**ARQUITETURA DE SOFTWARE PRECRITIVA PARA PROJETOS EM .NET**

Baseada em DDD

# Introdução

Arquitetura de software é centrada na ideia da redução da complexidade através da abstração e separação de interesses (separation of concerns (SoC)). O glossário do site oficial SOFTWARE ENGINEERING INSTITUTE (Instituto de Engenharia de Software) descreve que arquitetura de software é a estrutura ou estruturas de um sistema, com todos os elementos de software vendo e tendo suas propriedades vistas por todos os outros elementos e relacionamentos.

Referência: Software Engineering Institute - http://www.sei.cmu.edu/

Este documento pretende descrever uma arquitetura prescritiva de como desenvolver projetos de software baseados na plataforma .Net Framework, definindo modelos, padrões, e melhores práticas baseadas em conceitos de desacoplamento de Domain-Driven Desgin.

# Arquitetura Orientada ao Domínio (DDD)

Domain-Driver Design é uma abordagem prática para design de software com base na importância do Domínio de negócios, seus elementos e comportamento e relações entre sí.

Para se implementar o DDD é necessário identificarmos uma série de padrões de design.

## Design Patterns - O que é um padrão?

Um padrão é uma solução reutilizável que pode ser comumente aplicada em problemas que ocorrem no projeto de software, no nosso caso, em escrever aplicações web em C# .Net ou JavaScript.

Os padrões de projeto tem três benefícios principais:

Os padrões são soluções comprovadas: Eles fornecem abordagens sólidas para resolver problemas no desenvolvimento de software usando técnicas comprovadas que reflectem a experiência e conhecimentos que os desenvolvedores ajudaram a definir e trazer para o padrão.

Padrões podem ser facilmente reutilizados: Um padrão geralmente reflete uma solução fora da caixa que pode ser adaptado para atender às nossas próprias necessidades.

Os padrões podem ser expressivo: Quando olhamos para um padrão há geralmente uma estrutura de conjunto e vocabulário com a solução apresentada e de maneira elegante.

Referência: “Learning JavaScript Design Patterns”, A book by Addy Osmani

<http://addyosmani.com/resources/essentialjsdesignpatterns/book/>

### Categorias de padrões de projetos

Os padrões de projeto podem ser divididos em um número diferente de categorias. Neste documento vamos ver três destas categorias e mencionar alguns exemplos dos padrões que se enquadram nelas e mais a frente explorar em mais detalhes:

**Design Patterns Criacionais**

Padrões de projeto criacionais concentram em manipulação de mecanismos de criação de objeto, onde os objetos são criados de uma forma adequada para a situação em que está trabalhando. A abordagem básica à criação de objetos de outro modo poderia conduzir a complexidade de um projeto, enquanto esses padrões têm como objectivo resolver este problema e controlar o processo de criação.

Alguns dos padrões que se enquadram nesta categoria são: Constructor, Factory, Abstract, Prototype, Singleton and Builder.

**Design Patterns Estruturais**

Padrões estruturais estão preocupados com a composição de objetos e, normalmente, identificar maneiras simples de perceber relações entre objetos diferentes. Eles ajudam a garantir que, quando uma parte de um sistema muda, a estrutura inteira do sistema não precisa de fazer o mesmo. Eles também auxiliam na reformulação de partes do sistema que não se encaixam em um determinado propósito.

Padrões que se enquadram nesta categoria incluem: Decorator, Facade, Flyweight, Adapter and Proxy.

**Design Patterns Comportamentais**

Padrões de comportamento se concentrar em melhorar ou racionalizar a comunicação entre objetos díspares em um sistema.

Alguns padrões de comportamento incluem: Iterator, Mediator, Observer and Visitor.

Padrões que vamos explorar neste dcoumento:

* MVC
* MVP
* MVVM
* Repository
* Entity
* Aggregate
* Value-object
* Unit Of Work
* Services
* Singleton
* Observer
* Facade
* Factory
* Decorator

## Tecnologias utilizadas

Tecnologias (frameworks) e ferramentas que utilizamos nos projetos web em .Net:

* dotNET Framework 4.5
* MVC Framework 4.0
* SQL Server 2008 R2
* Entity Framework 5.1
* Visual Studio 2012
* Entity Framework Power Tools
* TestDriven.Net
* NUnit
* dotCover
* Selenium WebDriver

# Arquitetura em Camadas (Layers/N-tiers)

Para facilitar o entendimento, definiremos a solução arquitetural a partir da divisão lógica, em camadas que visa distinguir as partes da aplicação pelas macro responsabilidades, as Camadas (Layers) fazem uma divisão lógica da aplicação, enquanto que a arquitetura em Níveis (Tier) define a distribuição física da aplicação.

O modelo proposto para a arquitetura em camadas deste projeto será conforme exemplificado no diagrama de camadas abaixo.

Antes de definirmos as responsabilidades de cada camada, é importante fazer uma relação entre a arquitetura demostrada de forma abstrata no diagrama de camadas acima, com sua implementação no C# e Visual Studio 2012.

Criamos, uma prova de Conceito (POC) para ajudar a compreensão deste documento, sempre fazendo uma referência entre o que é definido no documento com sua implementação, o repositório que contém esse projeto exemplo está disponível em "https://servidorDeDesenvolvimento/svn/NomeDaEmpresa-POC" e pode ser acessando com o Tortoise e sua conta de domínio.

## Camada de Apresentação

### A Arquitetura MVC

### .Net Framework MVC 4

#### Razor Syntax

#### A View

#### Pasta Content

#### Pasta Scripts

#### Pasta Views

#### WebConfig e AppSettings

#### Global.sax

#### O Controller

#### O ViewModel

#### O Profile Mapper

### HTML 5

### JavaScript e Fameworks

### JQuery

Framework javascript para desenvolvimento cliente-side, toda implementação em javascript será feita por intermédio deste framework. <http://jquery.com/>

Exemplo de **Facade Pattern** com javascript:

Para entender como padrões podem ser úteis, vamos rever um problema de seleção de elemento muito simples que a biblioteca jQuery resolve para nós.

Imagine que temos um script onde para cada elemento do DOM encontrado em uma página com a classe "Teste" queremos incrementar em um contador. Qual é a maneira mais eficiente de acessar esta coleção de elementos? Bem, existem algumas maneiras diferentes de abordar este problema:

* Selecione todos os elementos da página e, em seguida, armazenar as referências a eles. Em seguida, filtrar esta coleção e usar expressões regulares (ou outro meio) para armazenar apenas aqueles com a classe “Teste”.
* Use um recurso do navegador nativo moderno, como querySelectorAll () para selecionar todos os elementos com a classe “Teste”.
* Use um recurso nativo como getElementsByClassName () para semelhante voltar a coleção desejada.

A opção 3 é a mais rápida; porém em um aplicativo do mundo real ela não funcionará em versões do Internet Explorer anteriores a 10, portanto, é necessário usar a opção 1. Pois a opção 2 também não são suportadas em versões antigas de navegadores.

Os desenvolvedores que usam jQuery não tem que se preocupar com este problema, que é abstraída para nós usando o padrão **Facade**. Este padrão oferece um simples conjunto de interfaces abstratas (por exemplo $el.css()) para diversas partes subjacentes mais complexas de código. Isso significa menos tempo tendo que se preocupar com detalhes do nível de implementação.

Nos bastidores, a biblioteca simplesmente opta por uma abordagem ideal para selecionar elementos, dependendo do que o nosso navegador suporta e só precisamos consumir a camada de abstração. Com o jQuery $("seletor") é significativamente mais fácil selecionar elementos HTML em uma página; do que optar manualmente por getElementById(), getElementsByClassName() ou getElementByTagName().

Referência: http://addyosmani.com/resources/essentialjsdesignpatterns/book/.

### JQuery UI

<http://jqueryui.com/>

### KnockoutJS

http://knockoutjs.com/

Exemplo de Observer Pattern:

...

### AngularJS

https://angularjs.org/

## Camada de Serviços Distribuídos

### Profile Mapper

## Camada de Aplicação

### Não fazer na camada de aplicação:

### Vantagens

## Camada de Domínio

### Entidades do Domínio

### Agregadores

### Objetos de Valor

### Serviços de Tarefas

### Serviços de Workflow

### Especificações(Specifications)

#### Exemplo: Specification Aluno maior de Idade 3

#### Exemplo: serviço consumindo a specification

### Contratos de Repositórios

### Padrão Unidade de Trabalho(Unit Of Work)

### Gestão de Exceções

#### Log de Exceções

### Validações

#### Validações de Integridade de Domínio

#### Regras de transação de negócio

#### Desacoplamento e Injeção de dependência com IoC do Unity

## Camada de infraestrutura Transversal

### Segurança

#### Certificado digital

#### Aquisição do Certificado digital

#### Infraestrutura para o certificado digital

#### Certificado digital e a aplicação

#### Usando o módulo de segurança

#### Autenticação

#### Contas Locais

### Loggers

### Validadores

### Adaptadores

## Camada de infraestrutura de Dados

### Serviços Batch

### Sub-Camada Repositórios

### Sub-Camada EntityConfiguration

### Sub-Camada Agentes de Serviço

### Gestão do Ciclo de Vida da Aplicação

# Auditoria

## Listner para execução da auditoria

## Integração Contínua

## Build por Check-in

### Comentários nas alterações

## Execução de Testes Unitários

### Testes unitários com TDD

## Testes automatizados

## Testes de Carga

## Distribuição

## Deploy Automático

### Publish Profile

### Publish Method

### Web.config Transformation

### Regras de Desempenho

# Ambiente de Produção

## Balancing Server

## Servidor de Aplicação

## ASP.NET State Server

## Regras de Desempenho

# Licenças de Software utilizadas