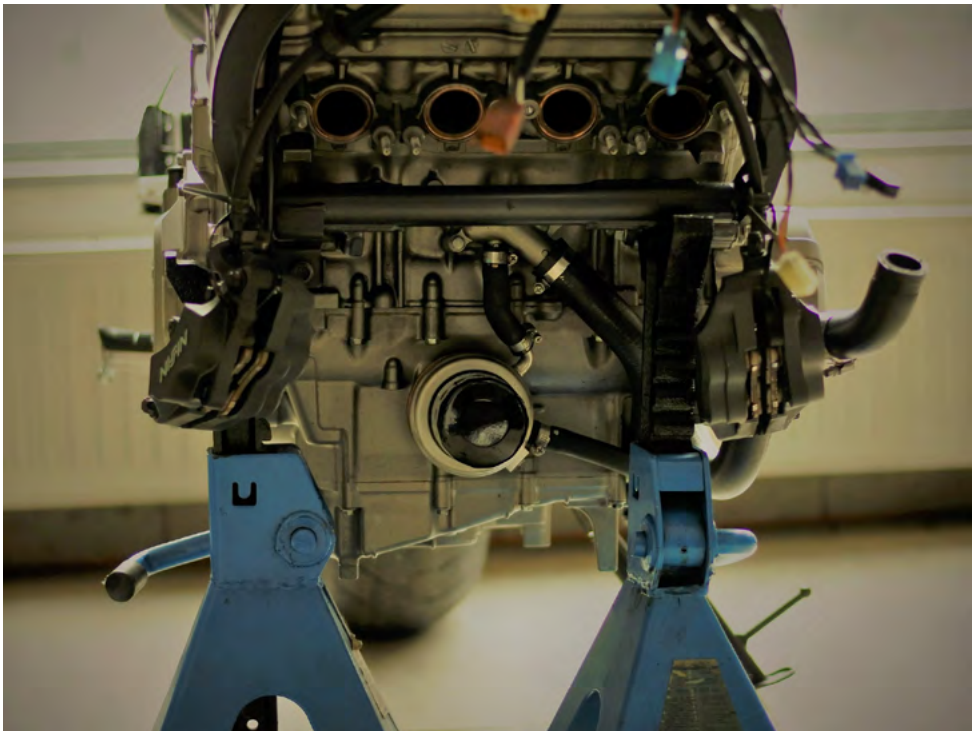
A prática de refatorar é fundamental ao desenvolvimento, pois identifica, constantemente, pontos a serem melhorados no código e padrões a serem aplicados. Segundo Pressman e Maxim (2016, p. 238):

> Quando um software é refatorado, o projeto existente é examinado em termos de redundância, elementos de projeto não utilizados, algoritmos ineficientes ou desnecessários, estruturas de dados mal construídas ou inadequadas ou qualquer outra falha de projeto que possa ser corrigida para produzir um projeto melhor.

Assim, a refatoração pode melhorar os quesitos:

LEGIBILIDADE
Simplifica o código, deixando-o mais legível para outros e para você mesmo em um futuro.
MANUTENIBILIDADE
Com a simplificação do código, facilita-se a sua manutenção.
REDUÇÃO DE BUGS
Ajuda a corrigir fontes de bugs antes que se tornem problemas maiores.
MELHORIA DE PERFORMANCE
Levando em conta o surgimento diário de novos recursos de programação, a refatoração pode melhorar a sua performance.

A prática de refatorar está ligada aos padrões de projeto, as quais visam melhorar a estrutura de código, que, ao refatorar o desenvolvedor, poderá identificar a necessidade da aplicação de padrões de projeto, trazendo, assim, os benefícios mencionados.



COMENTÁRIOS

Comentários no código são extremamente benéficos para a codificação e ambientação de novos desenvolvedores que venham a ingressar no projeto. Você pode explicar o trecho do código e como ele atuará. Em algumas passagens, os comentários se fazem necessários, no entanto, eles também devem ser usados com parcimônia, uma vez que, em excesso, podem poluir o código. Sendo assim, a correlação dos comentários de código com os padrões de projeto ajuda a promover contexto e explicação acerca de como e porque determinadas decisões de design foram tomadas.

REVISÕES DE CÓDIGO

Revisões no código são importantes. No momento em que outra pessoa analisa nosso código, ela identifica oportunidades de melhoria que não estavam tão claras aos nossos olhos, reconhecendo situações passíveis de se aplicar um padrão de projetos.

> Durante uma revisão, várias pessoas examinam o software e sua documentação associada, procurando problemas potenciais e não conformidades com os padrões. A equipe de revisão faz julgamentos informados sobre o nível de qualidade do software ou dos documentos do projeto (Sommerville, 2018, p. 674).

AUTOMATIZE TESTES

A automação dos testes trará uma porção de benefícios significativos para o desenvolvimento de software, otimizando tempo e erros detectados antes que o software seja entregue. Essa prática consiste em utilizar softwares para que roteiros definidos sejam executados e validados no software. "Testes de software têm a finalidade de assegurar que um programa atende às necessidades de seus usuários, como também servem para descobrir defeitos em seu funcionamento antes de disponibilizá-lo para uso" (Polo, 2020, p. 7).

Dentre as razões pelas quais a automação de testes é tão importante, encontram-se: economia de tempo, cobertura abrangente, melhoria na qualidade do software, redução de erros e falhas, facilidade de regressão, suporte à entrega contínua e DevOps.

Ao investir na automação dos testes, as equipes responsáveis podem acelerar o ciclo de desenvolvimento, reduzindo erros e consequentemente aumentando a satisfação do cliente.

FERRAMENTAS DE ANÁLISE ESTÁTICA

Essas ferramentas permitem ao desenvolvedor identificar problemas no código-fonte sem precisar executar o programa. Tais recursos examinam o código em busca de possíveis erros e violações de padrão de codificação. Sua utilização proporciona aspectos importantes, como:

IDENTIFICAÇÃO PRECOCE DE PROBLEMAS
Permite identificar o problema mais cedo, evitando a propagação de erros.
PADRONIZAÇÃO
Propicia o alinhamento de padrões, uma vez que sintaxe e código são avaliados.
MELHORIA DA QUALIDADE DO CÓDIGO
Cria um código mais limpo ao identificar erros ou inutilidades em sua estrutura.
SEGURANÇA
Identifica vulnerabilidades de segurança no código com a ajuda de ferramentas de análise estática.

MODULARIDADE

Ocorre quando um software é dividido em componentes, separadamente especificados e localizáveis, denominados módulos, os quais são integrados.

> Software monolítico (um grande programa composto de um único módulo) não pode ser facilmente entendido por um engenheiro de software. O número de caminhos de controle, abrangência de referências, número de variáveis e complexidade geral tornaria o entendimento quase impossível. Em quase todos os casos, devemos dividir o projeto em vários módulos para facilitar a compreensão e, consequentemente, reduzir o custo necessário para construir o software (Pressman; Maxim, 2016, p. 234).

Trata-se, portanto, de um ponto importante a ser pensado quando se escolhe um padrão de projeto.

APRENDA CONTINUAMENTE

A realização de cursos de capacitação em tecnologias emergentes e o acompanhamento do mercado são imprescindíveis para o aprendizado contínuo e para ilustrar seus conhecimentos às empresas, uma vez que certificações profissionais são importantes, pois atestam o conhecimento em determinada tecnologia.

EM FOCO

Acesse seu ambiente virtual de aprendizagem e confira a aula referente a este tema. **Recursos de mídia disponíveis no conteúdo digital do ambiente virtual de aprendizagem.**

NOVOS DESAFIOS

O desenvolvimento de software apresenta uma variedade desafios. Muitos deles podem ser contornados ou ter seu caminho facilitado, com o uso de padrões e boas práticas de desenvolvimento, áreas que estão continuamente sendo aprimoradas. A conexão entre teoria e prática no desenvolvimento de software, aliada às boas práticas e tendências tecnológicas, lhe preparará para enfrentar os desafios e aproveitar as oportunidades do mercado de trabalho de TI.

No início desta jornada, foram propostos uma situação-problema e um questionamento:

> Ao ser contratado por uma empresa de software conhecida no mercado, assim que suas atividades se iniciam, você se depara com um grande desafio: o código legado da empresa.
>
> Com o início do seu primeiro projeto, você percebe que o código está desorganizado, com poucos comentários, pouca descrição nos nomes das variáveis e falta de padrões de projeto claros. Isso torna a manutenção e a implementação de novas funcionalidades extremamente difíceis, impactando a produtividade da equipe.

A pergunta levantada acerca desse contexto foi: qual o problema central e como você resolveria essa questão? Pois bem, o problema central, nesse caso, é o código legado complexo e de baixa qualidade. Essa questão será resolvida por meio da refatoração das partes mais críticas do software.

Os pontos a refatorar dependem de cada aplicação do software em questão, mas em todos os códigos podem ser citados pontos críticos e passíveis de refatoração, quais sejam:

- Reduzir a complexidade do código, tornando-o mais simples.
- Encontrar códigos duplicados e eliminá-los.
- Eliminar códigos que não estão sendo utilizados.
- Verificar a má gestão de memória em cada objeto carregado.
- Verificar os pontos de lógica.
- Verificar as brechas de segurança.
- Analisar outros pontos que julgue necessário.

Podemos entender que as boas práticas de desenvolvimento contribuem de forma significativa em sua carreira, e seu aprofundamento se torna fundamental em sua evolução profissional.

1. "Você não pode apenas encontrar um padrão e copiá-lo para dentro do seu programa, como você faz com funções e bibliotecas que encontra por aí. O padrão não é um pedaço de código específico, mas um conceito geral para resolver um problema em particular. Você pode seguir os detalhes do padrão e implementar uma solução que se adeque às realidades do seu próprio programa" (Shvets, 2021, p. 29).

   A respeito dos padrões de projeto, marque a alternativa correta:

   a) Os padrões de design são soluções comprovadas para problemas específicos de design de software.
   b) Os padrões de design são apenas teorias sem aplicabilidade prática.
   c) Os padrões de design são exclusivamente voltados para problemas de hardware.
   d) Os padrões de design são uma moda passageira na indústria de software.
   e) Os padrões de design são soluções comprovadas para problemas específicos de hardware.

2. "Atualmente, agilidade se tornou a palavra da moda quando se descreve um processo de software moderno. Todo mundo é ágil. Uma equipe ágil é aquela rápida e capaz de responder de modo adequado às mudanças. Mudança tem tudo a ver com desenvolvimento de software. Mudança no software que está sendo criado, mudança nos membros da equipe, mudança devido a novas tecnologias, mudanças de todos os tipos que poderão ter um impacto no produto que está em construção ou no projeto que cria o produto" (Jacobson, 2002, p. 18, tradução nossa).

   No contexto do desenvolvimento ágil, a interação constante e o feedback rápido dos usuários são aspectos cruciais para o sucesso do projeto. Diante desse ponto, analise as afirmativas a seguir:

   I - O desenvolvimento ágil valoriza a entrega iterativa e incremental de funcionalidades.
   II - O desenvolvimento ágil prioriza a entrega de todo o escopo do projeto de uma vez.
   III - O desenvolvimento ágil desencoraja a colaboração entre os membros da equipe.

   É correto o que se afirma em:

   a) I, apenas.
   b) III, apenas.
   c) I e III, apenas.
   d) II e III, apenas.
   e) I, II e III.

3. "Quando um software é refatorado, o projeto existente é examinado em termos de redundância, elementos de projeto não utilizados, algoritmos ineficientes ou desnecessários, estruturas de dados mal construídas ou inadequadas ou qualquer outra falha de projeto que possa ser corrigida para produzir um projeto melhor" (Pressman; Maxim, 2016, p. 238).

   Com base no conceito de refatoração, marque a alternativa correta:

   a) A refatoração é um processo que visa reescrever completamente o código-fonte de um software, alterando sua estrutura e funcionalidades de forma radical.
   b) Refatoração é o processo de revisão do código-fonte de um software para torná-lo mais difícil de entender e manter.
   c) Refatoração é o processo de fazer pequenas melhorias incrementais no código-fonte de um software para melhorar sua legibilidade, eficiência e/ou manutenibilidade.
   d) A refatoração é um conceito obsoleto e desnecessário no desenvolvimento de software moderno.
   e) Refatoração é o processo de adicionar novos recursos e funcionalidades ao código-fonte de um software sem alterar sua estrutura ou funcionamento original.

REFERÊNCIAS

JACOBSON, I. A Resounding 'Yes' to Agile Processes – but also more. **Cutter IT Journal**, Boston, v. 15, n. 1, p. 18-24, 2002.

POLO, R. C. **Validação e teste de software**. São Paulo: Contentus, 2020. *E-book*.

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

SHVETS, A. **Mergulho nos padrões de projeto**. [*S. l.*: *s. n.*], 2021.

SOMMERVILLE, I. **Engenharia de software**. 10. ed. São Paulo: Pearson, 2018. *E-book*.

GABARITO

1. Alternativa A.

   Os padrões de design fornecem diretrizes e melhores práticas, que podem ser aplicadas para resolver problemas comuns de forma eficiente e comprovada.
   A alternativa B está incorreta, pois os padrões são amplamente utilizados na indústria de software e têm demonstrado sua eficácia na prática.
   A alternativa C está incorreta, pois os padrões de design são utilizados principalmente para resolver problemas de design de software, como arquitetura, estruturação de código, e interação entre componentes de software.
   A alternativa D está incorreta, pois os padrões de design são conceitos estabelecidos e amplamente reconhecidos que têm sido utilizados há décadas para melhorar a qualidade e a eficiência do desenvolvimento de software.
   A alternativa E está incorreta, pois os padrões de design são especificamente destinados a resolver problemas relacionados ao design de software, não de hardware.

2. Alternativa A.

   A afirmativa I é correta, pois o desenvolvimento ágil realmente valoriza a entrega iterativa e incremental de funcionalidades, permitindo que o software seja entregue em partes menores e mais frequentes, o que possibilita ajustes e melhorias contínuas.
   A afirmativa II é incorreta, pois o desenvolvimento ágil não prioriza a entrega de todo o escopo do projeto de uma vez. Pelo contrário, ele favorece a entrega de valor em incrementos menores ao longo do tempo, de acordo com as necessidades e prioridades do cliente.
   A afirmativa III é incorreta, pois o desenvolvimento ágil, na verdade, incentiva fortemente a colaboração entre os membros da equipe. A comunicação contínua e a colaboração são valores fundamentais do desenvolvimento ágil, que são essenciais para o sucesso do projeto.

GABARITO

3. Alternativa C.

   A refatoração é de fato o processo de fazer pequenas melhorias incrementais no código-fonte de um software para melhorar sua legibilidade, eficiência e/ou manutenibilidade. Isso geralmente envolve a reorganização do código, a eliminação de duplicações, a simplificação de estruturas complexas e a aplicação de padrões de design, entre outras técnicas.
   A alternativa A está incorreta, pois o objetivo da refatoração é realizar mudanças pequenas e incrementais para melhorar a estrutura e a qualidade do código, sem alterar seu comportamento externo.
   A alternativa B está incorreta, pois a refatoração deixa o código mais claro, legível e fácil de manter, eliminando duplicações, simplificando lógicas complexas e aplicando padrões de design.
   A alternativa D está incorreta, pois a refatoração não é um conceito obsoleto e desnecessário no desenvolvimento de software moderno; é uma prática amplamente reconhecida e recomendada para manter a qualidade do código ao longo do tempo, especialmente em ambientes ágeis.
   A alternativa E está incorreta, pois não é possível afirmar que refatoração seja o processo de adicionar novos recursos e funcionalidades ao código-fonte de um software sem alterar sua estrutura ou funcionamento original.

MINHAS ANOTAÇÕES

TEMA DE APRENDIZAGEM 9

COLABORAÇÃO E TRABALHO EM EQUIPE

MINHAS METAS

- Compreender a importância dos padrões de design na colaboração de um projeto.
- Entender a importância da colaboração em um projeto.
- Definir o modo como implantar os padrões de projeto em uma equipe de software.
- Definir ferramentas para colaboração em projetos de desenvolvimento.
- Aprender a integrar os padrões de design de forma eficaz no ciclo de vida do desenvolvimento de software.
- Nomear ferramentas de comunicação para implantação em ambientes de projetos de software.
- Entender o impacto dos padrões de projeto no trabalho em equipe de um projeto de software.

INICIE SUA JORNADA

Na indústria de tecnologia, em que a demanda por softwares é crescente, o trabalho em equipe se tornou um pilar muito importante na área de desenvolvimento. Surge, então, uma questão fundamental: como a colaboração e o trabalho em equipe impactam no desenvolvimento de software?

Esse questionamento é essencial para compreendermos os desafios e benefícios associados à colaboração no contexto do desenvolvimento de software. Isso ocorre, pois, em um projeto de software em que uma equipe multidisciplinar é montada, geralmente, cada pessoa fica responsável por uma área específica. Está cada vez mais raro o projeto em que uma pessoa atua sozinha, no entanto, em situações de pressão, ou quando surge alguma adversidade, você pode se sentir compelido a abordar o problema individualmente.

Com essa questão em mente, convido-lhe a realizar um experimento a respeito dos conceitos mencionados. Imagine que uma equipe de TI foi alocada em um projeto de desenvolvimento de software, trata-se de um sistema de comunicação, que inclui recursos para acompanhamento de tarefas, compartilhamento de documentos e comunicação interna. Cada membro da equipe ficou responsável por uma tarefa. Tais atribuições, porém, estavam todas interligadas: desenvolvedores precisavam colaborar com analistas e *testers*, analistas com usuários, e assim por diante. Sabe-se, também, que a empresa em questão possui a cultura de usar padrões de projeto em seus desenvolvimentos.

Diante do cenário montado, escreva de que forma os padrões de projeto poderiam impactar na colaboração entre os membros da equipe. Apresente sua ideia em forma de parágrafo, de cinco a dez linhas, como se fosse uma anotação para você mesmo. Você consegue perceber as diferenças de trabalhar individualmente e em equipe? Que tipos de desafios são enfrentados e que tipos de lições são aprendidas quando lidamos com a colaboração? Ao refletirmos acerca disso, você terá insights valiosos.

PLAY NO CONHECIMENTO

Ouça nosso podcast acerca do tema *Padrões de Design e Desenvolvimento Ágil*. Acesse e atualize seus conhecimentos. **Recursos de mídia disponíveis no conteúdo digital do ambiente virtual de aprendizagem.**

VAMOS RECORDAR?

O vídeo do canal Front Beginners discute a questão do trabalho em equipe na vida de um programador. Muitas vezes, a rotina desse profissional aparenta ser solitária, no entanto, o trabalho em equipe é essencial nessa área e será o padrão dessa profissão. Acesse e saiba mais!

Disponível em: **https://www.youtube.com/watch?v=xGtz78iI8Nk**.

DESENVOLVA SEU POTENCIAL

COMO INTEGRAR PADRÕES DE DESIGN EM EQUIPES DE DESENVOLVIMENTO

Os padrões de design ocupam um lugar de destaque no desenvolvimento de software, fornecendo soluções comprovadas para problemas recorrentes. A aplicação desses parâmetros, em times de projetos, é fundamental para manter a equipe coesa.

> A adoção de padrões de projetos tem como finalidade aumentar o entendimento do software, sendo considerada uma parte da documentação do sistema. O uso de padrões tem como intuito a produção de sistemas eficientes com características de reusabilidade, extensibilidade e manutenibilidade, fornecendo uma solução completa e melhor estruturada ao invés de uma solução imediata (Sturm; Silva, 2014, p. 1).

É comum encontrar problemas durante o processo de desenvolvimento de softwares. Nesses momentos de dificuldade, Segundo Pressman e Maxim (2016, p. 347), todos já pensamos:

> [...] será que alguém já desenvolveu uma solução para esse problema? A resposta é quase sempre sim. O problema é encontrar a solução; garantir que, de fato, adapte-se ao problema em questão;

> entender as restrições que talvez limitem a maneira pela qual a solução é aplicada e, por fim, traduzir a solução proposta para seu ambiente de projeto.

Vamos trabalhar, agora, algumas práticas recomendadas para integrar padrões de design em equipes de desenvolvimento.

Educação e conscientização

É essencial que toda a equipe de desenvolvimento tenha uma compreensão sólida dos princípios e conceitos acerca dos padrões de design. Por meio da aplicação de workshops e treinamentos, consegue-se impulsionar esse conhecimento, que, certamente, será benéfico, já que, além da aplicação individual das técnicas, o time se comunicará melhor, uma vez que todos saberão e compartilharão dos mesmos conhecimentos.

> Durante o desenvolvimento de software, a equipe procura aplicar técnicas que melhorem a qualidade do código, deixando-o legível, organizado e de fácil manutenção. Como resultado, há grandes chances de se obter um sistema livre de defeitos e muito próximo do desejado (Sturm; Silva, 2014, p. 3).

Padronização de nomenclatura e documentação

É importante que a empresa aplique padrões de nomenclatura e documentação, que sejam comuns a todos. Isso inclui nomes de classes, métodos, variáveis e situações, aos quais o respectivo padrão seja recomendado dentro da organização.

Revisões de código e colaboração

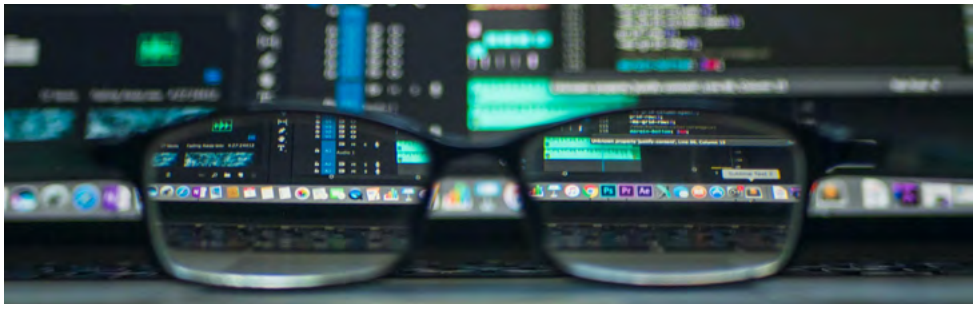
As revisões de código proporcionam uma ótima oportunidade para garantir que os padrões de design sejam aplicados corretamente e de maneira consistente. Sendo assim, ao revisar o código, os membros da equipe identificarão possíveis violações dos padrões de design.

Com a adoção de metodologias ágeis, que preconiza a aproximação e colaboração da equipe, é essencial que os membros do time compartilhem problemas e soluções de seus códigos. Nesse ponto, os padrões de projeto terão grande importância na comunicação.

INDICAÇÃO DE LIVRO

Use a Cabeça!

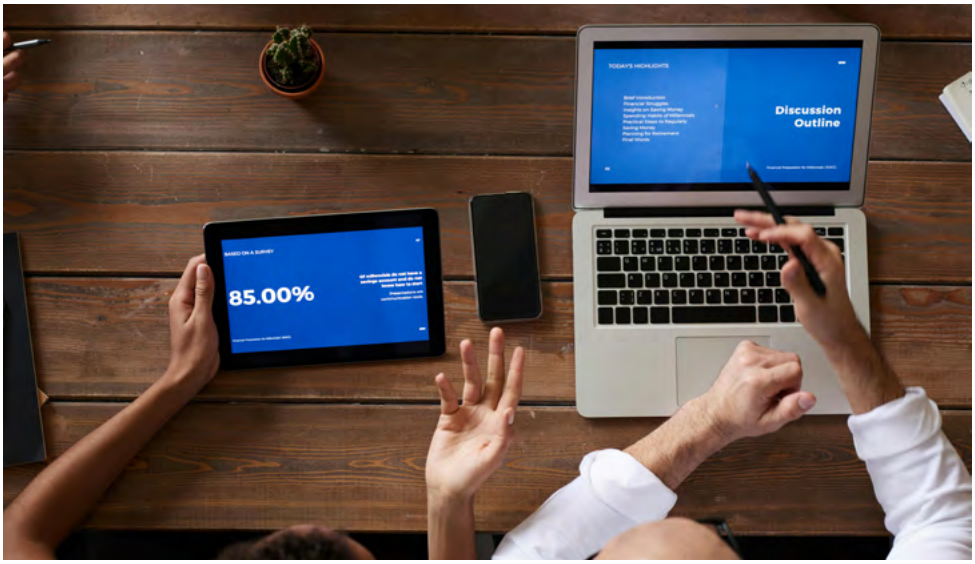
O livro traz um compilado de informações acerca das lições aprendidas por aqueles que se depararam com os mesmos problemas de desenvolvimento de software. Apresentada de maneira leve e muito informativa, a publicação é recomendada para a fixação dos conhecimentos a respeito dos padrões de projeto: ele associa a teoria com exemplos práticos por meio de analogias divertidas.



Utilização de ferramentas de análise estática

Ferramentas, como *linters* e *static code analyzers*, podem ser utilizadas para identificar, de forma automática, possíveis violações dos padrões de design no código-fonte dos projetos. Esses recursos podem fornecer sugestões e recomendações para corrigir violações dos padrões de design, facilitando a conformidade com as diretrizes da equipe.

Seguindo tais práticas, as equipes poderão colher os benefícios de uma abordagem centrada em padrões de design, incluindo código modular, de fácil manutenção, escalável e reutilizável, além de melhorar a fluidez da comunicação da equipe no projeto.



APROFUNDANDO

Atualmente, existem várias ferramentas de detecção de vulnerabilidades por análise estática disponíveis na internet. No entanto, elas estão vocacionadas a encontrar certos tipos específicos de vulnerabilidades e utilizam diversas técnicas para fazer a detecção (por exemplo, análise sintática, provadores de teoremas etc.). Por isso, é difícil a comparação das capacidades efetivas de cada ferramenta, tornando impossível a escolha entre elas (Teixeira; Antunes; Neves, 2007).

Para consolidar o conhecimento, resumimos as boas práticas a seguir:

EDUCAÇÃO E CONSCIENTIZAÇÃO

Toda a equipe deve ter uma compreensão sólida dos princípios e conceitos dos padrões de design.

PADRONIZAÇÃO DE NOMENCLATURA E DOCUMENTAÇÃO

É importante que a empresa aplique padrões de nomenclatura e documentação, que sejam comuns a todos.

REVISÕES DE CÓDIGO E COLABORAÇÃO

As revisões de código proporcionam uma ótima oportunidade para garantir que os padrões de design sejam aplicados.

UTILIZAÇÃO DE FERRAMENTAS DE ANÁLISE ESTÁTICA

São recursos que podem fornecer sugestões e recomendações para a correção de violações dos padrões de design.

COLABORAÇÃO EM DESIGN: FERRAMENTAS E METODOLOGIAS

Um projeto de desenvolvimento de software é um projeto que requer a participação de várias pessoas com diferentes habilidades, montando uma equipe interdisciplinar, portanto, para garantir o sucesso de um projeto de software, é essencial que tenhamos uma colaboração de todo o time. Diante disso, ferramentas de colaboração se fazem necessárias.

Prototipagem e design de interface de usuário (UI/UX)

Essa categoria de ferramentas é fundamental para envolver membros da equipe de design do projeto, ferramentas, como Adobe XD, Figma e Sketch, permitem a criação de protótipos interativos, fluxos de usuário e *wireframes*, que posteriormente servirão de suporte para o desenvolvimento.

Versionamento

Em qualquer artefato construído para o projeto, é essencial termos o versionamento. Nesse ponto, o software Figma é um ótimo representante, pois permite acompanhar evoluções, ao longo do tempo, e compará-las com versões anteriores.

Também temos o controle de código, o qual pode ser extremamente benéfico para a equipe. Nesse sentido, o Git é um dos representantes dessa categoria e possui muitas funcionalidades de acompanhamento de versões e comparações. Nesse software, os desenvolvedores podem criar ramificações para trabalhar em novas funcionalidades ou correções de bugs sem interferir no código principal. Uma vez concluídas as alterações, essas ramificações podem ser mescladas de volta ao código principal de forma suave e controlada.

Há dois tipos principais de controle de versão:

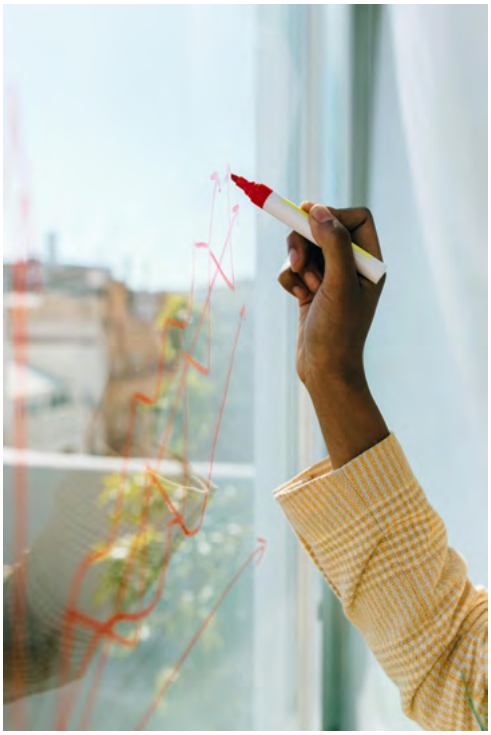
Centralizado: nesse sistema, há um único repositório de código-fonte, sendo que os desenvolvedores precisam se comunicar com essa coleção para realizar as alterações.

Sistema distribuído: cada desenvolvedor possui uma cópia completa do repositório em seu próprio computador, o que oferece mais flexibilidade e independência.

Desenvolver um software significa criar e modificar diferentes tipos de artefatos. De acordo com Oliveira (2005, p. 7), "o documento de especificação de requisitos, os modelos de análise e projeto, o código-fonte e os esquemas de banco de dados são exemplos desses artefatos".

Metodologias ágeis

Metodologias ágeis como Kanban ou Scrum contribuem com a colaboração entre os membros da equipe, sendo essenciais para a organização das atividades a serem desempenhadas pelo time. Algumas ferramentas, como o Trello, auxiliam na comunicação e organização das histórias de usuário a serem desenvolvidas.



Diagramas de modelagem

Os diagramas de modelagem são primordiais ao desenvolvimento, pois, ao criar um diagrama de modelagem, identificamos como será a arquitetura da aplicação, banco de dados, interação de classes e outros, possibilitando até antecipar erros de modelagem que só podem aparecer depois, um ótimo software a ser utilizado para a modelagem é o Draw.io.

A modelagem é importante para qualquer planejamento. Para o desenvolvimento de software bem-estruturado, ela é imprescindível. Com uma modelagem adequada, podemos antecipar visualizações a respeito de como um software será composto e quais serão suas interações. Além disso, podemos antecipar erros antes da codificação ou criação de um banco de dados.

Existem vários tipos de modelagem, cada um com seu propósito específico. Alguns dos mais comuns são:

DIAGRAMA DE ENTIDADE X RELACIONAMENTO
Diagrama de banco de dados que mostra um relacionamento entre as tabelas.
DIAGRAMA DE CASO DE USO
Representa as interações entre usuários e sistema, levando em conta o caso de uso em que se atua.
DIAGRAMA DE CLASSES
Demonstra a dependência entre as classes de um sistema; aqui, os *design patterns* atuam fortemente.
DIAGRAMAS DE SEQUÊNCIA
São os diagramas que mostram como os objetos em um sistema interagem.
DIAGRAMAS DE ATIVIDADE
Demonstra o sistema do ponto de vista das execuções das atividades e ilustra o fluxo de controle.
OUTROS DIAGRAMAS UML
A UML possui mais de uma dezena de diagramas para auxiliar na visualização do sistema.

Comunicação em tempo real

Atualmente, com uma parte da equipe trabalhando em home office e outra no escritório, é necessária uma comunicação em tempo real. Para tanto, ferramentas de colaboração, como Teams ou Slack, auxiliam a integração da equipe, facilitando as reuniões e o diálogo em grupo ou individuais.

PROTOTIPAGEM E DESIGN DE INTERFACE DE USUÁRIO (UI/UX)

Categoria de ferramentas fundamental para envolver membros da equipe e facilitar a comunicação.

VERSIONAMENTO

Controle de versões, fundamental para qualquer artefato do projeto.

METODOLOGIAS ÁGEIS

Essenciais para a organização das atividades a serem desempenhadas pelo time, flexibilizando todo o processo.

DIAGRAMAS DE MODELAGEM

Primordiais para identificação da arquitetura e planejamento.

COMUNICAÇÃO EM TEMPO REAL

Ferramentas essenciais para comunicação, aproximando a equipe distante geograficamente.

EM FOCO

Acesse seu ambiente virtual de aprendizagem e confira a aula referente a este tema. **Recursos de mídia disponíveis no conteúdo digital do ambiente virtual de aprendizagem.**

NOVOS DESAFIOS

No mercado de trabalho, cada vez mais, fazem-se necessários a colaboração e trabalho em equipe, com a integração de padrões de design ao ambiente de desenvolvimento. Esses pontos podem ser impulsionados, e mais facilmente alcançados, uma vez que, como vimos, os padrões de projeto trazem padronização, facilitando a comunicação entre a equipe.

Passamos por tópicos, como a integração de padrões em equipes de desenvolvimento, ressaltando como tais parâmetros são integrados e auxiliam as equipes, além de assuntos, como a conscientização e a educação da equipe no uso de padrões, nomenclatura ou revisões de código poderão auxiliar no desenvolvimento do software e integração do time.

Assuntos, como a colaboração em design, são fundamentais para facilitar a documentação, pois nos mostram, em formatos visuais, estruturas do sistema. No início do tema, propusemos o experimento de pesquisar a respeito do impacto que um padrão de projeto poderia causar na colaboração entre os membros da equipe. A resposta a esse questionamento indica que esse impacto pode ser positivo em alguns aspectos, como: comunicação facilitada, divisão clara de responsabilidades, reutilização de soluções comprovadas, flexibilidade e adaptabilidade.

Os padrões de desenvolvimento trazem esses benefícios para uma implantação, a qual, aliada a outros métodos gerenciais de comunicação, certamente, trará resultados positivos para seu projeto.

1. "A adoção de padrões de projetos tem como finalidade aumentar o entendimento do software, sendo considerada uma parte da documentação do sistema. O uso de padrões tem como intuito a produção de sistemas eficientes com características de reusabilidade, extensibilidade e manutenibilidade, fornecendo uma solução completa e melhor estruturada ao invés de uma solução imediata" (Sturm; Silva, 2014, p. 1).

   Referente a esse assunto, assinale a alternativa correta:

   a) É essencial que toda a equipe de desenvolvimento tenha uma compreensão sólida dos princípios e conceitos por trás dos padrões de design.
   b) Padrões de projeto são soluções únicas e exclusivas para problemas de desenvolvimento de software, não havendo necessidade de sua reutilização ou adaptação para diferentes contextos.
   c) Padrões de projeto são restritos a uma única linguagem de programação e não podem ser aplicados em diferentes ambientes de desenvolvimento.
   d) Padrões de projeto são obsoletos e não oferecem benefícios significativos no desenvolvimento de software moderno, sendo mais vantajoso criar soluções personalizadas em cada projeto.
   e) Padrões de projeto são apenas diretrizes teóricas sem aplicação prática no desenvolvimento de software, sendo irrelevantes para a construção de sistemas reais.

2. "Durante o desenvolvimento de software, a equipe procura aplicar técnicas que melhorem a qualidade do código, deixando-o legível, organizado e de fácil manutenção. Como resultado, há grandes chances de se obter um sistema livre de defeitos e muito próximo do desejado" (Sturm; Silva, 2014, p. 3).

   Tendo como base as boas práticas de software, analise as afirmativas a seguir:

   I - Uma técnica recomendada é a indentação (recuo do texto, indicando a hierarquia no código).
   II - Os padrões de projeto contribuem para um código de qualidade.
   III - Os comentários no código auxiliam a compreendê-lo.

   Assinale a alternativa correta:

   a) I, apenas.
   b) III, apenas.
   c) I e II, apenas.
   d) II e III, apenas.
   e) I, II e III.

3. "Atualmente, existem várias ferramentas de detecção de vulnerabilidades por análise estática disponíveis na internet. No entanto, estas estão vocacionadas para encontrarem certos tipos específicos de vulnerabilidades e utilizam diversas técnicas para fazerem a detecção (por exemplo, análise sintática, provadores de teoremas etc.). Daí ser extremamente difícil a comparação das capacidades efetivas de cada ferramenta, tornando impossível a escolha entre elas" (Teixeira; Antunes; Neves, 2014, p. 2).

   Referente a esse assunto, assinale a alternativa correta:

   a) As ferramentas de análise estática são especializadas em encontrar vulnerabilidades específicas e empregam diferentes técnicas, como análise sintática e provadores de teoremas, para detectá-las.
   b) As ferramentas de análise estática são todas iguais e têm o mesmo conjunto de funcionalidades, tornando fácil a escolha entre elas.
   c) As ferramentas de análise estática são capazes de detectar todos os tipos de vulnerabilidades com a mesma eficácia, independentemente das técnicas utilizadas.
   d) As ferramentas de análise estática não são úteis na detecção de vulnerabilidades, pois são limitadas em suas capacidades e, muitas vezes, falham em encontrar problemas no código.
   e) Todas as ferramentas de análise estática têm as mesmas limitações e não oferecem nenhuma vantagem discernível em termos de detecção de vulnerabilidades.

REFERÊNCIAS

OLIVEIRA, H. L. R. **Odyssey-VCS**: uma abordagem de controle de versões para elementos da UML. 2005. Dissertação (Mestrado em Engenharia de Sistemas e Computação) – Programa de Pós-Graduação em Engenharia, Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2005.

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

STURM, J.; SILVA, M. P. da. Aplicação de padrões de projeto no desenvolvimento de software para a melhoria de qualidade e manutenibilidade. **Revista Retec**, Ourinhos, v. 5, n. 1, p. 41-50, 2014.

TEIXEIRA, E.; ANTUNES, J.; NEVES, N. F. Avaliação de ferramentas de análise estática de código para detecção de vulnerabilidades. *In*: CONFERÊNCIA NACIONAL SOBRE SEGURANÇA INFORMÁTICA NAS ORGANIZAÇÕES, 3., 2007, Coimbra. **Anais** [...]. Coimbra: Universidade de Coimbra, 2007.

GABARITO

1. Alternativa A.

   Os padrões de design são soluções comprovadas para problemas recorrentes no desenvolvimento de software. Uma compreensão sólida desses padrões permite que a equipe os aplique de maneira eficaz, melhorando a qualidade, a manutenibilidade e a escalabilidade do software.
   A alternativa B é incorreta, pois os padrões de projeto são soluções genéricas e flexíveis que podem ser adaptadas e reutilizadas em diferentes contextos e em diferentes linguagens de programação.
   A alternativa C é incorreta, pois os padrões de projeto são conceitos de design de software que podem ser aplicados em diferentes linguagens de programação e ambientes de desenvolvimento.
   A alternativa D é incorreta, pois os padrões de projeto continuam sendo uma parte essencial do desenvolvimento de software moderno.
   A alternativa E é incorreta, pois não é possível afirmar que padrões de projeto são apenas diretrizes teóricas sem aplicação prática no desenvolvimento de software, sendo irrelevantes para a construção de sistemas reais.

2. Alternativa E.

   Todas as afirmativas estão corretas.
   A afirmativa I é verdadeira, pois a indentação é uma prática comum e altamente recomendada na programação. Ela ajuda a melhorar a legibilidade do código, facilitando aos desenvolvedores o entendimento de sua estrutura e hierarquia.
   A afirmativa II é verdadeira, pois os padrões de projeto contribuem para um código de qualidade. Ao seguir os padrões de projeto apropriados, os desenvolvedores podem criar sistemas mais coesos, flexíveis e de fácil manutenção.
   A afirmativa III é verdadeira, pois os comentários no código auxiliam a compreendê-lo. Eles são úteis para explicar o propósito de partes específicas do código, fornecer insights acerca da lógica ou da intenção por trás de determinadas implementações, assim como alertar acerca de possíveis armadilhas ou considerações importantes.

3. Alternativa A.

   A alternativa A está correta, pois reflete a natureza das ferramentas de análise estática, destacando que elas são especializadas em encontrar vulnerabilidades específicas e empregam diferentes técnicas, o que dificulta a comparação de suas capacidades efetivas.
   A alternativa B está incorreta, pois as ferramentas de análise estática possuem conjuntos de funcionalidades diferentes e são especializadas em encontrar diferentes tipos de vulnerabilidades, tornando a escolha entre elas desafiadora.

GABARITO

A alternativa C está incorreta, pois as ferramentas de análise estática variam em sua capacidade de detectar diferentes tipos de vulnerabilidades e algumas podem ser mais eficazes em determinados cenários do que outras.
A alternativa D está incorreta, pois as ferramentas de análise estática são úteis na detecção de vulnerabilidades. Elas continuam sendo uma parte importante do processo de garantia de qualidade de software.
A alternativa E está incorreta, pois as ferramentas de análise estática variam em suas capacidades e abordagens, e algumas podem oferecer vantagens distintas em termos de detecção.

MINHAS ANOTAÇÕES

MINHAS ANOTAÇÕES