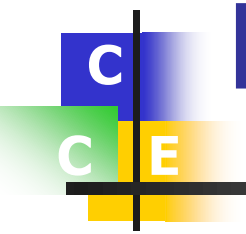
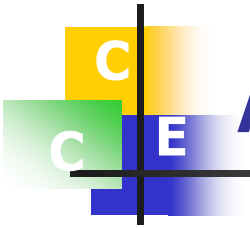


Tecnologias Atuais de Desenvolvimento de *Software*

Arquitetura, Padrões, *Frameworks* e MDA

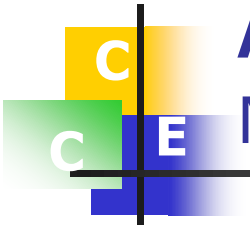
Prof. Luiz Antônio

lpereira@uninet.com.br



Agenda

- Arquitetura de *Software*
- *Patterns e Frameworks*
  - Introdução
  - Padrões de projeto
  - *Frameworks*
- MDA

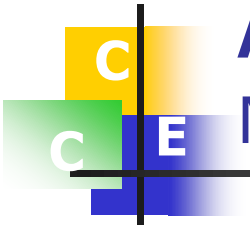


Arquitetura de S/W
Motivação

- Durante o desenvolvimento de um sistema
  - Parte das decisões está ligada ao ambiente onde o *software* irá operar...
  - ... e parte diz respeito a quais serão os *componentes* do *software,* como eles se comunicarão e quais serão suas funcionalidades e restrições de uso.
- Essas questões dizem respeito à *arquitetura* do sistema sendo desenvolvido.

Arquitetura de S/W
Motivação

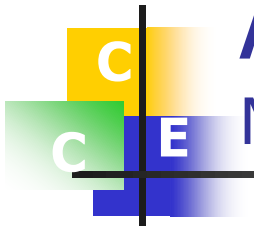
- Por que a nossa preocupação com a arquitetura?
  - Porque arquiteturas diferentes envolvem custos diferentes:
    - Arquiteturas diferentes impõem qualificações diferentes da equipe de desenvolvimento;
    - Arquiteturas diferentes impõem custos diferentes, inclusive de implantação e de infra-estrutura.
  - Porque arquiteturas diferentes envolvem projetos diferentes.



Arquitetura de S/W

Motivação

- A opção por uma arquitetura tem de ser feita, portanto, antes do início do projeto para que:
  - Os custos globais sejam mais precisamente definidos e, portanto, possam ser perseguidos;
  - Idem com relação aos prazos.



Arquitetura de S/W

Motivação

Engenharia de *Software* envolve Arquitetura de *Software*.

Arquitetura de S/W
Motivação

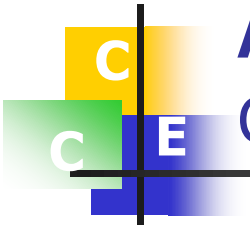
- Arquitetura de software ganhou força com a OO e componentes;
- Já é tratada como uma área de especialização;
- Principal referência: Mary Shaw & David Garland, "*Software Architecture – Perspectives on an Emerging Discipline*", 1996.

Arquitetura de S/W
O que é?

- Descrição, de alto nível, da estrutura do sistema como um todo. Inclui:
  - Organização dos componentes do sistema;
  - Estruturas globais de controle;
  - Protocolos;
  - Funcionalidades dos componentes;
  - Distribuição física dos componentes;
  - Capacidade de evolução ...

Arquitetura de S/W
Como descrever?

- Descrever a arquitetura de um sistema é descrever
  - Os componentes
  - Os conectores
  - As restrições
- Normalmente utiliza-se diagramas, onde os *componentes* são representados por blocos e as *interações* por linhas (diagramas de classes, de componentes, de implantação).

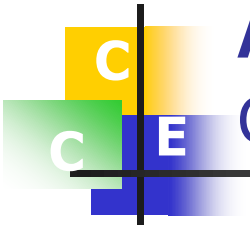


Arquitetura de S/W

Componentes e conectores

Componentes:

- Matéria prima para descrição da arquitetura de um sistema. Exemplos:
  - Clientes
  - Servidores
  - Filtros
  - Bancos de Dados
  - Camadas de *Software* (*Layers*)



Arquitetura de S/W

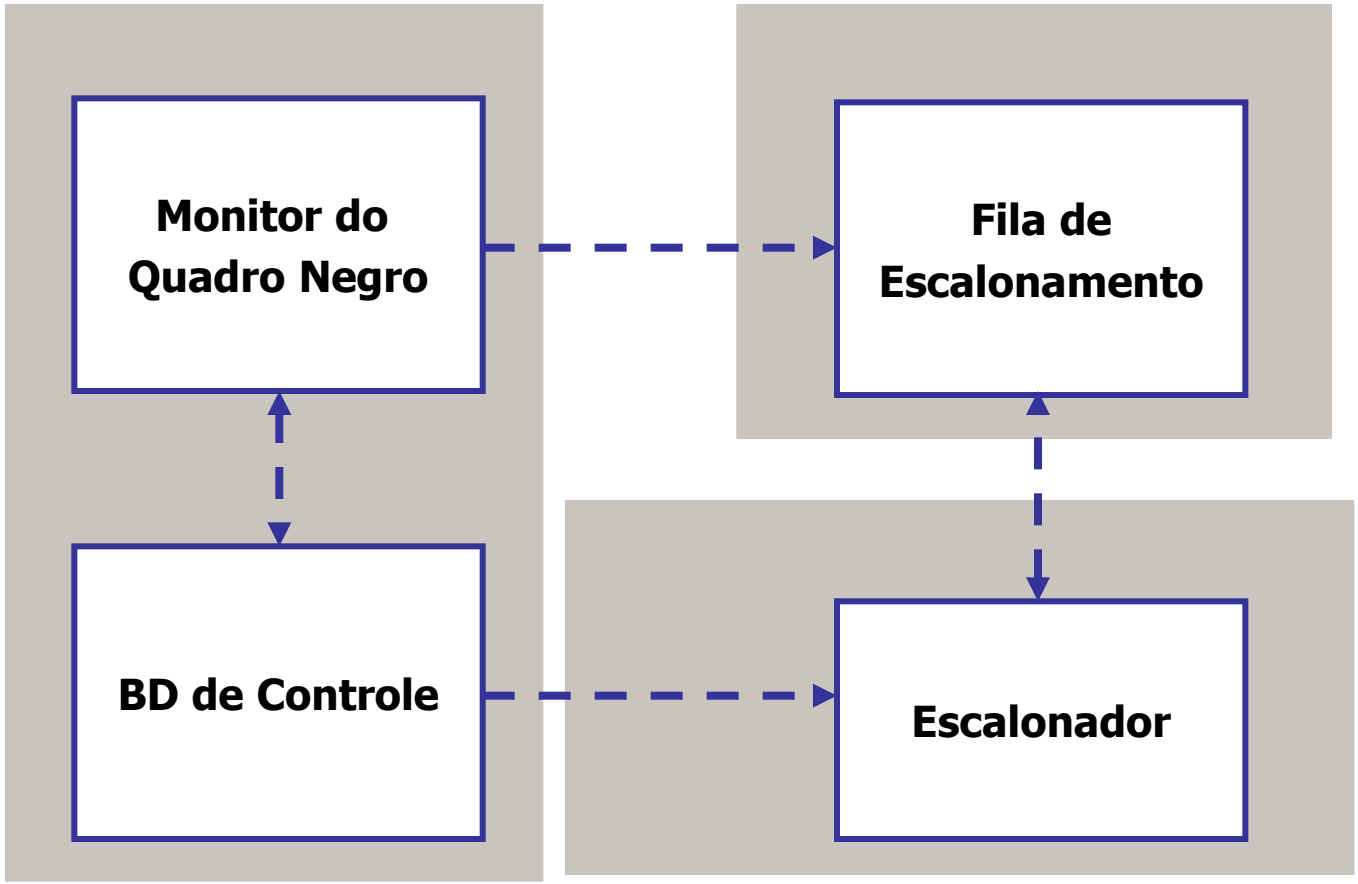
Componentes e conectores

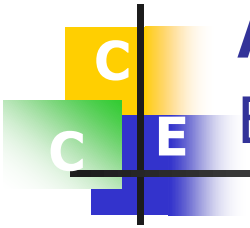
Conectores:

- Provêem a interação entre os componentes, para que o sistema funcione. Exemplos:
  - Chamadas de rotinas
  - Variáveis compartilhadas
  - Protocolos Cliente-Servidor
  - *Broadcasting*
  - Eventos
  - ...

Arquitetura de S/W

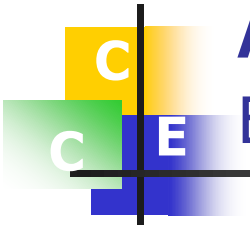
Componentes e conectores – exemplo (quadro negro)





Arquitetura de S/W
Estilos

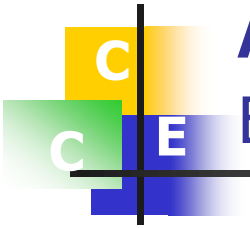
- Existem diferentes estilos para a organização do *software* –> Estilos de Arquitetura;
- Compreendem conjuntos de componentes, conectores e formas (regras) de como combiná-los.



Arquitetura de S/W

Estilos

- Principais Estilos:
  - Sistemas em Camadas
  - *Pipes and Filters*
  - Invocação Implícita
  - Repositórios
  - Sistemas Distribuídos



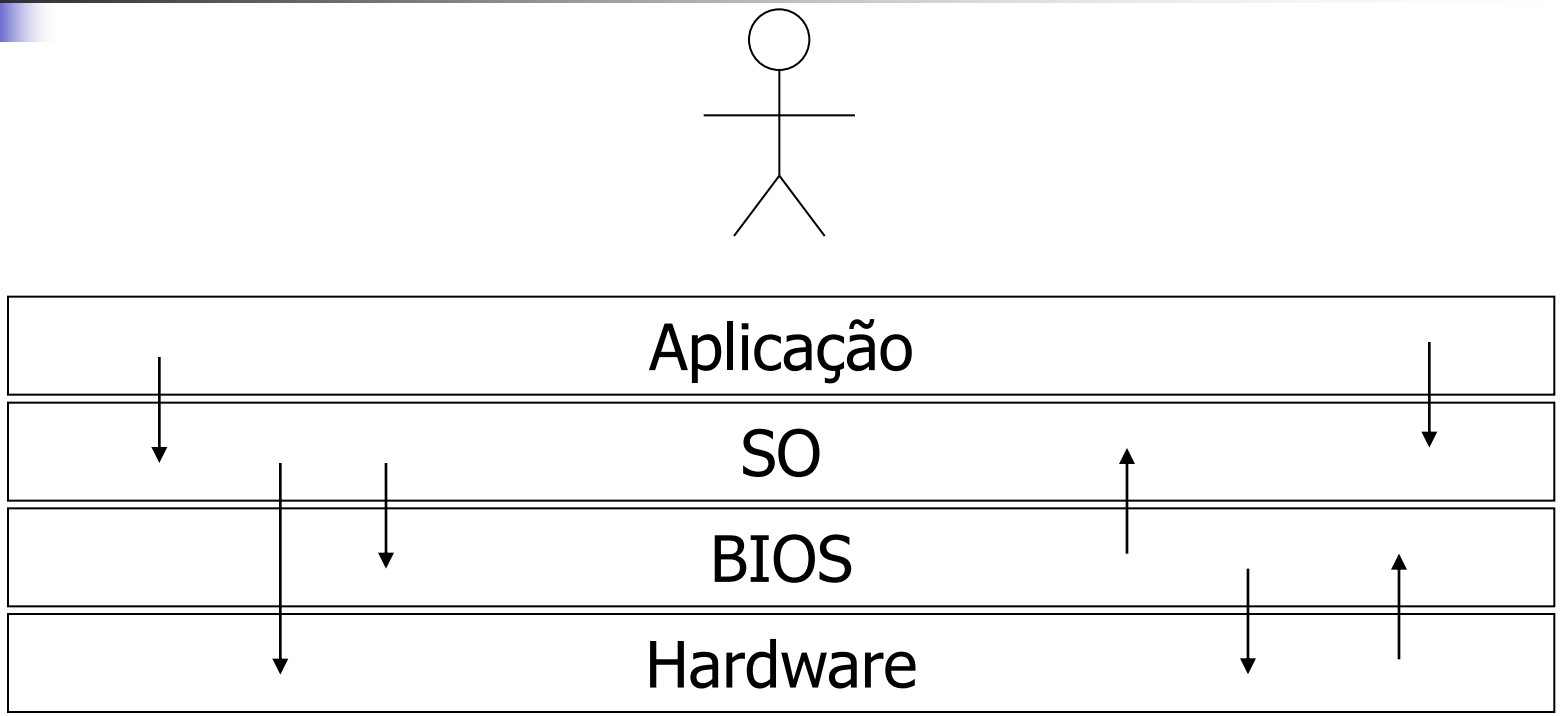
Arquitetura de S/W

Estilos: Sistemas em camadas

- Componentes
  - Camadas
- Conectores
  - Protocolos que definem as interações entre as camadas

Arquitetura de S/W

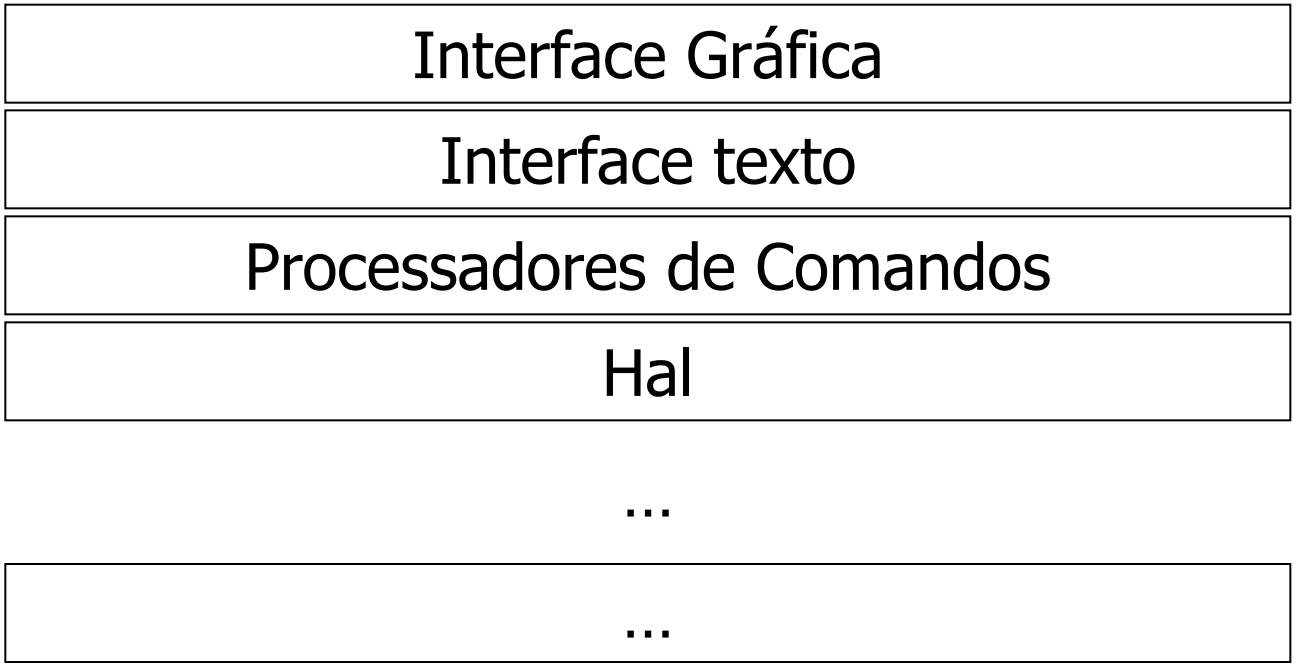
Estilos: Sistemas em camadas



Arquitetura de S/W

Estilos: Sistemas em camadas

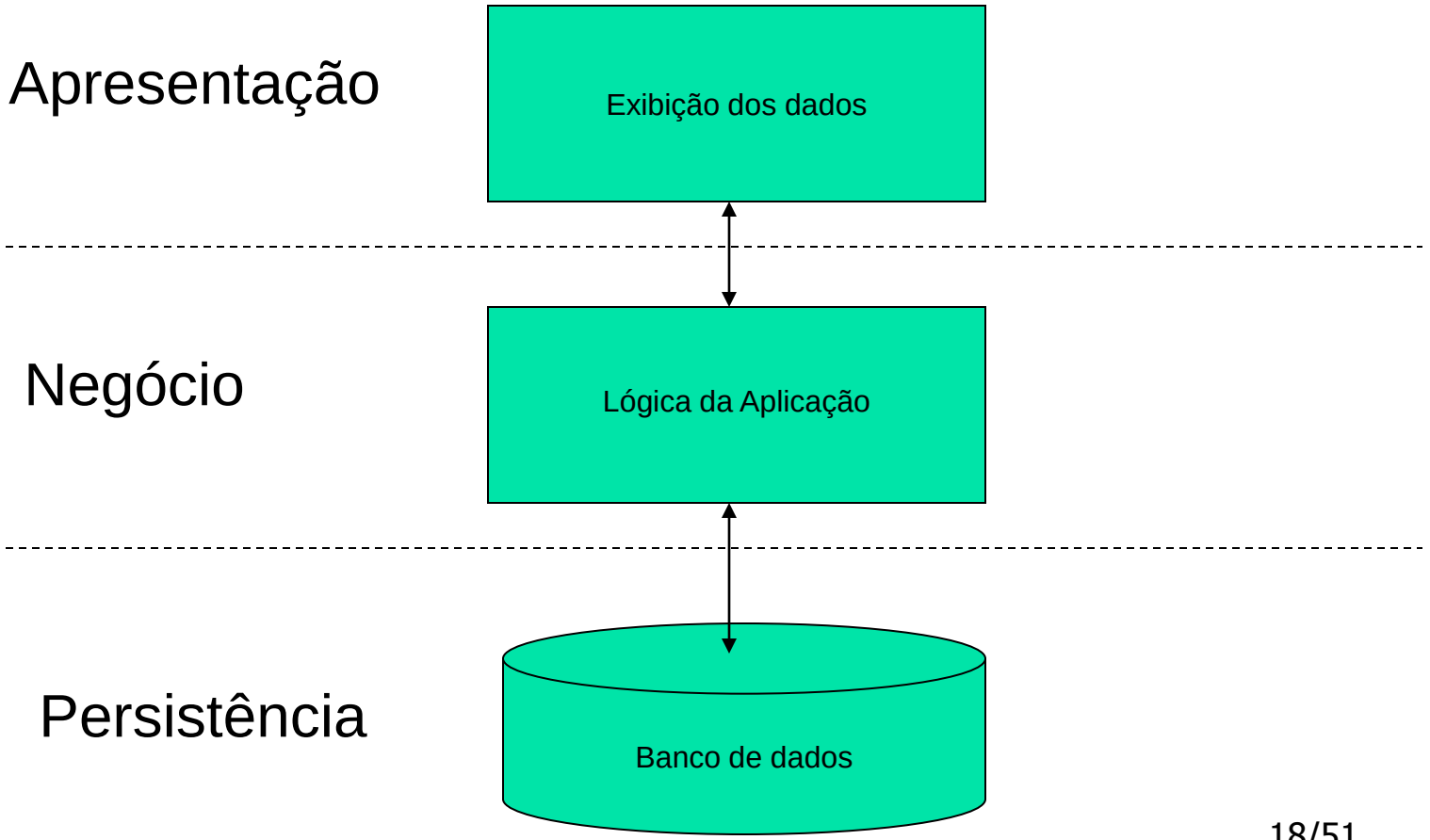
Camada do SO:



Arquitetura de S/W

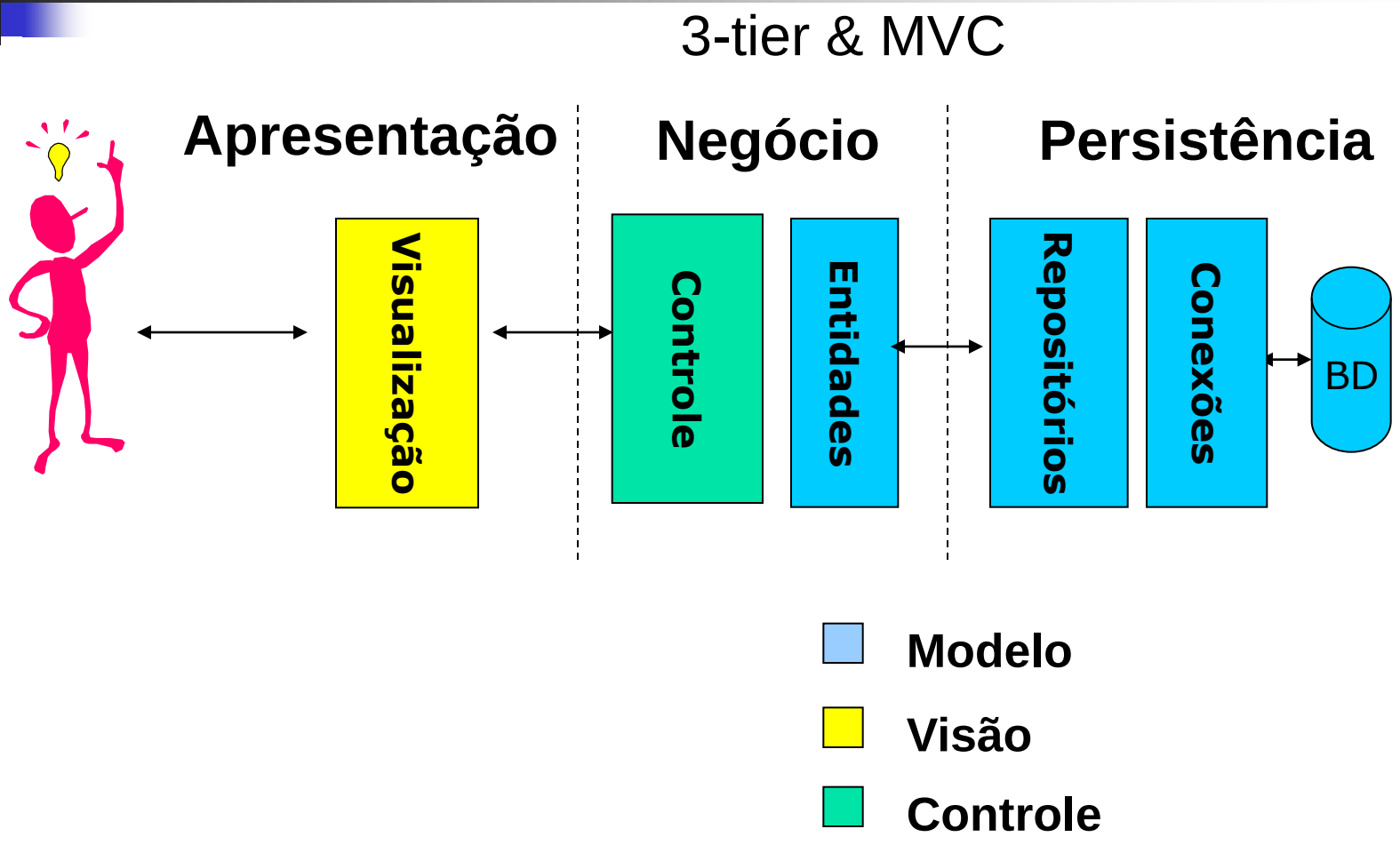
Estilos: Sistemas em camadas

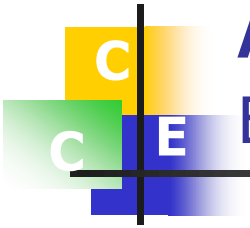
A arquitetura clássica de três camadas



Arquitetura de S/W

Estilos: Sistemas em camadas

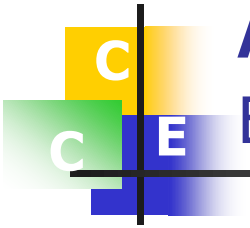




Arquitetura de S/W

Estilos: Sistemas em camadas

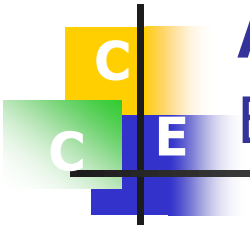
- Vantagens
  - Suportam diferentes níveis de abstração;
  - Permitem, também, a organização da aplicação em camadas com funcionalidades/conceitos coesos (e.g. MVC, *3-tier*)
  - Permitem a evolução (retira-se uma camada e coloca-se outra no lugar, bastando manter a API).



Arquitetura de S/W

Estilos: Sistemas em camadas

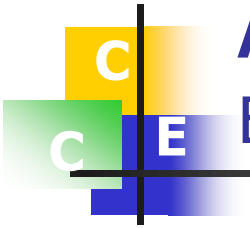
- Desvantagens
  - A estruturação em camadas pode ser impossível em alguns sistemas (por exemplo, por conta da degradação da performance devido a necessidade de pelo menos uma chamada para atravessar cada camada de software em sistemas de controle em tempo real);
  - Dificuldade em se determinar os níveis corretos de abstração durante o projeto.



Arquitetura de S/W

Estilos: Sistemas distribuídos

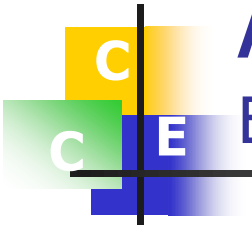
- Componentes
  - Módulos distintos e especializados que interagem de forma colaborativa, eventualmente processados em nós diferentes
- Conectores
  - Protocolos de requisição de serviços e provimento de respostas
- Aplicações
  - Sistemas tolerantes a falhas, p. exemplo



Arquitetura de S/W

Estilos: Sistemas distribuídos

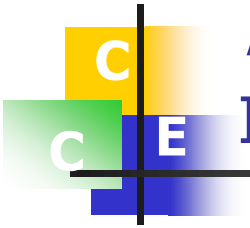
- Vantagens
  - Distribuição da carga de processamento (cada nó processa um pouquinho)
  - Reuso
  - Especialização
- Desvantagens
  - Performance/confiabilidade sujeitos aos critérios da distribuição (qualidade da infraestrutura de comunicação, segurança física também é distribuída -> custo)



Arquitetura de S/W

Estilos: Sistemas distribuídos

- Variação importante
  - Sistemas cliente/servidor
    - Servidores especializados em prover serviços específicos simultaneamente a vários clientes
    - Clientes executam processamento local, junto ao usuário (provendo IU e alguma inteligência local) e se comunicam com os servidores solicitando serviços.
  - Exemplos: 3-tier, SGBDs Oracle/Clientes Delphi, Servidores de Páginas/IE.

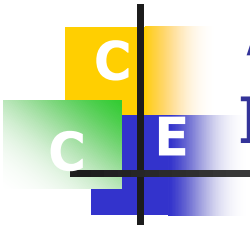


Patterns e Frameworks

Introdução

- Atual busca por
  - Economia de recursos (tempo, "fosfato" e $$$);
  - Qualidade.
- Aliada ao fato de que a similaridade de problemas pressupõe a similaridade de soluções;

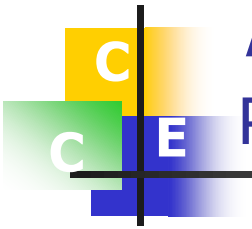
⇨ **Reúso** de técnicas, artefatos, processos, ferramentas, ..., *software*, ..., que já se provaram corretos, eficazes, eficientes, etc.



Patterns e Frameworks

Introdução

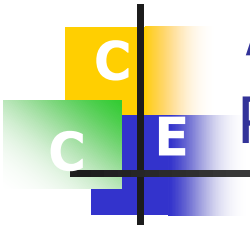
- **Reúso**
  - De uma solução em uma mesma área de aplicação, para um problema *grande* ⇒ ***framework***. Ex:
    - Gestão de Pessoal,
    - Gestão Financeira,
    - Gestão de Estoques...
  - De uma solução em várias áreas de aplicação, para um problema *pequeno* ⇒ ***pattern***. Ex:
    - Como impedir a criação de mais de um objeto de uma mesma classe,
    - Como estabelecer o isolamento entre módulos de um sistema...



Patterns e Frameworks

Padrões de projeto: histórico

- *Patterns* são soluções genéricas e reutilizáveis para a aplicação em classes de problemas bem conhecidos, ou seja, soluções que um dia funcionaram são transformadas em *receitas* para situações parecidas;
- Essas soluções devem ser projetadas com flexibilidade (... para que sejam genéricas).

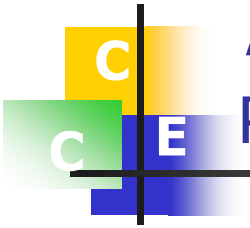


Patterns e Frameworks

Padrões de projeto: histórico

- Em 1994, 4 autores – Erich Gamma, Richard Helm, Ralph Johnson e John Vlissides – publicaram o primeiro catálogo de *Design Patterns* (23 patterns) para projeto de programas OO: *Design Patterns – Elements of Reusable Object-Oriented Software* (*The GoF* book*).

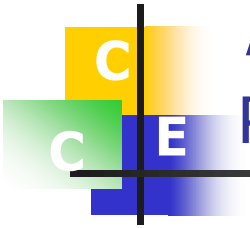
(*) GoF = Gang of Four



Patterns e Frameworks

Padrões de projeto: classificação

- Padrões podem ser classificados quanto ao:
  - **Escopo** – especificam se o padrão se aplica, primariamente, a classes ou a objetos.
    - Padrões de classes: lidam com relacionamentos entre classes e suas sub classes. São estabelecidos por herança. Não se alteram em tempo de execução.
    - Padrões de objetos: lidam com relacionamentos entre objetos, que podem ser alterados em tempo de execução.
  - **Propósito** – refletem o que o padrão faz.
    - Padrões de criação: se preocupam cm o processo de criação de objetos;
    - Padrões estruturais: lidam com a composição (conjuntos) de classes e objetos;
    - Padrões comportamentais: caracterizam as formas segundo as quais objetos e classes interagem e distribuem responsabilidade.
- A tabela a seguir ilustra como os padrões GoF se enquadram nessas categorias.



Patterns e Frameworks

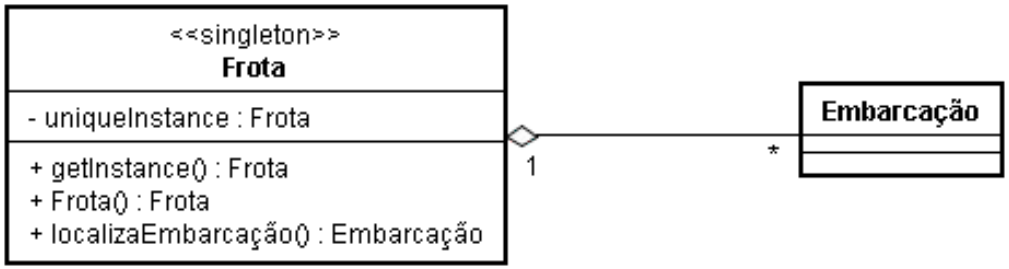
Padrões de projeto: classificação

		Propósito		
		Criação	Estrutural	Comportamental
Escopo	Class	Factory Method	Adapter (classe)	Interpreter
				Template Method
	Objeto	Abstract Factory	Adapter (objeto)	Chain of Responsibility
		Builder	Bridge	Command
		Prototype	Composite	Iterator
		Singleton	Decorator	Mediator
			Façade	Memento
			Flyweight	Observer
			Proxy	State
				Strategy
				Visitor

Patterns e Frameworks

Padrões de projeto: exemplos

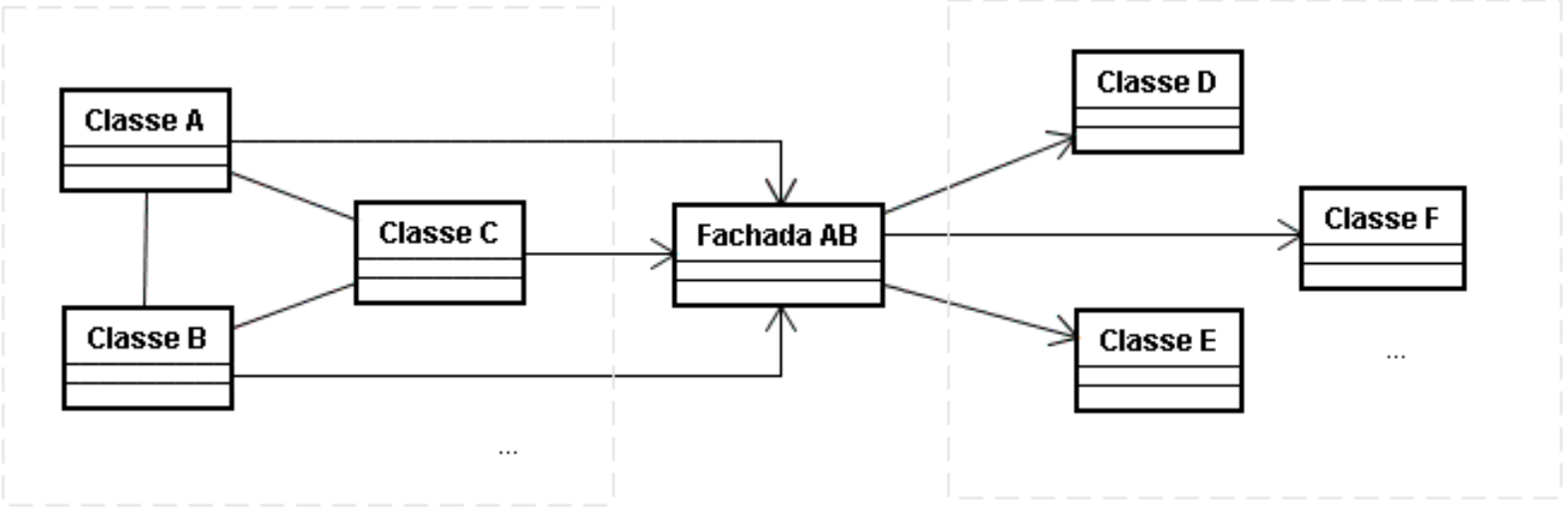
- Singleton (Criação)
  - Objetivos
    - Garantir que uma classe tenha somente uma instancia;
    - Fornecer um ponto de acesso global para a mesma.
  - Exemplo



Patterns e Frameworks

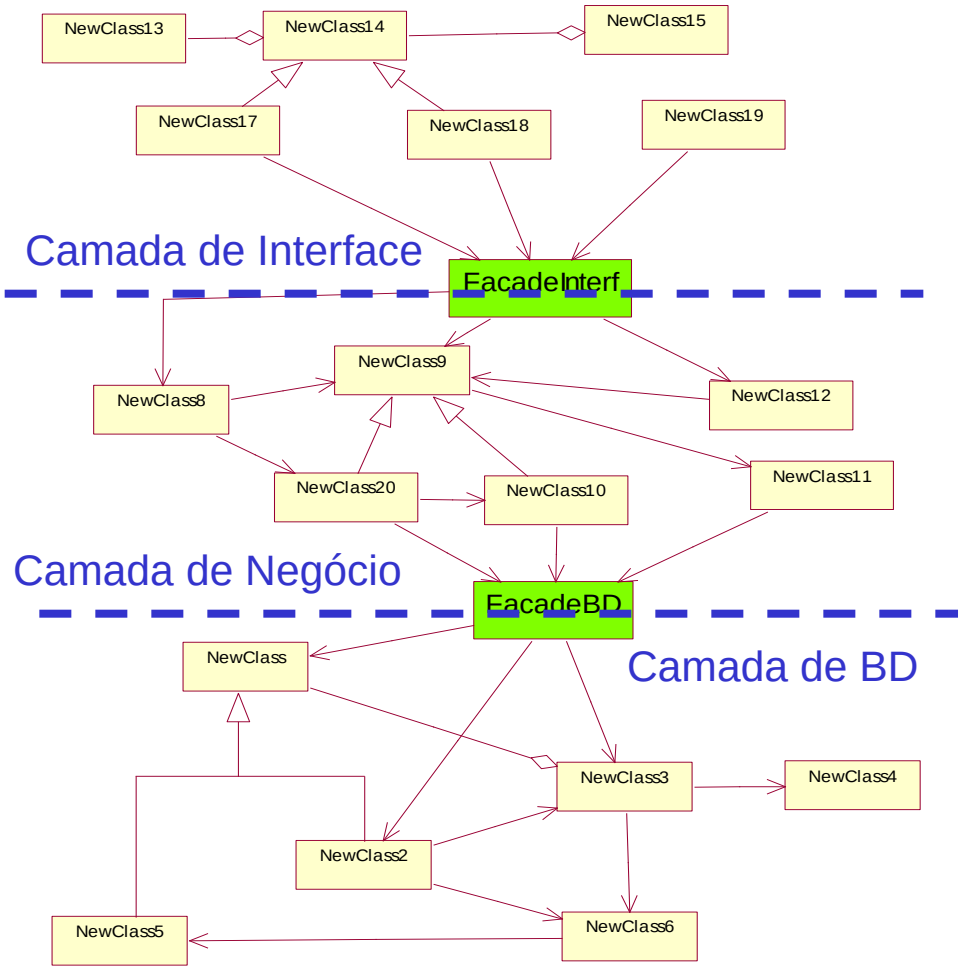
Padrões de projeto: exemplos

- Fachada – *Façade* (Estrutural)
  - Objetivos
    - Substitui um conjunto de interfaces por uma única.
    - Diminui acoplamento, encapsula...
  - Exemplo



Patterns e Frameworks

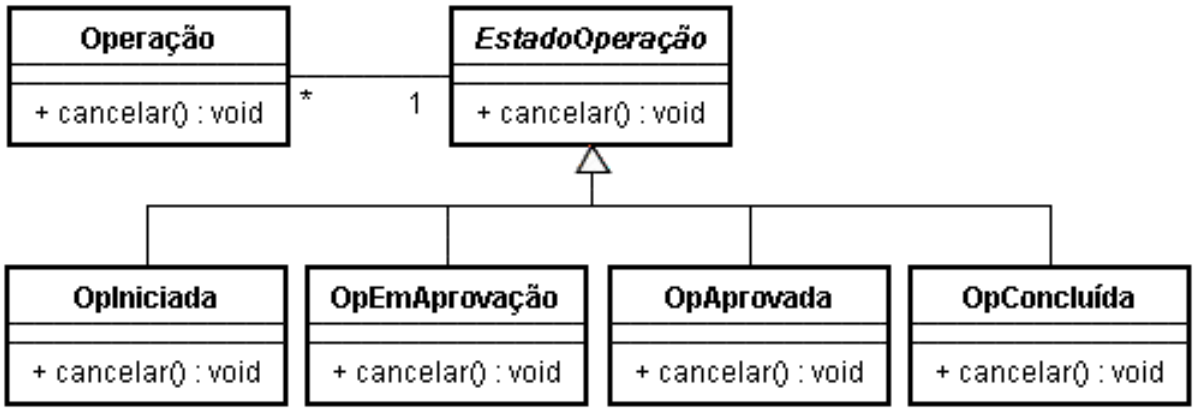
Padrões de projeto: exemplos

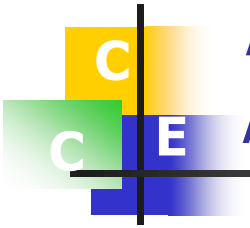


Patterns e Frameworks

Padrões de projeto: exemplos

- State (Comportamental)
  - Objetivos
    - Permite mudar-se o comportamento de um objeto quando o estado dele muda.
  - Exemplo

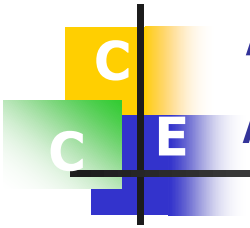




Patterns e Frameworks

Frameworks

Ainda, com o propósito de se promover o reúso de uma solução (reúso é uma das principais metas de um engenheiro de *software*)...



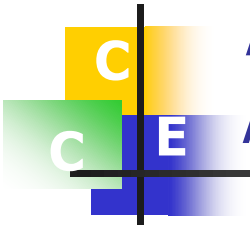
Patterns e Frameworks

Frameworks

- Visando à diminuição do tempo para produção de uma aplicação e
- Estendendo a reutilização também para o projeto (além do código*).

⇨ ***Frameworks*** (Arcabouços)

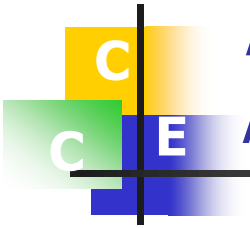
(*) O início, com as Folhas de Pagamento



Patterns e Frameworks

Frameworks

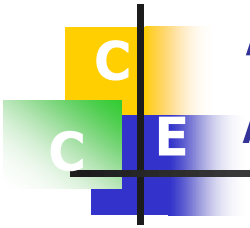
- Um *framework* é
  - Um arcabouço de aplicação desenvolvido para emprego em um domínio com o objetivo de reutilizá-lo em mais de uma aplicação nesse domínio.



Patterns e Frameworks

Frameworks

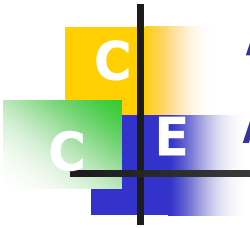
- Desenvolvimento de *software* baseado em *frameworks* é uma técnica de reuso que compreende as etapas de projeto e implementação;
- Um *framework* é representado em termos de classes abstratas e concretas;
- O *framework* captura os conceitos mais gerais de um domínio, ou seja, os conceitos comuns a uma família de aplicações dentro de um mesmo domínio;
- Uma aplicação gerada a partir de *framework* especializa ou estende os conceitos embutidos no *framework,* podendo ainda adicionar novos conceitos.



Patterns e Frameworks

Frameworks

- As partes do *framework* que são abertas à adaptação (customização e extensão) são chamadas de ***hot spots*** (pontos de flexibilização);
- As outras partes (as partes "imexíveis" ☺) são chamadas de **frozen spots** (kernel/núcleo).



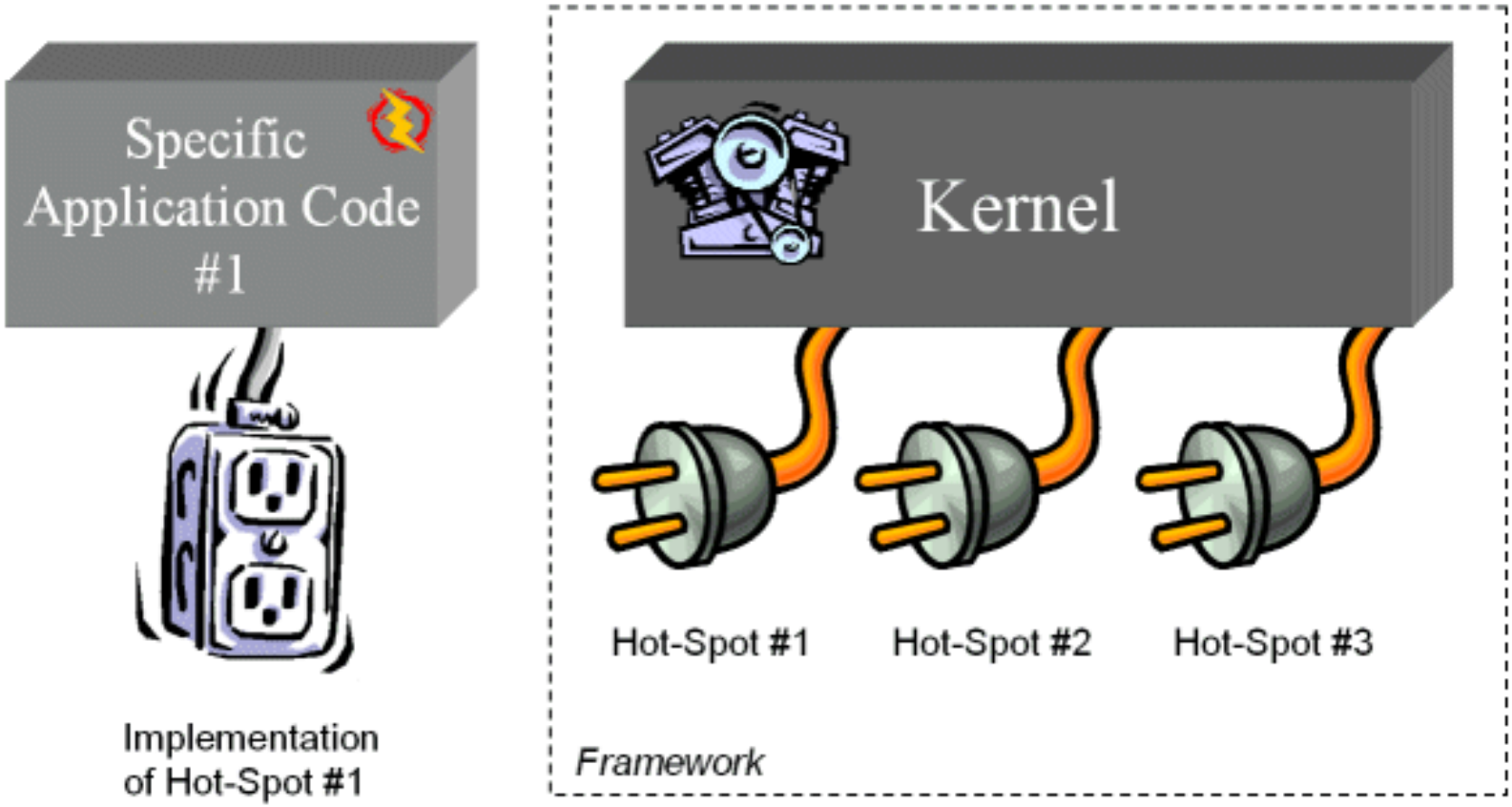
Patterns e Frameworks

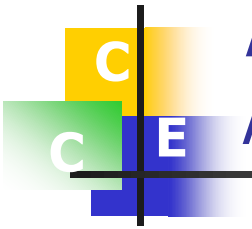
Frameworks

- *Frameworks* não são executáveis (são arcabouços). Uma aplicação completa, executável, é obtida por *instanciação* do *framework*, implementando código específico da aplicação para cada *hot spot.*
- Os *frozen spots* são partes de código já implementadas no *framework* que chamam um ou mais *hot spots* definidos em cada instância ("*old code calls new code*" – ilustração a seguir)

Patterns e Frameworks

Frameworks





Patterns e Frameworks

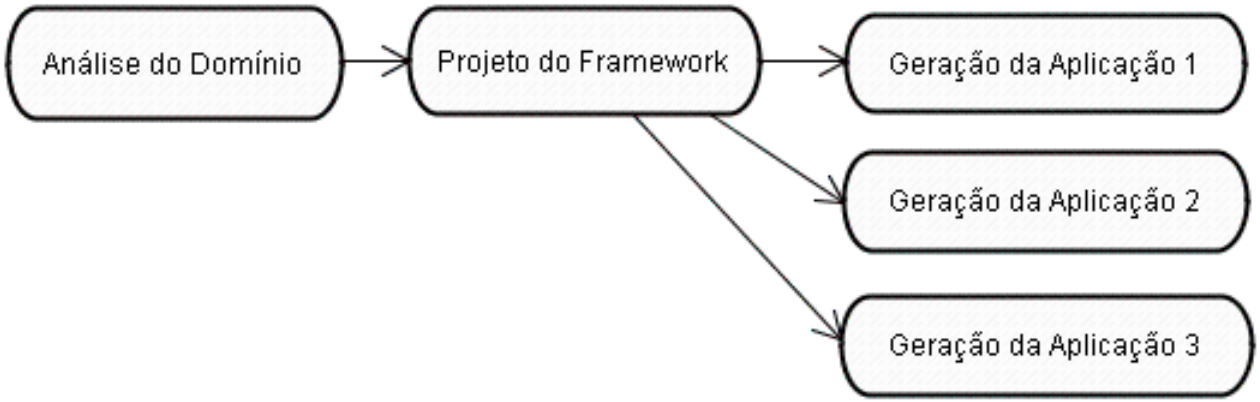
Frameworks

- Nem sempre é possível saber no princípio do desenvolvimento de um projeto qual será a tecnologia dominante ao final dele (e.g. tecnologia de persistência)
  - *Frameworks* podem proporcionar independência de tecnologia;
  - Proporcionam evolução rápida e barata de aplicações, embora o custo de desenvolvimento de um *framework* seja muito maior do que o de desenvolvimento de uma aplicação isolada.

Patterns e Frameworks

Frameworks

Desenvolvimento OO tradicional

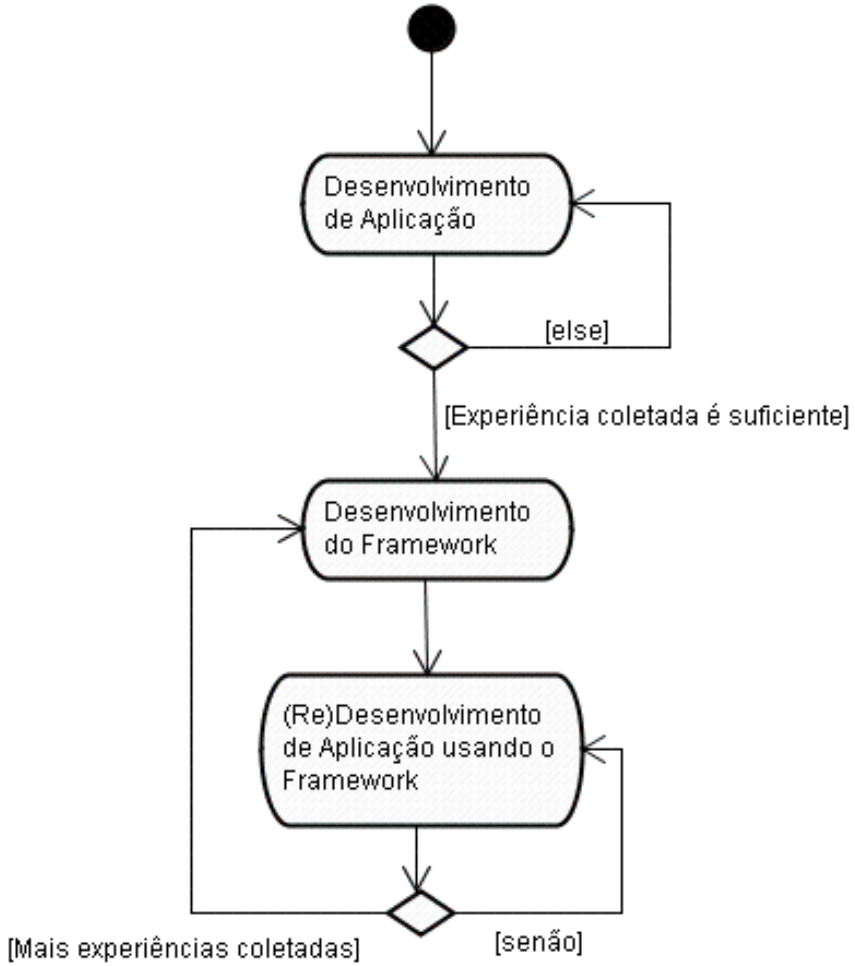


Desenvolvimento baseado em frameworks

Patterns e Frameworks

Frameworks

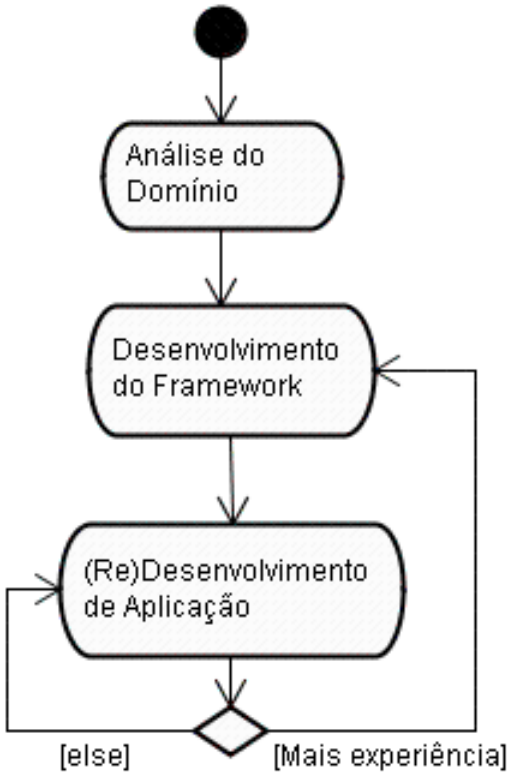
Processo de desenvolvimento baseado na aquisição de experiência.

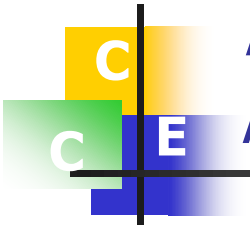


Patterns e Frameworks

Frameworks

Processo de desenvolvimento baseado na análise do domínio.

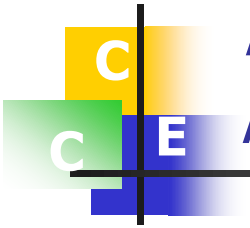




Patterns e Frameworks

Frameworks

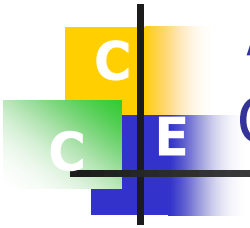
- Benefícios
  - Modularidade
  - Reutilização
  - Extensibilidade
  - ⇨> Qualidade
  - ⇨< Preço



Patterns e Frameworks

Patterns e *Frameworks*: resumo

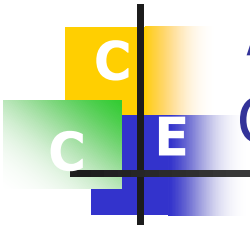
- Resumindo:
  - ***Pattern***: é o registro de uma solução para um problema menor, específico, mas não necessariamente em um domínio específico
  - ***Framework***: é o registro (descrição) de uma solução para uma aplicação em um domínio específico



Model-Driven Architecture

Conceitos

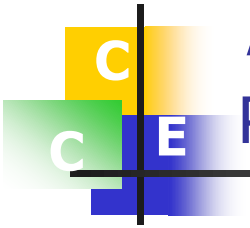
- MDA é uma abordagem para projeto de *software* proposta e patrocinada pelo OMG;
- É uma arquitetura que provê uma metodologia para se estruturar especificações de sistemas expressas como modelos;
- É *model-driven* (direcionada a modelos) porque provê formas de se usar modelos para direcionar o curso no entendimento, projeto, construção, implantação, operação e manutenção (enfim, todo o ciclo de vida) de um sistema;
- MDA agrega importância à modelagem.



Model-Driven Architecture

Conceitos

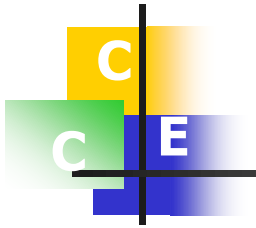
- Metodologia MDA:
  - As funcionalidades do sistema devem ser, primeiramente, especificadas como um modelo independente de plataforma (PIM), também chamado de modelo de negócio ou de domínio;
  - Dado um modelo de definição de plataforma (PDM) - .NET, Web, … - o PIM deve ser traduzido para um ou mais modelos específicos da plataforma escolhida (PSM) para que o código seja gerado na linguagem escolhida.
- A derivação PIM→PSM é feita, normalmente, de forma automatizada, através de ferramentas de transformação de modelos.



Model-Driven Architecture

Principais objetivos

- Separar-se projeto, arquitetura e tecnologia no desenvolvimento de um sistema.
- Permitir a alteração da tecnologia e da arquitetura com um menor (se possível nenhum) impacto sobre as questões de negócio.



Obrigado