DESCREVER OS CONCEITOS GERAIS DE PADRÕES DE PROJETO, SEUS ELEMENTOS E SUAS CARACTERÍSTICAS

Motivação e origem dos padrões de projeto

Imagine que você foi alocado em um novo trabalho, ficando responsável por evoluir um sistema desenvolvido por outras pessoas ao longo de anos.

Você não se sentiria mais confortável se o software tivesse sido projetado com estruturas de solução que você já conhecesse?

Ao aprendermos a programar em uma linguagem orientada a objetos, por exemplo, somos capazes de definir classes, atributos, operações, interfaces e subclasses com essa linguagem. Esse conhecimento nos permite responder a perguntas operacionais como, por exemplo:

“Como eu escrevo a definição de uma classe nesta linguagem de programação?”.

Entretanto, quando temos que estruturar um sistema, a pergunta passa a ser bem mais complexa:

“Quais classes, interfaces, relacionamentos, atributos e operações devemos definir para resolvermos este problema de forma adequada?”.

Ao desenvolvermos um sistema, nos deparamos inúmeras vezes com essa pergunta. É comum um desenvolvedor iniciante se sentir perdido diante da diversidade de respostas possíveis, sendo que algumas podem facilitar, enquanto outras podem dificultar bastante as futuras evoluções do sistema.

Que tipo de conhecimento, então, um desenvolvedor experiente utiliza para estruturar um sistema?

Fazendo uma analogia com o jogo de xadrez, um desenvolvedor inexperiente guarda semelhanças com um jogador iniciante, que sabe colocar as peças no tabuleiro, conhece os movimentos permitidos para cada peça e domina alguns princípios elementares do jogo, como a importância relativa das peças, e o centro do tabuleiro.

Um desenvolvedor inexperiente conhece as construções básicas de programação (ex.: sequência, decisão, iteração, classes, objetos, atributos, operações) e alguns princípios elementares de estruturação de um sistema, tais como dividi-lo em módulos e evitar construir módulos longos e complexos.

O que diferencia os grandes enxadristas dos iniciantes?

Clique nas setas para ver o conteúdo.
Os grandes enxadristas estudaram ou experimentaram inúmeras situações de jogo e acumularam experiência sobre as consequências que determinado lance pode gerar no desenrolar da partida.

Quanto maior for o arsenal de situações conhecidas por um jogador, maiores as chances que ele terá de fazer uma boa jogada em pouco tempo.

É isso mesmo: os jogadores de xadrez também têm um prazo para cumprir. Existem campeonatos que definem um tempo máximo de duas horas para que cada jogador realize seus primeiros quarenta lances.

Um jogador iniciante não aprende apenas jogando e vivenciando essas situações. Ele pode estudar inúmeras situações catalogadas pelas quais outros jogadores já passaram.

Comentário

Existem catálogos de situações e estratégias para o **início**, para o **meio** e para a **finalização** de um jogo de xadrez.

As estratégias para o início de jogo são conhecidas como **aberturas**. Cada abertura tem um nome e corresponde a um conjunto de jogadas que geram consequências positivas e, eventualmente, negativas para cada lado (peças brancas ou pretas). Uma dessas aberturas é o nome de uma das séries originais mais vistas na história da Netflix: ***O gambito da rainha***.

E o que diferencia os desenvolvedores experientes dos iniciantes?

Resposta

Desenvolvedores experientes acumularam conhecimento sobre as consequências de resolver um problema aplicando determinada estrutura de solução, isto é, dividindo a solução em um conjunto específico de módulos e estabelecendo uma estrutura particular de interação entre eles. Essa experiência acumulada permite-lhes reusar, no presente, soluções que funcionaram bem no passado em problemas similares, produzindo sistemas flexíveis, elegantes e em menor tempo.

Muitos problemas em estruturação de software são recorrentes, então que tal registrá-los com as respectivas soluções para que, de forma análoga aos livros de abertura do xadrez, possamos compartilhar esse conhecimento com desenvolvedores que estejam encarando um problema análogo pela primeira vez?

Foi o que pensaram quatro engenheiros de software (Erich Gamma, Richard Helm, Ralph Johnson e John Vlissides) em 1994, quando publicaram o livro ***Design patterns: elements of reusable object-oriented software*** , descrevendo 23 padrões que eles utilizaram no desenvolvimento de diversos sistemas ao longo de suas carreiras. Os autores ficaram mais conhecidos como a "**gangue dos quatro**" (tradução de ***gang of four***), e os padrões publicados são conhecidos como os "**padrões GoF**". Desde então, diversas publicações surgiram relatando novos padrões, críticas a padrões existentes, padrões específicos para determinadas linguagens e paradigmas de programação, padrões arquiteturais e até mesmo antipadrões – construções que tipicamente trazem consequências negativas para a estrutura de um sistema.

O que é um padrão de projeto

Um padrão de projeto (***design pattern***) descreve um problema recorrente e a estrutura fundamental da sua solução definida por módulos e pelas comunicações entre eles.

Em uma estrutura de solução orientada a objetos, a solução é descrita por classes, interfaces e mecanismos de colaboração entre objetos.

Porém, o conceito de padrões de projeto é mais abrangente, uma vez que existem padrões voltados para outros paradigmas de programação, como, por exemplo, **programação funcional** e **programação reativa**.

Um padrão de projeto apresenta quatro elementos fundamentais:

NOME
O nome possibilita a comunicação entre os desenvolvedores em um nível mais abstrato. Em uma discussão de projeto, em vez de descrevermos todos os elementos participantes da solução para um problema, a forma de comunicação entre eles e as consequências do seu uso, basta mencionarmos o nome do padrão apropriado.

Um desenvolvedor experiente pode recomendar algo como: "use o Observer para resolver esse problema!", e isso será suficiente, pois conhecendo o padrão Observer, você já saberá como estruturar a solução. Portanto, os nomes dos padrões foram um vocabulário compartilhado pelos desenvolvedores, para agilizar a discussão de soluções de projeto.

PROBLEMA

Um padrão deve descrever as situações nas quais a sua utilização é apropriada, explicando o problema e o contexto. Os problemas podem ser específicos, como, por exemplo: “como podemos representar diferentes algoritmos com o mesmo propósito na forma de classes?” ou “como podemos instanciar uma classe específica, dentre várias similares, sem criar uma dependência rígida com a implementação escolhida?”.

Este elemento também pode descrever e discutir soluções inadequadas ou pouco flexíveis para o contexto apresentado, de forma que, se você estiver pensando em uma solução similar às descritas, o seu problema passará a ser: “ok, estou vendo que a estrutura que estava pensando em aplicar não é adequada. Como devo, então, estruturar a solução?”.

SOLUÇÃO

O padrão deve descrever os elementos recomendados para a implementação, suas responsabilidades, seus relacionamentos e colaborações. A solução é uma descrição abstrata que pode ser replicada em diferentes situações e sistemas. O desenvolvedor deve traduzir essa descrição abstrata em uma implementação específica no problema particular que ele estiver resolvendo. Note que a solução é uma ideia que precisa ser construída e implementada pelo desenvolvedor. Um catálogo de padrões de projeto é bem diferente de uma biblioteca de código, pois um padrão de projeto não tem o propósito de promover reutilização de código, mas sim de conhecimento, *know-how*.

CONSEQUÊNCIAS

Um padrão precisa discutir os custos e benefícios da solução proposta. Uma solução pode dar maior flexibilidade ao projeto, ao mesmo tempo em que pode trazer maior complexidade ou até mesmo resultar em problemas de performance. Antes de utilizarmos um padrão, devemos sempre avaliar os custos e benefícios em relação ao problema particular que estamos querendo resolver.

Vantagens e desvantagens do uso de padrões de projeto

Por que você deve conhecer padrões de projeto? Podemos enumerar alguns motivos:

Padrões facilitam o desenvolvimento, pois permitem a reutilização de soluções bem-sucedidas em problemas similares. Um padrão não pode ser simplesmente uma ideia; ao contrário, só pode ser considerado como tal se passar pela regra dos três: ele deve ter sido utilizado em pelo menos três diferentes situações ou aplicações.

Padrões possibilitam maior produtividade ao desenvolvimento, uma vez que não desperdiçamos tempo pensando, fazendo e refinando soluções até encontrarmos a mais adequada. Padrões nos fornecem um atalho para uma boa solução.

Padrões reforçam práticas de reutilização de código com estruturas que acomodam mudanças por meio do uso de mecanismos como delegação, composição e outras técnicas que não sejam baseadas em estruturas de herança.

Padrões fornecem uma linguagem comum para os desenvolvedores, agilizando a troca de ideias e experiências.

E quais são as desvantagens ou os pontos de atenção na utilização de padrões de projeto?

Desde que o conceito foi apresentado na década de 1990, surgiu uma grande quantidade de padrões. Portanto, a curva de aprendizado pode ser grande.

Decidir se um padrão pode ser empregado em um problema específico nem sempre é uma tarefa fácil e requer alguma experiência.

Às vezes é necessário adaptar ou mudar alguma característica da solução proposta por um padrão para tornar sua utilização adequada a um problema específico, o que também requer experiência.

É comum um iniciante achar que os padrões devem estar por toda a implementação e acabar fazendo uso inadequado deles.

Não existe consenso sobre a qualidade de todos os padrões. Singleton, por exemplo, é um padrão GoF que gera grande controvérsia, sendo considerado um antipadrão por muitos.

Principais padrões em linhas gerais

Os 23 padrões de projeto GoF são classificados em três grandes categorias:

Criação	Os padrões de projeto desta categoria têm como objetivo tornar a implementação independente da forma com que os objetos são criados, compostos ou representados. Os **cinco** padrões GoF desta categoria são: **Abstract Factory, Builder, Factory Method, Prototype e Singleton**.
Estrutura	Os padrões de projeto desta categoria tratam de formas de combinar classes e objetos para formar estruturas maiores. Os **sete** padrões GoF desta categoria são: **Adapter, Bridge, Composite, Decorator, Facade, Flyweight e Proxy**.
Comportamento	Os padrões de projeto da categoria comportamental tratam de algoritmos, da distribuição de responsabilidades pelos objetos e da comunicação entre eles. Os **onze** padrões GoF desta categoria são: **Chain of Responsibility, Command, Interpreter, Iterator, Mediator, Memento, Observer, State, Strategy, Template Method e Visitor**.

Saiba mais

Existem inúmeros padrões publicados para diferentes problemas: arquitetura e projeto detalhado de software, padrões específicos para uma linguagem de programação, testes, gerência de projeto, gerência de configuração, entre outros. Você não precisa estudar todos os padrões disponíveis, mas é importante saber que eles existem, onde estão descritos e conhecer os problemas que eles resolvem. Assim, no dia em que tiver

que resolver um problema similar, você poderá aproveitar uma solução utilizada com sucesso por outras pessoas.

Como achá-los? Experimente fazer uma busca na internet com termos gerais, tais como “pattern books”, “patterns catalog”, ou mais específicos, como “java design patterns”, “cloud architecture patterns”, “node js patterns”.