**Arquitetura de software Prescritiva para Projetos em .Net**

Guia de utilização da arquitetura baseada em DDD

Versão 1.0

# 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-Driven Design (DDD) é uma abordagem para o desenvolvimento de software para as necessidades complexas por conectar profundamente a implementação de um modelo de evolução dos conceitos de negócio. DDD não é uma tecnologia ou uma metodologia. DDD fornece uma estrutura de práticas e terminologia para a tomada de decisões de design que se concentram e acelerar projetos de software que tratam de domínios complicados.

Referência: Livro Domain-Driven Design Tackling Complexity in the Heart of Software, por Eric Evans.

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

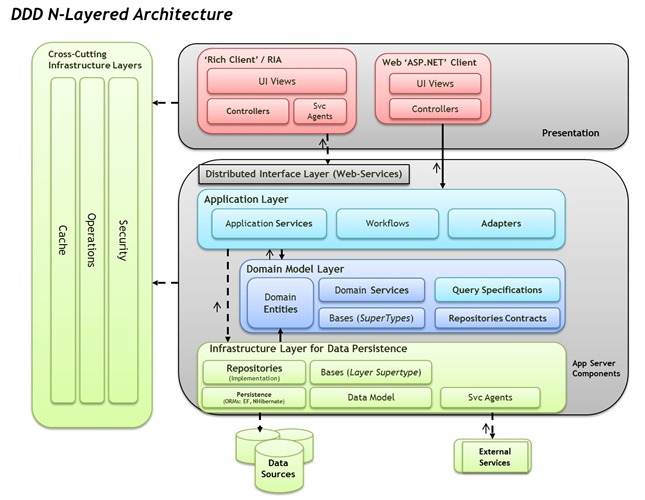
Tecnologias (frameworks) e ferramentas que utilizamos nos projetos web em .Net:

* 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.



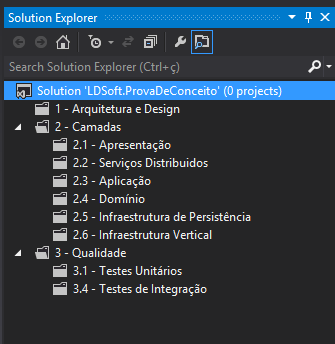
Antes de definirmos as responsabilidades de cada camada, é importante fazer uma relação entre a arquitetura demostrada de forma abstrata no diagrama de camadas acima, com sua implementação no C# e Visual Studio 2012.

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.

## Divisão conceitual do projeto no Visual Studio:

|  |
| --- |
| 1 - Arquitetura e Design  2 - Camadas  2.1 - Apresentação  LDSoft.ProvaDeConceito.UI.Web  2.2 - Serviços Distribuidos  LDSoft.ProvaDeConceito.ServiceHost  2.3 - Aplicação  LDSoft.ProvaDeConceito.Application  2.4 - Domínio  LDSoft.ProvaDeConceito.Domain  2.5 - Infraestrutura de Persistência  LDSoft.ProvaDeConceito.Infrastructure.Data  -> LDSoft.Comum.Domain  2.6 - Infraestrutura Vertical  LDSoft.ProvaDeConceito.Infrastructure  -> LDSoft.Comum.Infrastructure  3 - Qualidade  3.1 - Testes Unitários  3.2 - Testes de Carga  3.3 - Testes de Automatizado  3.4 - Testes de Integração  LDSoft.ProvaDeConceito.Application.Tests  3.5 - Testes de Interface |

|  |
| --- |
| **Atenção** |
| Nome da solução da solução que possui os projetos no Visual Studio deve respeitar a regra de formação: Nome da Empresa "ponto" Nome do Produto  Exemplo: LDSoft.ProvaDeConceito |



As pastas da imagem acima, são as “Solution Folder´s”, pastas virtuais na solução do projeto utilizadas para facilitar o entendimento e a organização, o diagrama demonstrado acima será implementado em cada pasta contida dentro da pasta “2 - Camadas”.

Agora descreveremos cada camada, definindo suas responsabilidades, tecnologias utilizadas e padrões a serem adotados.

## Camada de Apresentação

Responsável por mostrar as informações para os usuários e interpretar seus comandos. A camada de apresentação, ou interface com o usuário, é a porta de entrada ao sistema.

Com uma arquitetura em camadas, é extremamente possível desacoplar a interface com o usuários das demais camadas da aplicação. Neste documento definiremos uma camada de interface voltada para soluções em plataforma web.

Criação do primeiro projeto da Solução "Prova de Conceito". Projeto que representa a camada de apresentação Web; seguindo a regra de criação de nomes: LDSoft.ProvaDeConceito.UI.Web

**UI - User Interface**

No Visual Studio, acesse o menu “File”, selecione a opção “New” -> “Project”.

Na janela “Add New Project”:

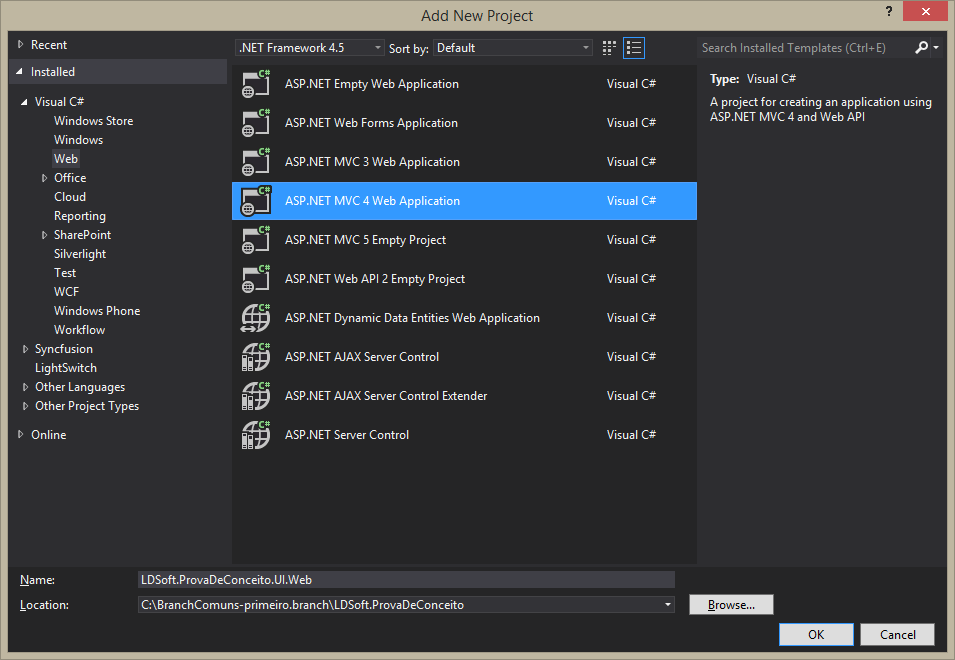
Na janela esquerda, selecione “Visual C#” -> “Web”.

Na janela direita, seleciona o template “ASP.NET MVC 4 Web Application”.

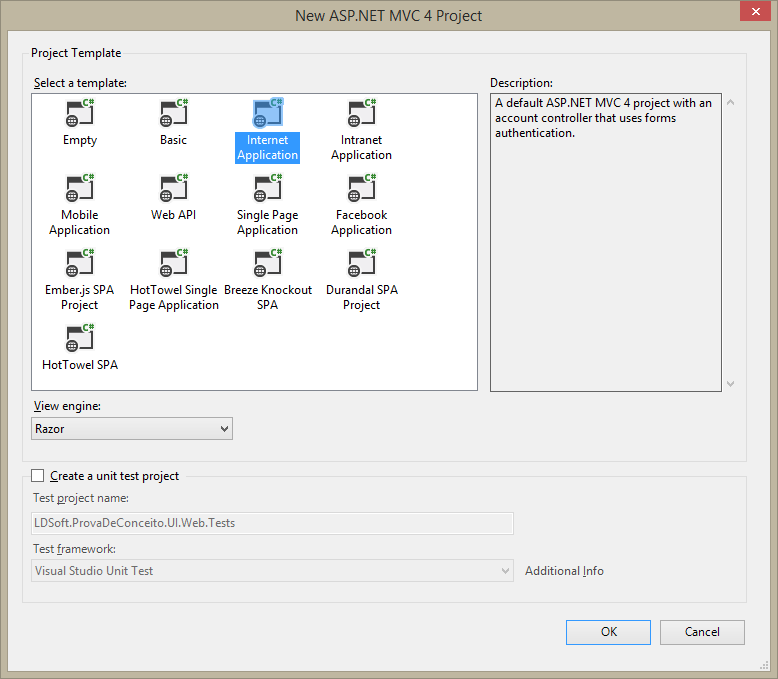
No campo “Nome” digite: LDSoft.ProvaDeConceito.UI.Web

No campo “Location” selecione um endereço local para gravar o projeto.

No campo “Solution name” digite: LDSoft.ProvaDeConceito.



Selecione a opção para utilizar forms authentication



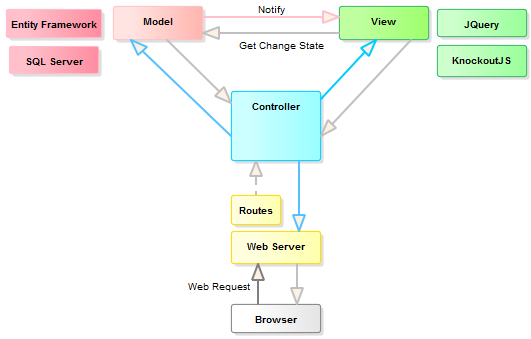
Durante a instalação do template do projeto MVC 4 da Microsoft, já são adicionadas a maioria das dependências externas necessárias para a web, só é preciso atualizar algumas bibliotecas pelo NuGet. Para isso utilizar os seguintes comandos pelo "Package Manager Console" dentro do Visual Studio:

|  |
| --- |
| update-package EntityFramework  update-package jQuery  update-package jQuery.UI.Combined  update-package jQuery.Validation  update-package knockoutjs  update-package Modernizr  update-package Newtonsoft.Json |

### A Arquitetura MVC

A arquitetura adotada para a implementação da interface com o usuário na plataforma web, é o Model-View-Controller. É um modelo considerado uma arquitetura padrão para plataforma web, ele isola a lógica da aplicação da interface com o usuário.

Referência: https://www.microsoftvirtualacademy.com/en-US/training-courses/introduction-to-asp-net-mvc-8322



### .Net Framework MVC 4

Framework MVC para implementação do padrão M-V-C em plataforma .Net.

Referência: http://www.asp.net/mvc

#### Razor Syntax

Linguagem de interface (conhecida como View Engine) utilizada para escrever a View.

Referência: <http://www.asp.net/web-pages/overview/getting-started/introducing-razor-syntax-(c)>

#### A View

Esta é a camada mais externa da aplicação, é nesta camada que implementaremos toda a codificação client-side, ou seja, que é executada no cliente, neste caso, no navegador. É aqui que implementaremos HTML, Javascript, e Razor Syntax.

#### Pasta Content

Pasta responsável por guardar os arquivos CSS, e as imagens que compõem o HTML.

#### Pasta Scripts

Responsável por guardar os arquivos de extensão “.js”, inclusive os scripts dos frameworks JQuery e JQuery UI. Todo arquivo de script deve estar acompanhado de um arquivo com o sufixo “–vsdoc.js”; este arquivo possui os comentários dos métodos implementados nos arquivos de scripts que deverá possuir a seguinte estrutura:

|  |
| --- |
| /// <summary>  /// Descrição do Método  /// </summary>  /// <param name="selector" type="String">  /// Descrição sobre a funcionalidade do parâmetro  /// </param>  /// <returns type="jQuery" /> |

#### Pasta Views

Esta pasta contém as telas do sistema, nela implementamos o HTML, o Javascript e Razor Syntax. Essas páginas possuirão a extensão .cshtml e deverão possuir o nome com semântica compatível com seu objetivo, seguindo a lei de formação NomeComSemânticaNomeDoController, por exemplo, para o Controller “Cliente” a view que retorna o filtro do Cliente deverá se chamar “ListarCliente”.

Em aplicações maiores, que tenham um crescimento exponencial das views, será necessário fazer uso das “Areas” que permite agrupar nossa aplicação por módulos.

Referência: https://msdn.microsoft.com/en-us/library/ee671793(v=vs.100).aspx

#### WebConfig e AppSettings

As chaves de configuração definidas na área AppSettings devem estar acompanhadas de comentário de sua função e as opções possíveis destas chaves. Estas chaves também deverão estar documentadas no Pacote de Aplicação.

Exemplo de configuração para conexão com banco de dados:

|  |
| --- |
| <add name="ProvaDeConceitoContext" connectionString="Data Source=Netuno; Initial Catalog=ProvaDeConceito; Persist Security Info=True;User ID=sa; Password=5XdR!#6?; MultipleActiveResultSets=True" providerName="System.Data.SqlClient" /> |

#### Global.sax

O arquivo Global.asax é responsável pelas requisições globais em uma aplicação .net, no caso do MVC é nele que configuramos as rotas de nossa aplicação, em outras palavras, configuramos qual caminho executará qual ActionResult.

#### O Controller

Esta camada é responsável por fazer o controle de fluxo da aplicação, as requisições são feitas nas ActionResult que são implementadas no Controller. O Controller conhece as demais camadas do sistemas, podendo apenas ele acessar estas camadas e repassar apenas as informações que a View realmente necessita.

[IMAGEM da implementação do controller na POC]

#### O ViewModel

O ViewModel são as classes que representam a View, dessa forma tudo que a View precisa mostrar ou capturar ao usuário estão representadas na ViewModel. Assim, não precisamos levar para a View toda a Entidade de Domínio de nossa aplicação, garantindo o desacoplamento necessário.

[IMAGEM exemplo de código de uma view model na POC]

A utilização do ViewModel dentro do MVC consiste em adaptar outra arquitetura a Model-View-ViewModel (MVVM), baseada em Data Binding. Para qua a adaptação desse conceito funcione na nossa aplicação web, usaremos os frameworks KnockoutJs e JQuery.



#### O Profile Mapper

A pasta Profile Mapper será utilizada com o fim de informar o Mapeamento de tradução entre uma Entidade e uma ViewModel, também conhecido como adaptador (Adapter Pattern). As classes de adaptadores podem ser criadas como Extensions Methods. Aumentado significativamente a produtividade. O framework utilizado para o mapeamento será o AutoMapper. Que pode ser adicionar na aplicação pelo NuGet.

Referência: <https://github.com/AutoMapper/AutoMapper>.

### HTML 5

Versão final do html, ideal para aplicações cross-browser.

Referência: <http://www.w3.org/TR/html5/>

### JavaScript e Fameworks

Quando queremos criar sites responsivos, de fácil manutenção, entre outras funcionalidades, é muito difícil e trabalhoso de se fazer manualmente e é aí que entram os frameworks, facilitando o nosso trabalho.

#### 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

Suite de componentes de interface com usuários, quem melhora a experiência de uso do sistema, utiliza o framework JQuery. <http://jqueryui.com/>

#### KnockoutJS

Knockout é uma biblioteca de JavaScript que o ajuda a criar telas ricas e responsivas. Toda vez que você tem seções de interface do usuário que a atualização é dinâmica (por exemplo, mudar dependendo das ações do usuário ou quando muda a fonte de dados externas), podem ser implementadas de uma maneira mais simples e fácil com o Knockout.

Refrência: http://knockoutjs.com/

Exemplo de Observer Pattern:

...

#### AngularJS

<https://angularjs.org/>

## Camada de Serviços Distribuídos

A camada de distribuição de serviços visa ser utilizada quando for necessário, distribuir serviços da aplicação de forma Interoperável. Esta camada nem sempre existirá em um projeto, entretanto quando algum outro sistema externo necessitar consumir alguma informação deste sistema, a camada de Serviços Distribuídos será criada com o objetivo de implementar recursos técnicos que permitam a interoperabilidade entre diversas arquiteturas.

Esta camada se assemelha a arquitetura SOA (Service Oriented Architecture), implementado mais especificamente a camada de Mensageria desta arquitetura.

O WCF será a solução técnica utilizada nesta camada.

### Profile Mapper

Será o adaptador, entre as entidades de domínio e as classes Contrato.

## Camada de Aplicação

A camada de Aplicação, seguindo as tendências da arquitetura DDD, deve ser uma fina camada que coordena atividades de aplicação, essa lógica, é a lógica de minha aplicação e não a lógica de negócio. Normalmente os serviços nesta camada coordenam serviços de camadas inferiores.

Para facilitar o entendimento, suponha que sua aplicação precise de um xml com os dados do cliente, com suas compras e vendas. Fornecer os dados do cliente com suas devidas compras e vendas e saldo calculado, é responsabilidade da camada de domínio, entretanto disponibilizar o esses dados em um formato de xml, é responsabilidade da sua aplicação.

[3.1.ClassLibraryApplicationProjeto.png]

[3.2.PastasDoProjetoApplication.png]

Toda aplicação precisa ter ciência do usuário logado, essa não é uma responsabilidade da camada de domínio, pois em um negócio não existe o conceito de usuário logado, usuários logados só existem em aplicação de sistemas de informação, por isso é uma responsabilidade da camada de aplicação. O código Abaixo mostra um serviço de aplicação, chamando um serviço de domínio.

[IMAGEM implementado a camada de aplicação]

### Não fazer na camada de aplicação:

A lógica da aplicação não deve incluir qualquer lógica de domínio, apenas tarefas de coordenação relativas a requisitos técnicos da aplicação, conversão de formatos de dados de entrada para entidades de domínio, componentes de infraestrutura exige a execução de tarefas complementares (e-mail confirmações, documentos de exportação para formatos de escritório, etc.)

### 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