

Padrões de Software (Software Patterns)

Cleudson de Souza - cdesouza@ufpa.br
Departamento de Informática
Universidade Federal do Pará



Agenda

- Definição
- Histórico
- Motivação
- Exemplo
 - Estratégia
 - MVC
- Forma de um Padrão
- Vantagens
- Categorias de Padrões
- O Catálogo de Gamma
- Exemplos
 - Padrões e Estratégias de Coad
 - O Padrão de Projeto State de Gamma
 - O padrão MVC
 - Idioma de Pree
- Anti-padrões
- Referências

Definição

- Padrões (...) são um modo de capturar experiências (...) de modo que outras pessoas possam utilizá-las [Gamma93].
- Padrões de projeto constituem um conjunto de regras descrevendo como realizar determinadas tarefas no desenvolvimento de software [Pree94].

Definição(2)

- Apesar das diferentes definições, existe um consenso de que padrões são, uma parcela de experiência destilada, descrita como uma solução para um problema em uma dada situação [Cam96].

Histórico

- Termo surgiu na década de 70 através do arquiteto Christopher Alexander, que encontrou temas recorrentes na arquitetura e os capturou em descrições e instruções chamadas padrões.
- Cada padrão descreve um problema que ocorre **repetidamente** em um ambiente, ele descreve o núcleo da solução do problema, de modo que você pode usar esta solução **inúmeras** vezes.

Histórico(2)

- Durante a década de 90, os projetistas de software descobriram a idéia de Alexander e tem aplicado-a no desenvolvimento de software.
- Padrões de projeto são um meio para representar, registrar e reutilizar micro-arquiteturas de projeto repetitivas, bem como a experiência acumulada por projetistas durante o desenvolvimento de software.

Histórico(3)

- Atualmente, padrões tem sido amplamente utilizados em todas as fases do processo de desenvolvimento de software, desde a análise até a implementação.
 - Padrões de análise;
 - Padrões de projeto;
 - Padrões de implementação - idiomas;
 - Padrões para hipertexto;
 - Padrões para colaboração - CSCW;
 - Etc.

Histórico Detalhado

- 87, Cunningham e Beck: padrões (patterns) para Smalltalk
- 91, Jim Coplien: Livro com Idiomas para C++
- 91, Bruce Andersen OOPSLA workshop
- 93, GoF: Livro sobre Padrões para Design
- 94: Primeira Conferência de Padrões PLoP
- 94 em diante: Padrões por toda parte !!!

Motivação

Apesar de todos os avanços na tecnologia de software, ou mesmo por causa destes avanços, um problema básico ainda existe:

a comunicação entre pessoas sobre as melhores práticas e soluções em Engenharia de Software

Motivação (2)

- O conhecimento de projetistas experientes
 - É intangível e muito valioso
 - Principal Fator para:
 - Sucesso: Gerentes e Técnicos Experientes
 - Fracasso: Gerentes e Técnicos Inexperientes
 - Adquirido lentamente, através de trabalho duro e persistente.
- Capturar, comunicar e assimilar este conhecimento é difícil

Observações

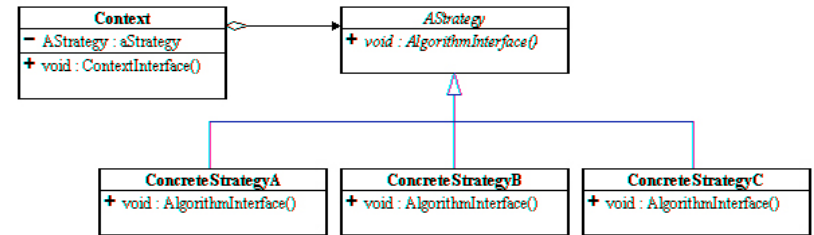
- Padrões **complementam** outras abordagens. Ou seja, eles podem ser utilizados em conjunto com diferentes abordagens para desenvolvimento de SW.
- Existem padrões para cada uma das fases do desenvolvimento: análise, projeto, implementação, testes, etc.
- **Padrões são identificados e não inventados.**

Observações

- Não se aplicam **apenas** à Orientação a Objetos. Por exemplo, podem existir padrões para análise estruturada, aspectos, etc.
- Existem padrões organizacionais que sugerem como organizar uma empresa de desenvolvimento de SW, etc [[Coplien, 1995](#)].

Exemplos

Exemplo: Padrão Estratégia (Strategy)



- Problema: é necessário mudar facilmente um algoritmo (em tempo de execução).
- A idéia básica deste padrão de projeto consiste em encapsular o algoritmo em uma classe isolada.

Exemplo: Padrão Estratégia (Strategy)

- Quando usar:
 - Diferentes variações de um mesmo algoritmo são necessárias;
 - Um algoritmo usa dados que os clientes não deveriam acessar;
 - Uma classe tem diferentes comportamentos, desta forma o padrão pode ser usado ao invés de várias condições;
- **AStrategy:** declara uma interface comum a todos os algoritmos.
- **ConcreteStrategy:** implementa o algoritmo usando a interface definida na superclasse Astrategy.
- **Context:** tem uma referencia para um objeto Astrategy, que é configurado com um objeto ConcreteStrategy.

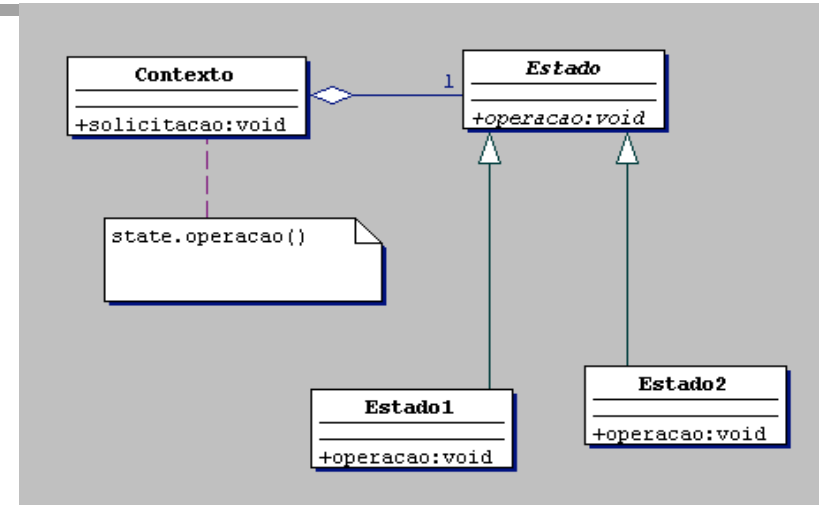
Exemplo: Padrão State

- O padrão de projeto *State* apresenta uma solução genérica para o problema de modelagem de objetos que apresentam diferentes estados ``lógicos`` e também comportamentos variados dependendo do estado lógico corrente.

Exemplo: Padrão *State*

- O padrão define uma hierarquia de estados paralela à hierarquia de classes e o objeto da aplicação delega as operações dependentes de estado para um objeto representando o seu estado corrente. Em tempo de execução o objeto da aplicação pode mudar seu estado lógico, implicando também numa alteração em seu comportamento.

Exemplo: Padrão *State*



Vantagens dos Padrões

- Constituem uma base de experiência reutilizável para a construção de software. Eles fornecem um modo de reutilizar o conhecimento de projetistas experientes, e com isso facilitam o treinamento de novos desenvolvedores;
- Formam um vocabulário comum para os projetistas se comunicarem e para a documentação e exploração das alternativas de projeto;

Vantagens dos Padrões (2)

- Atuam como "blocos de construção" que podem ser utilizados para criar aplicações mais complexas;
- Reduzem o tempo de aprendizado de bibliotecas de classes e frameworks. Uma vez que um usuário tenha aprendido uma biblioteca, ele pode reutilizar esta experiência para aprender novas bibliotecas;

Vantagens dos Padrões (3)

- Permitem larga reutilização de arquiteturas de software, mesmo que a reutilização de algoritmos, implementações, interfaces ou projetos detalhados não seja permitida;
- Reduzem a complexidade do sistema, pois nomeiam e identificam abstrações que estão em um nível de abstração acima das classes e instâncias;

Forma de um Padrão

- Um padrão apresenta, pelo menos, quatro elementos essenciais:
 - O Nome do padrão;
 - O Problema;
 - Solução; e
 - Conseqüências

Forma: Nome do padrão

- É usado para descrever um problema, sua solução e suas conseqüências. Deste modo, o vocabulário dos projetistas é ampliado, facilitando o desenvolvimento de software, uma vez que existirá um vocabulário comum que permitirá a troca de experiência entre projetistas através da comunicação das decisões de projeto, e facilitará a documentação do sistema.

Forma: O Problema

- Descreve quando aplicar o padrão. Explica o problema em questão e o contexto. Pode descrever problemas específicos ou genéricos. Normalmente, pelo menos dois exemplos são apresentados para cada padrão.

Forma: Solução

- Descreve os elementos que formam o padrão, seus relacionamentos, responsabilidades e documentação. A solução é apresentada para um problema genérico e uma estrutura geral.
- Dependendo da categoria do padrão, a solução é apresentada de maneira diferente.

Forma: Conseqüências

- São os resultados e decisões que a utilização daquele padrão acarretam. A descrição destas conseqüências é vital para uma avaliação dos custos e benefícios de um padrão e, conseqüentemente, para auxiliar a decisão sobre sua utilização. As conseqüências de um padrão incluem, por exemplo, seu impacto sobre a flexibilidade, portabilidade e expansibilidade do sistema.



Exemplos de Padrões

Exemplos de Padrões

- Software Orientado a Objetos (GoF)
- Organizações e Processos (James Coplien)
- Arquitetura de Software (Mary Shaw; e Frank Buschmann et. al.)
- Programação em Smalltalk (Kent Beck)
- Integridade de Informações; (Ward Cunningham)
- Programação em Java (*Doug Lea*)

Exemplos de Padrões (*cont.*)

- Patterns para (Análise) Orientação a Objetos (*Peter Coad*)
- Introdução de Novas Tecnologias em Organizações (*Linda Rising e outros*)
- Alexander e Qualidade de Software (*Richard Gabriel*)
- Patterns para Software para Comunicação em Redes (*Douglas Schmidt*)

Classificação de Padrões

- Padrões de software podem ser classificados segundo o domínio no qual se aplicam.
 - Assim, podem existir padrões genéricos, que podem ser aplicados em diferentes domínios;
 - Mas também podem existir padrões específicos para determinados domínios como por exemplo, tolerância a falhas, criptografia, telecomunicações, etc.

Classificação de Padrões

- Padrões de projeto possuem diferentes níveis de abstração e podem auxiliar em todas as fases do ciclo de vida de desenvolvimento. Segundo Buschmann [, 1996], os padrões podem ser agrupados em três categorias, a saber:
 - Padrões de Arquitetura;
 - Padrões de Análise e Projeto; e
 - Idiomas

Padrões de Arquitetura

Padrões de Arquitetura:

- Expressam um esquema da organização **global** da estrutura do sistema. Eles provêem um conjunto de subsistemas pré-definidos, especificando os relacionamentos entre eles e estabelecendo regras para esses relacionamentos.
- Exemplo: Reflexão, MVC (*Model-View-Controller*).

O Padrão MVC

- O padrão MVC (*Model-View-Controller*) é um padrão de arquitetura, indicando como deve ser a organização global do sistema.
- Ele sugere a separação entre o modelo, a visão e o controle de uma aplicação:
 - O modelo corresponde às classes do domínio da aplicação.
 - A visão corresponde às classes de interface gráfica da aplicação.
 - Finalmente, o controle corresponde as classes que conectam o modelo à visão.

O Padrão MVC (2)

- Este padrão também indica que as classes do modelo não devem conhecer as classes da visão. Isto diminui o acoplamento entre estes componentes, implicando que podemos mudar a interface do sistema (de gráfica para textual, por exemplo) sem afetar as classes do modelo.
- De maneira análoga, as classes da visão (UI) não devem implementar regras de negócio (modelo).

O Padrão *MVC* (3)

- As classes de controle são o mecanismo que integra as classes da UI com as classes do modelo. Elas encapsulam como os objetos interagem promovendo um baixo acoplamento entre elas e permitindo que suas implementações possam ser modificadas independentemente.

O Padrão *MVC* (4)

- As classes de controle contém tipicamente informação de sequenciamento das operações. Deve-se tomar cuidado, pois as classes de controle NÃO DEVEM executar tarefas que tipicamente pertençam às outras classes.

Padrões de Análise e Projeto

- Fornecem um esquema para refinar os subsistemas ou componentes do sistema de software.
- Esses padrões possuem um grau de granularidade considerado **médio** e são **independentes** de linguagem de programação. Estes padrões, geralmente, correspondem a uma abstração de duas, três ou um pequeno número de classes.

Padrões de Análise e Projeto

- Os padrões de Coad são famosos padrões de análise.
- Os padrões de projeto mais famosos são os padrões de Erich Gamma, Richard Helm, Ralph Johnson, e John Vlissides. Estes autores ficaram conhecidos como *Gang of Four (GoF)*.
- Estes padrões correspondem ao chamado catálogo de padrões.
 - Exemplo: *Strategy, State*, etc.



Padrões de Análise

Exemplo: A abordagem de *Coad*

- A abordagem de Coad para padrões de software divide-se na apresentação de estratégias e de padrões propriamente ditos.
- Coad apresenta 177 estratégias e 31 padrões, que são mais adequados a fase de análise.
 - Coad não faz nenhuma distinção sobre quando os padrões podem ser aplicados.

Exemplo: A abordagem de *Coad*

- Uma estratégia é um plano de ação que deve ser utilizado para alcançar determinados objetivos durante a realização das atividades de construção do modelo de objetos.
- Um padrão é um bloco de construção útil para o desenvolvimento de software orientado a objetos.

Exemplo: A abordagem de *Coad*

- As estratégias estão divididas em 6 categorias, a saber:
 - Atividades e Componentes do Modelo
 - Identificar o propósito e características do sistema
 - Selecionar objetos
 - Estabelecer Responsabilidades
 - Construir cenários
 - Descobrir novas estratégias e padrões

Exemplo: A abordagem de *Coad*

- Exemplo de uma Estratégia para Identificação de Objetos. Estratégia Número 13: Selecionar Atores.
- Procurar por atores, que são pessoas e organizações que atuam como participantes no sistema em construção.
- Exemplos: pessoa, organização (agência, companhia, corporação, fundação, etc.).

Exemplo: A abordagem de *Coad*

- Exemplo de Padrão: "Transação - Item de Transação"



Em um restaurante ..

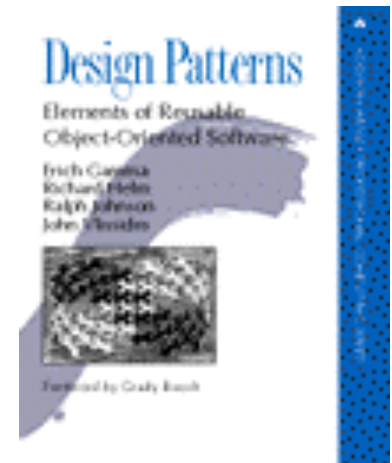
Padrões de Projeto

O Catálogo de Gamma

**Design Patterns:
Elements of
Reusable Object-
Oriented Software**

por Erich Gamma,
Richard Helm,
Ralph Johnson, e
John Vlissides.

Addison Wesley 1994.



O Catálogo de Gamma

- O catálogo de Gamma é o exemplo mais famoso de padrões de projeto. A partir deste livro, diversos outros trabalhos foram publicados. Este catálogo contém 23 padrões.
- Ele baseia-se na experiência obtida pelos autores com diversos *frameworks*, no quais estruturas de projeto elegantes, flexíveis e bem-estruturadas foram identificadas.

Forma dos Padrões no Catálogo

- Nome do Padrão e Classificação: o nome do padrão e sua classificação são apresentados.
- Intenção: esta seção deve responder às seguintes questões: O que o padrão faz? Qual sua razão e sua intenção? Que características particulares do problema o padrão resolve?
- Também conhecido como: outros nomes conhecidos para o padrão.
- Motivação: descreve um cenário com um problema prático onde o padrão é utilizado para resolver o problema.
- Aplicabilidade: esta seção deve responder à seguinte questão: em que situação o padrão pode ser aplicado?
- Estrutura: descreve a representação gráfica das classes do padrão através de uma notação baseada na OMT, além de diagramas de interação.

Forma dos Padrões no Catálogo

- Participantes: descreve as classes e objetos que participam do padrão, além de suas responsabilidades.
- Colaborações: descreve como os participantes colaboram para atingir as suas responsabilidades.
- Conseqüências: apresenta os resultados e decisões que devem ser tomadas para a implementação do padrão de projeto.
- Trechos de Código: fragmentos de código C++ ou Smalltalk que ilustram como o padrão pode ser implementado.
- Usos Conhecidos: exemplos do padrão encontrados em sistemas reais. No mínimo dois exemplos de diferentes domínios são apresentados.
- Padrões Relacionados: descreve outros padrões relacionados que podem substituir ou complementar o padrão que está sendo descrito.

Classificação dos Padrões no Catálogo

Escopo vs. Propósito	Criação	Estrutura	Comportamento
Classe	Factory Method	Adapter (classe)	Interpreter Template Method
Objeto	Abstract Factory Builder Prototype Singleton	Adapter (objeto) Bridge Composite Decorator Facade Flyweight Proxy	Chain of Responsibility Command Iterator Mediator Memento Observer State Strategy Visitor



Idiomas

Idiomas

- São os padrões de nível mais baixo, **específicos** para uma determinada linguagem de programação.
- Eles descrevem como implementar aspectos particulares de componentes e dos relacionamentos entre eles usando características específicas da linguagem alvo.

Idiomas

- Além disso, formam uma base para a padronização da nomenclatura e da estrutura do código fonte. Sob este ponto de vista, eles agem como diretrizes de projeto ou, ainda, como convenções para o modo de atribuir nomes.

Exemplo: o idioma de Pree

- Pree apresenta uma outra abordagem que se refere a convenções de nomes a serem dados a variáveis, constantes e métodos. O trabalho de Pree é baseado nas convenções descritas no framework ET++.
- A utilização de convenções de nomenclatura em frameworks é importante, pois facilita o aprendizado e utilização do mesmo.

Exemplo: o idioma de Pree

- Nomes de classes e de métodos: devem iniciar com letras maiúsculas seguidos de letras minúsculas.
- Variáveis locais devem iniciar com letras minúsculas.
- Se um nome consiste de várias palavras, a segunda e as palavras subsequentes iniciam com letras maiúsculas, como por exemplo *DoLeftButtonDownCommand(...)*.
- Variáveis globais: devem iniciar com o prefixo *g*, como: *gFileDialog*, *gApplication*.

Exemplo: o idioma de Pree

- Constantes: devem iniciar com o prefixo *c*, como: *cIdNone*.
- Métodos: quando definem o valor de uma variável instanciada devem iniciar com *Set*, como em *SetSaldo(..)*. Métodos que retornam tais valores devem iniciar com *Get*, como *GetOrigin(...)*.
- Métodos que desenham objetos na tela devem iniciar com *Gr*, como *GrPaintRect()*.
- Etc...

Anti-Padrões

- Os padrões indicam boas estruturas de análise, projeto ou implementação no desenvolvimento de software.
- **Os anti-padrões indicam exatamente o contrário.** O foco dos anti-padrões são os erros comuns dos projetistas e as principais falhas que levam ao fracasso no desenvolvimento de software.
- Anti-padrões apresentam também dicas práticas em como detectá-los e como corrigi-los.

Conclusões

- Como ser um bom jogador de xadrez? Primeiro, aprende-se as regras. Depois, estuda-se os exemplos de bons jogadores.
- Os padrões de software implementam a mesma idéia para desenvolvimento de software. Desta forma, são uma abordagem bastante promissora para o desenvolvimento de software.

Referências: Livros

- Livros do Christopher Alexander
- Gamma et al. "Design Patterns: ...", 1994.
- Coplien&Schmidt (eds.) "Pattern Languages of Program Design" Addison-Wesley, 1995.
- **Buschmann et al. "Pattern-Oriented Software Architecture", Wiley, 1996.**

Referências: Internet e Conferências

- <http://st.www.cs.uiuc.edu/users/patterns/patterns.html>
- PLoP: Conferência, EUA.
- EuroPloP: Conferência, Europa.
- OOPSLA: Sobre Orientação a Objetos, EUA.

Referências: Colunas de Revistas

- J. Coplien, *The Column Without a Name*, C++ Report, 94+
- R. Gabriel, *Critic-at-Large*, JOOP, 93-94
- Vlissides, *Pattern Hatching*, C++ Report, 95+
- Johnson, *Patterns of Thought*, ROAD, 94+

Outras Referências

- Lea, D. "Christopher Alexander: An Introduction for Object-Oriented Designers", SEN, 19(1), 1994
- Brad Appleton, "Patterns and Software: Essential Concepts and Terminology", <http://www.enteract.com/~bradapp/docs/patterns-intro.html>
- IEEE Software, janeiro/fevereiro 1997: Object Methods, Patterns, and Architectures
 - Coplien: Idioms and Patterns as Architectural Literature
 - Kerth and Cunningham: Using Patterns to Improve our Architectural Vision

Perguntas?
