Padrão de projeto de software

Origem: Wikipédia, a enciclopédia livre.

Em Engenharia de Software, um **padrão de desenho** (português europeu) ou **padrão de projeto** (português brasileiro) (do inglês **design pattern**) é uma solução geral para um problema que ocorre com frequência dentro de um determinado contexto no projeto de software. Um padrão de projeto não é um projeto finalizado que pode ser diretamente transformado em código fonte ou de máquina, ele é uma descrição ou modelo (*template*) de como resolver um problema que pode ser usado em muitas situações diferentes. Padrões são melhores práticas formalizadas que o programador pode usar para resolver problemas comuns quando projetar uma aplicação ou sistema. Padrões de projeto orientados a objeto normalmente mostram relacionamentos e interações entre classes ou objetos, sem especificar as classes ou objetos da aplicação final que estão envolvidas. Padrões que implicam orientação a objetos ou estado mutável mais geral, não são tão aplicáveis em linguagens de programação funcional.

Padrões de projeto residem no domínio de módulos e interconexões. Em um nível mais alto há padrões arquiteturais que são maiores em escopo, usualmente descrevendo um padrão global seguido por um sistema inteiro.[1]

As características obrigatórias que devem ser atendidas por um padrão de projeto é composto basicamente por 4 (quatro) elementos que são:

- **Nome** do padrão;
- **Problema** a ser resolvido;
- **Solução** dada pelo padrão; e
- **Consequências**.

Os padrões de projeto:

- visam facilitar a reutilização de soluções de desenho - isto é, soluções na fase de projeto do software - e
- estabelecem um vocabulário comum de desenho, facilitando comunicação, documentação e aprendizado dos sistemas de *software*.

Índice

História

O arquiteto **Christopher Alexander**,[2][3][4] em seus livros (1977/1979) *Notes on the Synthesis of Form*, *The Timeless Way of Building* e *A Pattern Language*, estabelece que um padrão deve ter, idealmente, as seguintes características:

- **Encapsulamento:** um padrão encapsula um problema ou solução bem definida. Ele deve ser independente, específico e formulado de maneira a ficar claro onde ele se aplica.
- **Generalidade:** todo padrão deve permitir a construção de outras realizações a partir deste padrão.
- **Equilíbrio:** quando um padrão é utilizado em uma aplicação, o equilíbrio dá a razão, relacionada com cada uma das restrições envolvidas, para cada passo do projeto. Uma análise racional que envolva uma abstração de dados empíricos, uma observação da aplicação de padrões em artefatos tradicionais, uma série convincente de exemplos e uma análise de soluções ruins ou fracassadas pode ser a forma de encontrar este equilíbrio.
- **Abstração:** os padrões representam abstrações da experiência empírica ou do conhecimento cotidiano.
- **Abertura:** um padrão deve permitir a sua extensão para níveis mais baixos de detalhe.
- **Combinatoriedade:** os padrões são relacionados hierarquicamente. Padrões de alto nível podem ser compostos ou relacionados com padrões que endereçam problemas de nível mais baixo.

Além da definição das características de um padrão, Alexander definiu o formato que a descrição de um padrão deve ter. Ele estabeleceu que um padrão deve ser descrito em cinco partes:

1. **Nome:** uma descrição da solução, mais do que do problema ou do contexto;
2. **Exemplo:** uma ou mais figuras, diagramas ou descrições que ilustrem um protótipo de aplicação;
3. **Contexto:** a descrição das situações sob as quais o padrão se aplica;
4. **Problema:** uma descrição das forças e restrições envolvidos e como elas interagem;
5. **Solução:** relacionamentos estáticos e regras dinâmicas descrevendo como construir artefatos de acordo com o padrão, frequentemente citando variações e formas de ajustar a solução segundo as circunstâncias. Inclui referências a outras soluções e o relacionamento com outros padrões de nível mais baixo ou mais alto.

Os *patterns* de Alexander procuravam prover uma fonte de ideias provadas para indivíduos e comunidades para serem usadas em construções, mostrando assim o quanto belo, confortável e flexível os ambientes podem ser construídos.

Em 1987, a partir dos conceitos criados por Alexander, os programadores Kent Beck e Ward Cunningham propuseram os primeiros padrões de projeto para a área da ciência da computação. Em um trabalho para a conferência OOPSLA, eles apresentaram alguns padrões para a construção de aplicações comerciais em linguagem Smalltalk.[5] Nos anos seguintes Beck, Cunningham e outros seguiram com o desenvolvimento destas ideias.

Porém, o movimento ao redor de padrões de projeto só ganhou popularidade em 1995 quando foi publicado o livro *Design Patterns: Elements of Reusable Object-Oriented Software*. Os autores desse livro, Erich Gamma, Richard Helm, Ralph Johnson e John Vlissides, são conhecidos como a "Gangue dos Quatro" (Gang of Four) ou simplesmente "**GoF**".

Posteriormente, vários outros livros do estilo foram publicados, merecendo destaque *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development*, que introduziu um conjunto de padrões conhecidos como **GRASP** (*General Responsibility Assignment Software Patterns*).

Características de um padrão de projeto

Embora um padrão seja a descrição de um problema, de uma solução genérica e sua justificativa, isso não significa que qualquer solução conhecida para um problema possa constituir um padrão, pois existem características obrigatórias que devem ser atendidas pelos padrões:[3]

1. Devem possuir um NOME, que descreva o problema, as soluções e conseqüências. Um nome permite definir o vocabulário a ser utilizado pelos projetistas e desenvolvedores em um nível mais alto de abstração.

2. Todo padrão deve relatar de maneira clara a qual (is) PROBLEMA(s) ele deve ser aplicado, ou seja, quais são os problemas que quando inserido em um determinado contexto o padrão conseguirá resolve-lo.Alguns podendo exigir pré-condições.

3. Solução descreve os elementos que compõem o projeto, seus relacionamentos, responsabilidades e colaborações. Um padrão deve ser uma SOLUÇÃO concreta, ele deve ser exprimido em forma de gabarito (algoritmo) que, no entanto pode ser aplicado de maneiras diferentes.

4. Todo padrão deve relatar quais são as suas CONSEQUÊNCIAS para que possa ser analisada a solução alternativa de projetos e para a compreensão dos benefícios da aplicação do projeto.

Não pode ser considerado um padrão de projeto trecho de códigos específicos, mesmo que para o seu criador ele reflita um padrão, que soluciona um determinado problema, porque os padrões devem estar a um nível maior de abstração e não limitado a recursos de programação. Um padrão de projeto nomeia, abstrai e identifica os aspectos chaves de uma estrutura de projeto comum para torna-la útil para a criação de um projeto orientado a objetos reutilizável.[3]

Padrões GoF ('Gang of Four')

De acordo com o livro: "Padrões de Projeto: soluções reutilizáveis de software orientado a objetos", os padrões "GoF" são divididos em 23 tipos. Em função dessa grande quantidade de padrões, foi necessário classificá-los de acordo com as suas finalidades.

São 3 as classificações/famílias:

- **Padrões de criação**: Os padrões de criação são aqueles que abstraem e ou adiam o processo criação dos objetos. Eles ajudam a tornar um sistema independente de como seus objetos são criados, compostos e representados. Um padrão de criação de classe usa a herança para variar a classe que é instanciada, enquanto que um padrão de criação de objeto delegará a instanciação para outro objeto. Os padrões de criação tornam-se importantes à medida que os sistemas evoluem no sentido de dependerem mais da composição de objetos do que a herança de classes. O desenvolvimento baseado na composição de objetos possibilita que os objetos sejam compostos sem a necessidade de expor o seu interior como acontece na herança de classe, o que possibilita a definição do comportamento dinamicamente e a ênfase desloca-se da codificação de maneira rígida de um conjunto fixo de comportamentos, para a definição de um conjunto menor de comportamentos que podem ser compostos em qualquer número para definir comportamentos mais complexos. Há dois temas recorrentes nesses padrões. Primeiro todos encapsulam conhecimento sobre quais classes concretas são usadas pelo sistema. Segundo ocultam o modo como essas classes são criadas e montadas. Tudo que o sistema sabe no geral sobre os objetos é que suas classes são definidas por classes abstratas. Conseqüentemente, os padrões de criação dão muita flexibilidade no que é criado, quem cria, como e quando é criado. Eles permitem configurar um sistema com objetos “produto” que variam amplamente em estrutura e funcionalidade. A configuração pode ser estática (isto é, especificada em tempo de compilação) ou dinâmica (em tempo de execução).
- **Padrões estruturais**: Os padrões estruturais se preocupam com a forma como classes e objetos são compostos para formar estruturas maiores. Os de classes utilizam a herança para compor interfaces ou implementações, e os de objeto ao invés de compor interfaces ou implementações, eles descrevem maneiras de compor objetos para obter novas funcionalidades. A flexibilidade obtida pela composição de objetos provém da capacidade de mudar a composição em tempo de execução o que não é possível com a composição estática (herança de classes).
- **Padrões comportamentais**: Os padrões de comportamento se concentram nos algoritmos e atribuições de responsabilidades entre os objetos. Eles não descrevem apenas padrões de objetos ou de classes, mas também os padrões de comunicação entre os objetos. Os padrões comportamentais de classes utilizam a herança para distribuir o comportamento entre classes, e os padrões de comportamento de objeto utilizam a composição de objetos em contrapartida a herança. Alguns descrevem como grupos de objetos cooperam para a execução de uma tarefa que não poderia ser executada por um objeto sozinho.

Padrões "GoF" organizados nas suas 3 classificações/famílias:

Padrões de criação

- *Abstract Factory*
- *Builder*
- *Factory Method*
- *Prototype*
- *Singleton*

Padrões estruturais

- *Adapter*
- *Bridge*
- *Composite*
- *Decorator*
- *Façade* (ou *Facade*)
- *Business Delegate*
- *Flyweight*
- *Proxy*

Padrões comportamentais[3]

- *Chain of Responsibility*
- *Command*
- *Interpreter*
- *Iterator*
- *Mediator*
- *Memento*
- *Observer*
- *State*
- *Strategy*
- *Template Method*
- *Visitor*

Existem outros critérios de classificação dos padrões de projeto:

- **Finalidade:** reflete o que um padrão faz.

- **Escopo:** especifica se o padrão é aplicado à classe ou objeto.
  - Padrões com **escopo de classe**: utilizam a hierarquia para compor ou variar os objetos, mantendo a capacidade do sistema de se flexibilizar. Se realizarmos uma comparação com o outro tipo de classificação isso costuma acontecer quando se trata de um **padrão de criação**.
  - Padrões com **escopo de objeto**: encontrados no relacionamento entre os objetos definidos em tempo de execução. Realizando uma comparação com o outro tipo de classificação vemos que esses padrões aparecem muito nos **padrões estruturais e comportamentais**.

Padrões [6][7][8]

Os padrões GRASP, sigla para General Responsibility Assignment Software Patterns (or Principles), consistem de um conjunto de práticas para atribuição de responsabilidades a classes e objetos em projetos orientados a objeto.

Os padrões GRASP (General Responsibility Assignment Software Patterns), são responsáveis pela descrição de princípios de fundamental importância para a atribuição de responsabilidades em projetos orientados a objetos, oferecendo um melhor desempenho do código, e trabalhando acerca de solucionar problemas, garantindo melhor interface do projeto.

Sendo assim, é importante sabermos que a qualidade do projeto orientado a objetos está diretamente relacionada com a distribuição dessas obrigações, que promovem a não sobrecarga de objetos já que ocorre nesse processo a delegação de atividades, ou seja, cada objeto terá uma função específica, de modo que, o que ele não souber fazer será repassado para o objeto que está mais preparado para fazer.

Todos esses padrões servem para a resolução de problemas comuns e bastante típicos de desenvolvimento de software orientado a objeto. Portanto, tais técnicas apenas documentam e normatizam as práticas já consolidadas, testadas e conhecidas no mercado.

Os padrões GRASP estão mais como uma ferramenta mental ou uma filosofia de design, mas que ainda assim são úteis para o aprendizado e desenvolvimento de um bom design de software. Note que alguns padrões GoF implementam soluções correspondentes com padrões GRASP.

Tipos de Padrões Grasp

Dentro do GRASP podemos encontrar vários padrões relacionados aqueles de caráter básicos e avançados.

- Padrões Básicos:
  - Information Expert (ver *Especialista na Informação);*
  - Creator (ver Factory Method);
  - High Cohesion (ver Coesão);
  - Low Coupling(ver Acoplamento);
  - Controller(ver Model-view-controller).
- Padrões Avançados:
  - Polymorphism (ver *Polimorfismo*);
  - Pure Fabrication;
  - Indirection (ver *Indireção*);
  - Protected Variations (ver *Variações Protegidas*).

Information Expert

O que é?	Possíveis Problemas	Solução	Benefícios
É conhecido por ser o princípio fundamental para atribuir responsabilidade.	Qual é o princípio geral para a atribuição de responsabilidades aos objetos?	Atribua a responsabilidade ao especialista: a classe que tem as informações necessárias para assumir a responsabilidade.	▪ Encapsulamento é mantido; ▪ Fraco acoplamento (facilidade de manutenção); ▪ Alta coesão (objetos fazem tudo relacionado à sua própria informação).

Creator

O que é?	Possíveis Problemas	Solução	Benefícios
É uma das atividades mais comuns em um sistema de orientação a objetos. Atribui qual classe é responsável por criar os objetos.	Quem deve ser responsável por criar uma nova instância de uma classe?	Atribua à classe a responsabilidade de criar uma instância da outra classe.	Facilidade no desenvolvimento do projeto e compreensão.

High Cohesion

O que é?	Possíveis Problemas	Solução	Benefícios
É um padrão avaliativo que tenta manter os objetos adequadamente focado, gerenciado e compreensível.	Como manter a complexidade sob controle? Classes que fazem muitas tarefas não relacionadas são: ▪ Mais difíceis de entender; ▪ Mais difíceis de manter e de reusar; ▪ São mais vulneráveis à mudança.	Atribuir uma responsabilidade para que a coesão se mantenha alta.	▪ Melhor claridade e facilidade de compreensão do projeto; ▪ Simplificação da manutenção;

Low Coupling

O que é?	Possíveis Problemas	Solução	Benefícios
É um padrão de avaliação, que determina como atribuir responsabilidades para apoiar.	Como prover baixa dependência entre classes, reduzir o impacto de mudanças e obter alta reutilização?	Atribui as responsabilidades de modo que o acoplamento entre classes permaneça baixo.	▪ Menor dependência entre as classes; ▪ Mudar de uma classe que tem menor impacto sobre outras classes; ▪ Maior potencial de reutilização.

Controller

O que é?	Possíveis Problemas	Solução	Benefícios
O padrão controlador atribui a responsabilidade de lidar com os eventos do sistema para uma classe que representa a um cenário de caso de uso do sistema global.	Quem deve ser o responsável por lidar com um evento de uma interface de entrada?	Atribuir responsabilidades para receber ou lidar com um evento do sistema para: ▪ Uma classe que representa todo o sistema ou subsistema; ▪ Uma classe que representa cenário de caso de uso (controlador de caso de uso ou de sessão).	▪ Diminui a sensibilidade da camada de apresentação em relação à lógica de domínio; ▪ Oportunidade para controlar o estado do caso de uso;

Polymorphism

O que é?	Possíveis problemas	Soluções	Benefícios
É um princípio, no qual subclasses de uma superclasse são capazes de invocar métodos, que comportam-se de maneira diferente para cada classe derivada	Como tratar alternativas baseadas no tipo? Como criar componentes de software “plugáveis”?	É aconselhável atribuir responsabilidades aos tipos usando operações polimórficas, isso quando alternativas ou comportamento relacionados variam com o tipo(classe).	Facilita a manutenção e inserção de um novo tipo de classe

Pure Fabrication

O que é?	Possíveis Problemas	Solução	Benefícios
É uma classe que não representa um conceito no domínio do problema deve atender as seguintes características : baixo acoplamento e potencial de reutilização dos mesmos derivados.	Quando não se quer violar “Alta Coesão”e “Baixo Acoplamento” e o “Expert”não oferece soluções apropriadas que objeto deve ter responsabilidade?	Criar classes artificiais de baixa coesão e alto acoplamento(classes puras).	Apresenta vantagens como a criação de classes muito coesas e remove características não coesas das classes de domínios de negócios.

Indirection

O que é?	Possíveis Problemas	Solução	Benefícios
Suporta baixo acoplamento(e reusa potencial)entre dois elementos, atribuindo a responsabilidade de mediação entre eles a um objeto intermediário.	Como evitar acoplamento direto entre duas ou mais classes?Como evitar baixo acoplamento e manter alta possibilidade de reuso através do desacoplamento de objetos?	A um objeto intermediário que faz mediação entre componentes e serviços de modo que esses não sejam diretamente acoplados será atribuída a responsabilidade.	Visa atribuir responsabilidades, de modo que, olha para determinada responsabilidade determinando assim, as informações necessárias para cumpri-la e, consecutivamente, determina o local de armazenamento dessa informação.

Protected Variations

O que é?	Possíveis Problemas	Solução	Benefícios
É um princípio básico na motivação de grande parte dos mecanismos e padrões na programação e no projeto, para fornecer flexibilidade e proteção contra variações.	Como projetar objetos,subsistemas e sistemas, de modo que, as variações ou instabilidades nesses elementos não sejam responsáveis por impactos indesejáveis em outros elementos?	▪ Identificar pontos de variação e atribuir responsabilidades para criar uma interface estável em volta desses pontos; ▪ Evitar o envio de mensagens a objetos muito distantes.	Promove a contenção de variações e instabilidades, durante o projeto de objetos e sistemas, para que não ocorra a expansão de impactos a outros elementos.

Críticas

Segundo alguns usuários, alguns "padrões de projeto" são apenas evidências de que alguns recursos estão ausentes em uma determinada linguagem de programação (Java ou C++ por exemplo). Nesta linha, Peter Norvig demonstra que 16 dos 23 padrões do livro 'Design Patterns' são simplificados ou eliminados nas linguagens Lisp ou Dylan, usando os recursos diretos destas linguagens.[9]

Segundo outros, excessos nas tentativas de fazer o código se conformar aos 'Padrões de Projeto' aumentam desnecessariamente a sua complexidade.[10]

Bibliografia

- Christopher Alexander (1964). *Notes on the Synthesis of Form*. Estados Unidos: Harvard University Press. ISBN 0-674-62751-2
- Christopher Alexander (1979). *The Timeless Way of Building*. Estados Unidos: Oxford University Press. ISBN 0-19-502402-8
- Christopher Alexander (1977). *A Pattern Language*. Estados Unidos: Oxford University Press. ISBN 0-19-501919-9
- Craig Larman (2004). *Applying UML and Patterns*. An Introduction to Object-Oriented Analysis and Design and Iterative Development 1 ed. Estados Unidos: Prentice Hall. p. 736. ISBN 0-13-148906-2
- Gregor Hohpe, Bobby Woolf (2004). *Enterprise Integration Patterns*. Designing, Building, And Deploying Messaging Solutions 1 ed. Estados Unidos: Addinson-Wesley. p. 659. ISBN 0-321-20068-3
- Pablo Dall'Oglio (2007). *PHP Programando com Orientação a Objetos* (http://www.adianti.com.br/phpoo). Inclui Design Patterns 1 ed. São Paulo: Novatec. p. 576. ISBN 978-85-7522-137-2
- Alexandre Altair de Melo e Mauricio G. F. Nascimento (2007). *PHP Profissional* (http://www.novatec.com.br/livros/phppro). Aprenda a desenvolver sistemas profissionais orientados a objetos com padrões de projeto 1 ed. São Paulo: Novatec. p. 464. ISBN 978-85-7522-141-9

Referências

1. Martin, Robert C. «Princípios de Projeto e Padrões de Projeto» (http://www.objectmentor.com/resources/articles/Principles_and_Patterns.pdf) (PDF). Consultado em 2000 Verifique data em: `|acessodata=` (ajuda)
2. OS CONCEITOS de padrão de projeto foi criado na década de 70 pelo arquitecto Christopher Alexander.
3. Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides (1995). *Design Patterns*. Elements of Reusable Object-Oriented Software 1 ed. Estados Unidos: Addison-Wesley. ISBN 0-201-63361-2
4. Doug Lea. «Christopher Alexander:An Introduction for Object-Oriented Designers» (http://g.oswego.edu/dl/ca/ca/ca.html) (em inglês). Consultado em 18 de abril de 2007
5. Kent Beck, Ward Cunningham. «Using Pattern Languages for Object-Oriented Programs» (http://c2.com/doc/oopsla87.html) (em inglês). Consultado em 18 de abril de 2007
6. Santos, Ismael. «aulas/Modulo1_Intro_Grasp_GoF/Grasp/PadroesGRASP.pdf» (https://webserver2.tecgraf.puc-rio.br/~ismael/Cursos/Cidade_FPSW/aulas/Modulo1_Intro_Grasp_GoF/Grasp/PadroesGRASP.pdf) (PDF). tecgraf.puc. Consultado em 31 de março de 2017
7. Lima, Edirlei. «POO_Aula_03_Padroes_Projeto_GRASP_2015» (http://edirlei.3dgb.com.br/aulas/poo/POO_Aula_03_Padroes_Projeto_GRASP_2015.pdf) (PDF). edirlei.3dgb. Consultado em 30 de março de 2017
8. Murta, Leonardo. «Padrões Grasp» (http://www2.ic.uff.br/~leomurta/past/2008.2/es1/aula13.pdf) (PDF). ic.uff. Consultado em 30 de março de 2017
9. Norvig, Peter (1998). *Design Patterns in Dynamic Languages* (http://www.norvig.com/design-patterns/)
10. McConnell, Steve (2004). *Code Complete: A Practical Handbook of Software Construction, 2nd Edition*. [S.l.: s.n.] p. 105

Ligações externas

- O que são Design Patterns? (http://www.princiweb.com.br/blog/programacao/design-patterns/o-que-sao-design-patterns.html)
- Lista de alguns padroes comuns (http://www.mindspring.com/~mgrand/pattern_synopses2.htm) (em inglês)
- Página com descrições e exemplos dos padrões GoF (http://www.vincehuston.org/dp/) (em inglês)
- Jogo de Memorização Padrões de Projeto - GOF (http://www.questoex.com/games/gof/) (em português)
