FUNDAÇÃO DE ENSINO “EURÍPIDES SOARES DA ROCHA”
CENTRO UNIVERSITÁRIO EURÍPIDES DE MARÍLIA – UNIVEM
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

ANTONIO MIGUEL BATISTA DOURADO

O USO DE METODOLOGIAS ÁGEIS NO DESENVOLVIMENTO DE SOFTWARE COMO SERVIÇO: BASE BIBLIOGRÁFICA DIGITAL

MARÍLIA
2009

ANTONIO MIGUEL BATISTA DOURADO

O USO DE METODOLOGIAS ÁGEIS NO DESENVOLVIMENTO DE SOFTWARE COMO SERVIÇO: BASE BIBLIOGRÁFICA DIGITAL

Trabalho de Curso apresentado ao Curso de Bacharelado em Ciência da Computação da Fundação de Ensino "Eurípides de Soares da Rocha", mantenedora do Centro Universitário Eurípides de Marília – UNIVEM, como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.

Orientador:
Prof. Ms. ELVIS FUSCO

MARÍLIA
2009

Dourado, Antonio Miguel Batista

O Uso de Metodologias Ágeis no Desenvolvimento de Software como Serviço: Base Bibliográfica Digital / Antonio Miguel Batista Dourado; orientador: Elvis Fusco. Marília, SP: [s.n], 2009.

87 f.

Trabalho de Curso (Graduação em Ciência da Computação) – Curso de Bacharelado em Ciência da Computação, Fundação de Ensino "Eurípides Soares da Rocha", mantenedora do Centro Universitário Eurípides de Marília – UNIVEM, Marília, 2009.

1. Web 2.0 2. Metodologias Ágeis 3. OpenUP 4. MVC 5.Subsonic

CDD: 005.1

CENTRO UNIVERSITÁRIO EURÍPIDES DE MARÍLIA
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

TRABALHO DE CONCLUSÃO DE CURSO – AVALIAÇÃO FINAL

Antonio Miguel Batista Dourado

O USO DE METODOLOGIAS ÁGEIS NO DESENVOLVIMENTO DE SOFTWARE COMO SERVIÇO: BASE BIBLIOGRÁFICA DIGITAL

Banca examinadora da monografia apresentada ao Curso de Bacharelado em Ciência da Computação do UNIVEM/F.E.E.S.R., para obtenção do Título de Bacharel em Ciência da Computação.

Nota: 9,5 (nove e meio)

Orientador: Elvis Fusco

1°. Examinador: Fabio Lucio Meira

2°. Examinador: Leonardo Castro Botega

Marília, 01 de dezembro de 2009.

Dedico este trabalho primeiramente a Deus, a quem eu recorri todas as vezes que o desanimo e a tristeza ocuparam minha mente.

Aos meus pais e amigos pelo incentivo constante.

AGRADECIMENTOS

Novamente agradeço a Deus em primeiro lugar por jamais ter me deixado perder as forças e animo em continuar o curso e também por ter me fornecido o essencial para eu chegar onde cheguei até agora, a vida.

Agradeço aos meus pais, Francisco e Mariza, por terem me educado de forma correta em relação à postura que tenho diante das situações, obrigações e responsabilidades da vida bem como terem me apoiado cada minuto que passei estudando, por terem ouvido minhas inúmeras reclamações e por terem me aconselhado sempre.

Agradeço ao meu orientador, Elvis Fusco, pelo apoio e orientação durante o curso bem como conselhos de ordem acadêmica e profissional além da amizade.

Agradeço aos meus amigos de faculdade Naieli, Marco Antonio e Danilo César por todos esses anos de companheirismo, luta, trabalhos, provas, risadas, bebidas e até mesmo brigas, pois toda amizade que se preze também acaba havendo brigas onde tudo sempre acaba bem.

Agradeço ao pessoal do LAS (Laboratório de Arquitetura de Sistemas) pelos dias de estudo e correria passados nessa reta final.

Agradeço também aos demais amigos de faculdade que estão finalizando o curso assim como eu e também aqueles que já deixaram o curso pelo companheirismo e ajuda.

Agradeço a banda Iron Maiden por ter composto todas as musicas que escutei durante os quatro anos do curso enquanto estudava para as provas, fazia os trabalhos dos cursos, fazia o trabalho de conclusão de curso e escrevia esta monografia.

Agradeço-os também por terem proporcionado através de seus shows os dois melhores dias da minha vida, 02/03/2008 e 15/03/2009, e que me inspiraram de modo geral.

Agradeço a empresa Skepsys que desde 2006 vêm me apoiando profissionalmente e dessa forma contribuiu em grande parte do que sou hoje em termos de conhecimento e responsabilidade.

Agradeço ao Seiya dos Cavaleiros do Zodíaco por nunca deixar de acreditar que venceria e pelas vitórias no Coliseu, nas Doze Casas do Zodíaco, na Saga de Asgard, na saga de Poseidon e finalmente na Saga de Hades, sempre elevando seu cosmo além do imaginável para vencer seus oponentes.

Agradeço aos meus cachorros Nina, Scooby e mais recentemente Niquita por sempre terem ouvido minhas lamentações sem jamais reclamar e por sempre estarem do meu lado quando precisei me distrair brincando com eles.

Agradeço ao pessoal do ônibus Garça-Marília da UNIVEM pelos excelentes momentos durante estes quatro anos onde pude cantar até ficar rouco, rir até doer a boca e também chamado a atenção até não ligar mais.

Agradeço aqueles que já fizeram parte de minha vida de alguma forma e que por quaisquer motivos não mais estão presentes nela, mas me ajudaram e apoiaram enquanto presentes.

E por fim agradeço aqueles que duvidaram e/ou invejaram meus avanços durante o curso, pois estes também me ajudaram a ter forças para provar que é nas adversidades que as pessoas também criam forças para continuar.

Obrigado!

“Pain is temporary, glory lasts forever.”

Mark Allen

DOURADO, Antonio Miguel Batista. **O Uso de Metodologias Ágeis no Desenvolvimento de Software como Serviço: Base Bibliográfica Digital.** 2009. 87 f. Trabalho de Curso (Bacharelado em Ciência da Computação) – Centro Universitário Eurípides de Marília, Fundação de Ensino "Eurípides Soares da Rocha", Marília, 2009.

RESUMO

A Web como um ambiente em constante evolução, evoluiu do modelo "Web 1.0" com suas páginas estáticas, sem interação em um modelo unidirecional, para a "Web 2.0" que oferece maior colaboração entre os agentes construtores do conhecimento. Nessa evolução, o paradigma de desenvolvimento de software também está mudando e muitas empresas estão migrando suas aplicações ou fazendo versões das mesmas para a Web. Software como Serviço e Computação nas Nuvens são modelos de aplicações dessa nova geração de aplicativos da Web 2.0 e estão se tornando tendências. Juntamente com o crescimento de aplicações Web 2.0, as metodologias ágeis de desenvolvimento têm sido muito discutidas e utilizadas devido às suas promessas de melhores resultados no desenvolvimento de software. O objetivo deste trabalho é desenvolver, por meio do uso de metodologias ágeis, um software como serviço web a fim de estudar a viabilidade do uso destas metodologias neste modelo de aplicação. O software como serviço, chamado "Base Bibliográfica Digital", pretende funcionar como um serviço colaborativo em que os usuários poderão criar e compartilhar bases e itens bibliográficas.

Palavras-chave: Web 2.0. Metodologias Ágeis. MVC. OpenUP. Subsonic.

DOURADO, Antonio Miguel Batista. **O Uso de Metodologias Ágeis no Desenvolvimento de Software como Serviço: Base Bibliográfica Digital.** 2009. 87 f. Trabalho de Curso (Bacharelado em Ciência da Computação) – Centro Universitário Eurípides de Marília, Fundação de Ensino "Eurípides Soares da Rocha", Marília, 2009.

ABSTRACT

The Web as an evolving environment, has evolved from "Web 1.0" model with its static pages without interaction in a unidirectional model to the "Web 2.0" which offers better collaboration among knowledge constructors agents. With this evolution, the software development paradigm is also changing and many enterprises are migrating their applications or making versions of them for the Web. Software as Service and Cloud Computing are application models from this new Web 2.0 applications and they are becoming tendencies. As the Web 2.0 applications are growing, the development agile methodologies have been widely discussed and used due its promises of better results in software development. The main goal of this work is develop, through the use of agile methodologies, a web software as service with purpose of study the viability with purpose of study the viability of the use of these methodologies in this application model. The software as service, called "Base Bibliográfica Digital", pretends to work as a collaborative service where users will be able to create and share bases and bibliographic items.

Keywords: Web 2.0. Agile Methodologies. MVC. OpenUP. Subsonic.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

PC: Personal Computer

RSS: Really Simple Syndication

SaaS: Software as Service

HTML: HyperText Markup Language

CSS: Cascading Style Sheets

DOM: Document Object Model

XML: eXtensible Markup Language

AJAX: Asyncronous Javascript and XML

XP: eXtreme Programming

CLR: Common Language Runtime

MVC: Model View Controller

DCMI: Dublin Core Metadata Initiative

DCMES: Dublin Core Metadata Element Set

SGBD: Sistema Gerenciador de Banco de Dados

LISTA DE GRÁFICOS

LISTA DE QUADROS

SUMÁRIO

INTRODUÇÃO

Por muitos anos, o cenário mundial de desenvolvimento de software foi voltado ao desenvolvimento de aplicações *Desktop*. Os aplicativos eram instalados localmente em cada computador e executados a partir do mesmo e, dessa forma, estes aplicativos só podiam ser utilizados através do acesso físico à maquina onde estavam instalados. Com o passar dos anos e a difusão da internet, um novo paradigma de aplicação passou a ser estudado: as aplicações web. A visão de uma aplicação que independesse do acesso físico a um computador específico, que pudesse ser acessada de qualquer computador independentemente de seu sistema operacional e ainda a facilitação de atualização e manutenção da mesma atraiu a atenção das empresas de desenvolvimento de software e atrai cada vez mais nos dias atuais.

As primeiras ditas "Aplicações Web" surgiram no que era considerado como "Web 1.0" e com aumento destas aplicações, foi inevitável a evolução da Web 1.0. Surgiu então a "Web 2.0" com novos horizontes e possibilidades de expansão e, graças a esta evolução, muitas empresas migraram e/ou estão migrando suas aplicações para a Web.

Paralelas com a evolução web, novas tecnologias também tem evoluído de maneira rápida devido a pesquisas, experimentos, estudos, etc. São novas linguagens de programação com enfoque em agilidade, novos *frameworks* que procuram deixar cada vez menos codificação nas mãos do desenvolvedor, novas arquiteturas de computadores para maior desempenho e outras várias novidades no ramo da tecnologia. E a área de tecnologia é apenas uma área no meio de milhares de áreas do conhecimento que evoluem dia após dia. Todas essas evoluções, através das atividades de pesquisa e afins, geram documentos de grande importância. São monografias, teses, dissertações, artigos, publicações, etc. documentos que agregam conhecimento para aqueles que os lêem. Porém esses documentos frequentemente têm seu acesso restrito a uma pequena comunidade e/ou grupo de pessoas e, muitas vezes, esta restrição acontece por questões geográficas, ou seja, um artigo criado na cidade de Natal no Rio Grande do Norte pode nunca chegar ao conhecimento de um pesquisador da cidade de Porto Alegre no Rio Grande do Sul caso este artigo não seja publicado em algum lugar onde ambos tenham acesso como por exemplo repositórios institucionais e bibliotecas digitais.

A necessidade de um "local" que reúna estes tipos de documentos é clara. Com um "local" onde os autores possam disponibilizar seus trabalhos direta ou indiretamente (diretamente através da disponibilização do documento em si e indiretamente disponibilizando o registro da existência do documento bem como seu registro bibliográfico),

o fluxo de disseminação da informação e conhecimento se tornaria maior. Este “local” seria como uma base centralizada de bibliografias.

O objetivo deste trabalho é a demonstração do processo de desenvolvimento de um Software como Serviço (SaaS) embasado por princípios da Web 2.0 por meio do uso de uma metodologia ágil em conjunto com *frameworks* de implementação.

O serviço a ser desenvolvido, denominado “Base Bibliográfica Digital”, será utilizado como uma base de bibliografias temáticas tendo conceitos de Software como Serviço, Inteligência Coletiva, Tagsonomia e Colaboração, bases da Web 2.0, onde qualquer pessoa poderá utilizar o serviço para buscar e/ou adicionar novos itens bibliográficos de forma colaborativa além de poder disponibilizar para download o documento de sua obra não importando onde esteja localizada geograficamente. Por exemplo, quando um aluno de ensino superior está fazendo seu trabalho de conclusão de curso, é comum que peça auxílio de seu orientador para localizar bibliografias que o ajude a escrever sobre seu tema. Com o serviço “Base Bibliográfica Digital”, ao invés do orientador ter a preocupação de buscar tais bibliografias para indicar ao aluno, ele poderá simplesmente criar sua base bibliográfica e adicionar as bibliografias e solicitar ao aluno para acessar sua base bibliográfica do serviço. Assim, haverá a facilidade de outros alunos encontrarem bibliografias sobre o assunto sem que o orientador tenha que sempre procurar as bibliografias novamente.

Espera-se que o serviço desenvolvido centralize, de forma colaborativa, as informações sobre documentos resultantes de pesquisas, estudos, experimentos, etc. Além disso, melhorias deverão ser feitas no serviço, sejam elas solicitadas ou necessitadas, de forma que haja sempre melhoria na utilização.

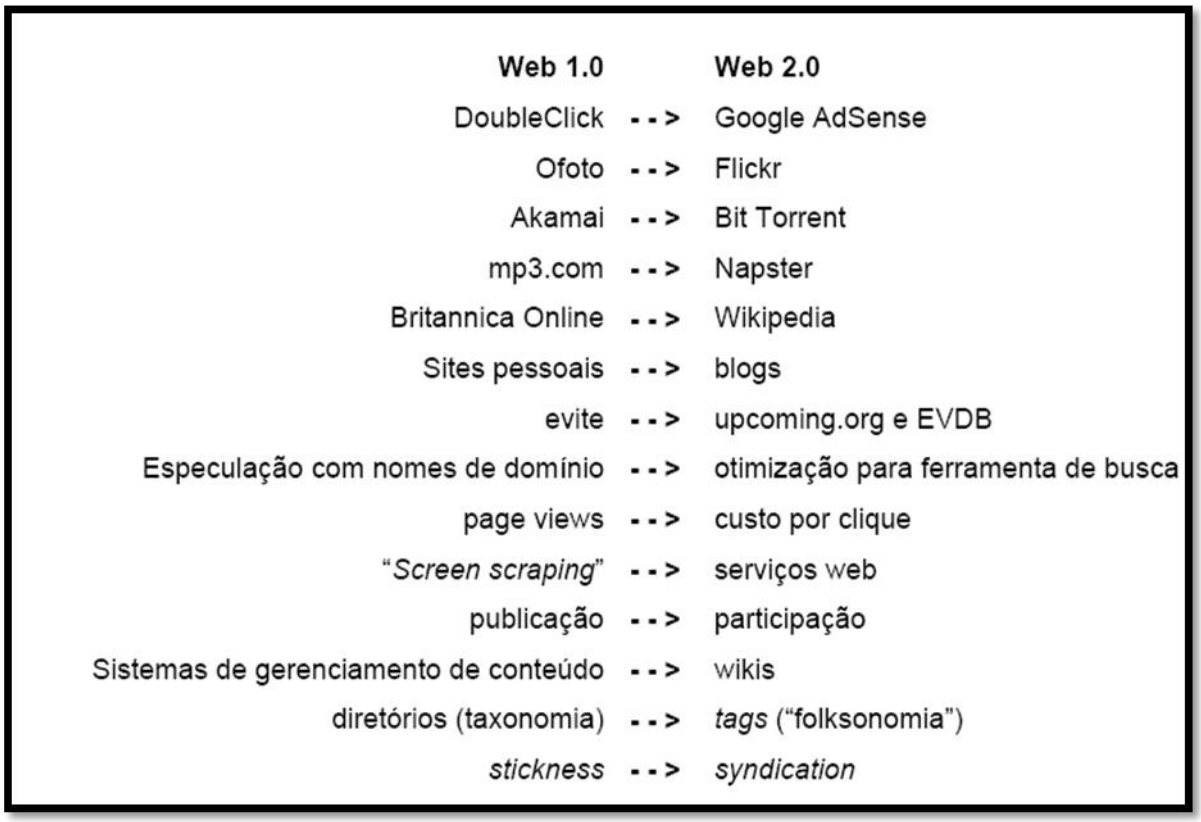
A hipótese é que o desenvolvimento de uma aplicação do tipo Software como Serviço web pode ser facilitada com o uso de uma metodologia ágil devido suas características.

Serão utilizadas metodologias e ferramentas que auxiliarão o desenvolvimento deste serviço web. Estas metodologias e ferramentas, que serão conceituadas no decorrer deste documento, são a metodologia de desenvolvimento OpenUP, o *framework* de desenvolvimento .NET, o padrão arquitetura de software MVC, o padrão de metadados Dublin Core, o *framework* de persistência de dados Subsonic e o núcleo do serviço web que é a Web 2.0.

CAPÍTULO 1 – APLICAÇÕES WEB 2.0

O termo "WEB 2.0" surgiu pela primeira vez em 2004 em uma conferência de *brainstorming* entre a O'Reilly e a MediaLive International. Neste ponto, foi notado que a web estava ganhando uma importância grandiosa devido às novas aplicações apresentadas pela mesma. Nesse *brainstorming* foi formulada uma idéia de "evolução" da Web 1.0 para a Web 2.0 da forma exibida na Figura 1.

Figura 1 – Lista de Evoluções da Web 1.0 para a Web 2.0



Web 1.0		Web 2.0
DoubleClick	- - >	Google AdSense
Ofoto	- - >	Flickr
Akamai	- - >	Bit Torrent
mp3.com	- - >	Napster
Britannica Online	- - >	Wikipedia
Sites pessoais	- - >	blogs
evite	- - >	upcoming.org e EVDB
Especulação com nomes de domínio	- - >	otimização para ferramenta de busca
page views	- - >	custo por clique
"*Screen scraping*"	- - >	serviços web
publicação	- - >	participação
Sistemas de gerenciamento de conteúdo	- - >	wikis
diretórios (taxonomia)	- - >	*tags* ("folksonomia")
stickness	- - >	*syndication*

Fonte: Adaptado de O'Reilly, 2005 (http://oreilly.com/pub/a/web2/archive/what-is-web-20.html)

Apesar da lista gerada no *brainstorming* ainda havia a dúvida do que realmente diferenciava a abordagem ou a identificação de um aplicativo como "Web 1.0" ou "Web 2.0". Até então muitas empresas estavam utilizando-se da "Web 2.0" como uma jogada de marketing sendo que nem mesmo sabiam do que se tratava tal nomenclatura e tão pouco eram, de fato, Web 2.0.

Segundo Tim O'Reilly (2005): "*É particularmente difícil porque muitas dessas novas empresas viciadas na palavra-chave definitivamente não são Web 2.0 e alguns dos*

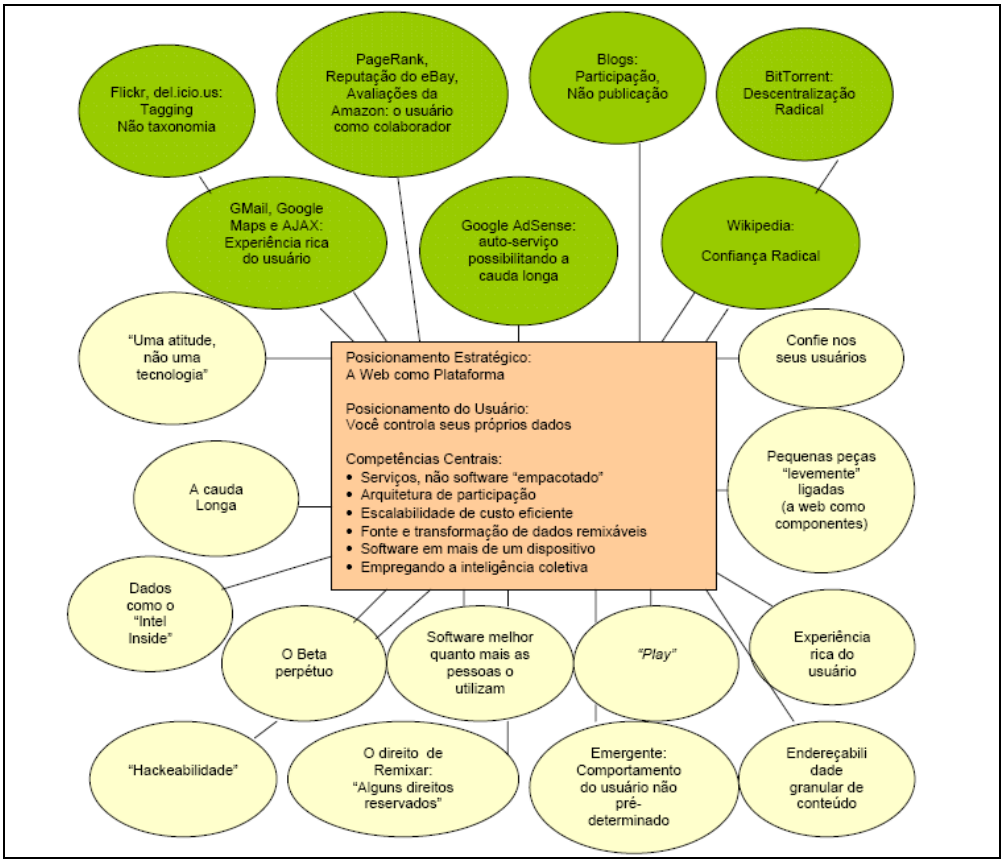
aplicativos que identificamos como Web 2.0, como o Napster e o BitTorrent nem mesmo são verdadeiros aplicativos web!".

Foram definidos então princípios que foram considerados sucesso na Web 1.0 e pelas novas aplicações apresentadas pela web.

1.1 – A web vista como plataforma

As fronteiras da Web 2.0 não são rígidas e a mesma possui um conjunto de princípios e práticas que se interligam através de um centro gravitacional e desta forma pode-se visualizar um "sistema solar" onde tais princípios e práticas tem distâncias variadas em relação ao centro.

Figura 2 – Mapa de noções da Web 2.0



Fonte: Adaptado de O'Reilly 2005

(http://www.oreillynet.com/oreilly/tim/news/2005/09/30/graphics/figure1.jpg)

A Figura 1 demonstra um mapa de noções da Web 2.0 que foi desenvolvido durante uma sessão de *brainstorming* feita na conferência Web 2.0 em outubro de 2004. Através deste mapa pode-se ter uma noção do que está presente na Web 2.0.

Até então a Web era algo estático. Páginas estáticas que se limitavam a exibir conteúdo estático, sem interação alguma.

Em termos de "Web como plataforma" pode-se dizer que as pioneiras foram a DoubleClick e a Akamai. A primeira, DoubleClick (hoje adquirida pelo Google), foi muito conhecida graças aos seus anúncios integrados às páginas web em forma de *banners*. Já a segunda, Akamai, através de computação distribuída utiliza um sistema de cache transparente que "alivia" o congestionamento da banda através da proximidade do usuário com qualquer servidor da Akamai.

Segundo O'Reilly (2005):

> Esses pioneiros forneceram contrastes úteis porque os que entraram depois levaram ainda mais longe suas soluções para o mesmo problema, compreendendo mais a fundo a natureza da nova plataforma. Tanto a DoubleClick como a Akamai foram pioneiras da Web 2.0, entretanto também podemos ver como é possível compreender mais sobre as possibilidades através da adoção de padrões adicionais de design Web 2.0.

Um grande exemplo de diferenciação entre a era da Web 1.0 e a era da Web 2.0 é um comparativo entre a Netscape e o Google. A Netscape definiu a "Web como plataforma" embasada no velho paradigma: através de um aplicativo desktop, seu navegador. A idéia inicial da Netscape era, através do controle dos padrões de exibição do conteúdo web e aplicativos no navegador, ter um poder relativo ao que a Microsoft obteve através de PC's. Além disso, a Netscape planejava promover a venda de seus servidores para provedores de informação através da construção de *Webtops* que seriam povoados por miniaplicativos e atualização de informação. Já o Google desde o inicio se situou como uma aplicação nativamente da Web não sendo jamais empacotada ou vendida. Os usuários dos aplicativos do Google não utilizavam de fato um software, mas sim um serviço fornecido pela empresa através da Web e pagavam direta ou indiretamente pelo serviço. O Google então se voltou aos dados e sua competência é dada pelo gerenciamento de bases de dados. Ainda segundo O'Reilly (2005):

> Sem os dados, as ferramentas são inúteis; sem o software, não se consegue gerenciar os dados. Licença de software e controle sobre os APIs – a alavanca de poder na era anterior – tornam-se irrelevantes porque o software não precisa mais ser distribuído mas apenas executado e também porque sem a habilidade para coletar e gerenciar os dados, o software tem pouca utilidade. Na verdade, o valor do software é proporcional à escala e dinamismo dos dados que ele ajuda a gerenciar.

Dadas as comparações, fica evidente a diferença entre as eras da Web 1.0 e Web 2.0 onde a Netscape pertence à era de empresas como a Microsoft, SAP, Lotus e outras empresas dos anos oitenta enquanto a Google é da era de empresas como Amazon, eBay e outras empresas que assim como as pioneiras Akamai e DoubleClick, são aplicativos situados na Web.

1.2 – Inteligência Coletiva

Pierre Lévy (1998) definiu Inteligência Coletiva:

> É uma inteligência distribuída por toda a parte, incessantemente valorizada, coordenada em tempo real, que resulta em mobilização efetiva das competências. Acrescentemos à nossa definição este complemento indispensável: a base e o objetivo da inteligência coletiva são o reconhecimento e o enriquecimento mútuo das pessoas, senão o culto de comunidades fetichizadas ou hipostasiadas.

O conhecimento é algo detido por bilhões de pessoas no mundo, porém ninguém sabe tudo, mas sim todos sabem algo sobre alguma coisa e desta forma a humanidade como um todo detém o saber. A inteligência é algo de extremo valor e graças aos meios digitais de comunicação esta pode ser compartilhada entre todos de forma a agregar valor mútuo das pessoas.

No contexto da Web, a Inteligência Coletiva representa uma importância de ordem imensa para as empresas web uma vez que a Inteligência Coletiva pode ter sido o principal principio seguido pelas empresas da Web 1.0 que sobreviveram com sucesso da Web 2.0.

Segundo O'Reilly (2005): "*Hiperlinks são o fundamento da rede. À medida que os usuários adicionam conteúdo e sites novos, esses passam a integrar a estrutura da rede à medida que outros usuários descobrem o conteúdo e se conectam a ele.*"

Desta forma pode-se destacar algumas empresas da Web que utilizam-se da afirmação feita por O'Reilly:

- Google: Através do PageRank, o Google possibilitou mediante às buscas dos usuários, melhores resultados nas buscas utilizando-se da estrutura de *links* da rede além de características dos documentos.
- eBay: Coletividade é a palavra chave para o eBay pois o seu crescimento é resultante diretamente da atividade de seus usuários sejam eles vendedores ou compradores.

Muitas outras empresas conseguiram compreender a idéia da Inteligência Coletiva e desta forma surgiram novos seguimentos de aplicações baseadas neste conceito, entre elas:

- Wikipedia: A enciclopédia online, Wikipedia, é hoje um dos cem portais mais acessados do mundo graças à sua estrutura colaborativa com a finalidade de compartilhamento de conhecimento. Usuários podem editar e adicionar novas informações sobre qualquer assunto.
- Del.icio.us: O serviço de *bookmarks* criado em 2003 utiliza-se do compartilhamento de informações e *bookmarks* entre seus usuários e além disso introduz um novo conceito chamado Tagsonomia. A tagsonomia trata-se de uma forma de "etiquetar" conteúdos com palavras chave que também são conhecidas como "*tags"*.

Além das aplicações Web, junto com a Web 2.0 a criação e disseminação dos blogs tem crescido consideravelmente e isso graças a uma nova tecnologia que chegou juntamente com a Web 2.0, o RSS. Com o RSS, as pessoas podem fazer uma assinatura através de um link e serão, utilizando um leitor de RSS, notificadas cada vez que o blog for atualizado. O RSS hoje é comumente usado para outros fins além de um alerta de atualização, ele também é utilizado para bolsas de valores, previsão de tempo, entre outros.

Se a Web 2.0 for imaginada como um grande cérebro que detém incontáveis dados e informações, os blogs podem ser considerados como um "bate-papo" mental onde, graças à Inteligência Coletiva, o conhecimento é disseminado amplamente através de dados.

Todo aplicativo da Web tem ao menos um banco de dados. Google, Yahoo!, eBay, Amazon, todas estas empresas concentram seus dados em um banco de dados. A Web 2.0 é voltada à forma de como os dados serão gerenciados, e mais, muitas empresas enriquecem graças à informação que detém. Um exemplo é NavTeq que diz ter investido valores altos em sua base de dados na construção de indicações de endereços e ruas. Desta forma empresas

como o Google licenciaram esta base de dados e utilizam-se dos mesmos para seus devidos fins. Porém em total aversão à tal prática de "venda de dados", projetos como Wikipedia e Creative Commons são abertos em relação aos seus dados.

1.3 – Software como Serviço

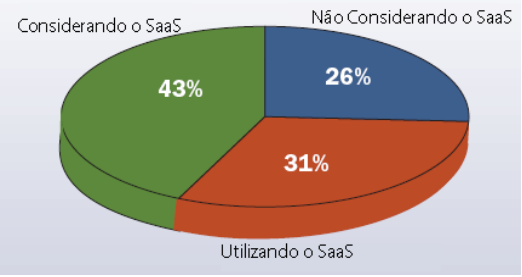
A Web 2.0 fez com que o software deixasse de ser apresentado como um produto para ser então apresentado como um serviço. Esta nova filosofia é conhecida como SaaS (Software as a Service) ou Software como Serviço. O software como serviço tem obtido grande crescimento na adoção nas empresas que desejam abrir mão da preocupação em monitorar a performance do software, implantá-lo, entre outros benefícios oferecidos pela filosofia de software como serviço. Segundo Kaplan (2007):

> Os clientes podem acessar as aplicações SaaS e seus dados através da internet e simplesmente alugar a aplicação de um provedor SaaS por um custo por usuário ou mensal. O provedor SaaS é responsável por entregar, assegurar e gerenciar a aplicação, dados e a infra estrutura básica.

Porém com toda essa despreocupação com o software por parte da empresa uma vez que esta preocupação será delegada ao provedor SaaS, algumas empresas criam preocupações em relação a outro assunto, a segurança de seus dados que agora serão guardados seguramente por terceiros e então surge a questão que decide se a adoção será feita ou não: A empresa está realmente pronta para isso?

Em 2006 foi feita uma pesquisa em relação ao uso ou consideração das empresas sobre software como serviço de acordo com o Gráfico 1.

Gráfico 1 – Sua empresa usa ou consideram uso do software como serviço?



Fonte: Adaptado de Revista Business Communications, 2007
(http://www.wyvilsystems.com/newsap/pdf/saas-friend-or-foe.pdf)

De acordo com o gráfico, 74% dos entrevistados afirmaram que suas empresas consideram o uso ou já utilizam software como serviço.

Com o crescimento da Web 2.0, estima-se que tal percentual cresça e que as empresas adotem cada vez mais a filosofia e uso de software como serviço.

1.4 – Experiência do usuário

A característica mais notável (porém não determinante) da Web 2.0 sem dúvida é em relação às aplicações construídas de forma que a experiência de seus usuários seja rica na utilização das mesmas. Pode-se definir uma experiência rica como sendo a utilização de uma aplicação de forma que a interação entre usuário e aplicação Web seja próxima ou até mesmo semelhante de uma aplicação desenvolvida para ambientes desktop. Para que isso aconteça, são envolvidas tecnologias, linguagens e práticas que juntas fornecem toda a dinâmica para a aplicação. Dentre elas, pode-se citar:

- HTML (HyperText Markup Language): É o coração das aplicações Web, é a linguagem de marcação onde é construída a aplicação com base em *tags* pré definidas.

Quadro 1 – Exemplo de HTML

```
<HTML>
<HEAD></HEAD>
        <BODY> Olá Mundo! </BODY>
</HTML>
```

- Javascript: Linguagem de programação interpretada pelas páginas HTML e que tem a capacidade de interagir com a mesma.

Quadro 2 – Exemplo de Javascript

```
<script language=”javascript”>
        alert(‘Olá Mundo!’);
</script>
```

- CSS (Cascading Style Sheets): Linguagem de folhas de estilo utilizada para definir como a informação será exibida em relação à estilização.

Quadro 3 – Exemplo de CSS

```
<style type=”text/css”>
        .classecss { color: #AABBCC }
</style>
```

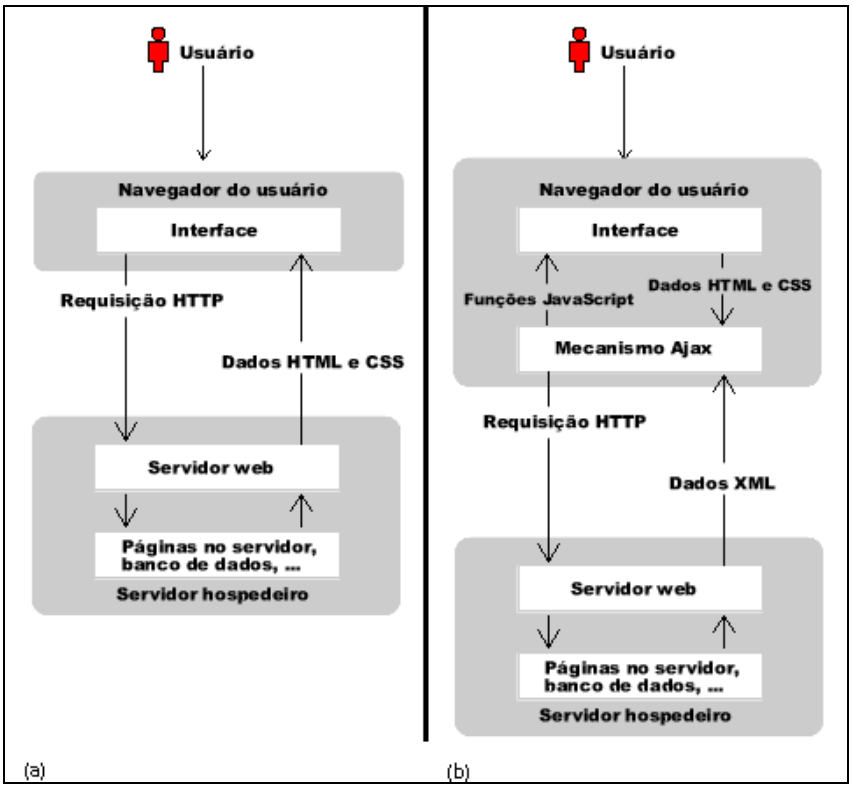
- DOM (Document Object Model): É uma prática onde através do Javascript, HTML e CSS, pode-se alterar conteúdo, estilo e estrutura de um documento.
- XML (eXtensible Markup Language): O XML é uma linguagem de marcação assim como o HTML, porém é amplamente utilizada para transferência de dados e não existe um dicionário de *tags* restrito para a linguagem como existe para o HTML:

Quadro 4 – Exemplo de XML

```
<?xml version=”1.0” encoding=”UTF-8”>
<exemploxml>
        <informacao>
                <nome> Antonio </nome>
                <sexo>Masculino</sexo>
        </informacao>
</exemploxml>
```

- AJAX (Asyncronous Javascript and XML): Para muitos é a palavra chave associada à Web 2.0. O Ajax é a junção de HTML, Javascript, CSS e XML onde através de um objeto XMLHttpRequest, é possível fazer chamadas assíncronas ao servidor, receber a resposta e processá-la de acordo com a necessidade. A Figura 3 realiza um comparativo entre uma requisição comum e uma requisição AJAX:

Figura 3 – Requisição comum e requisição com AJAX.



Fonte: Adaptador de Revista Computer, 2005 (http://xml.csie.ntnu.edu.tw/JSPWiki/attach/Vicky/Building%20Rich%20Web%20Applications%20with%20Ajax.pdf)

Na Figura 3, a requisição comum (a) funciona da seguinte forma: o browser cliente faz a requisição síncrona para o servidor, este processa os dados solicitados pela requisição e ao devolvê-los, o faz juntamente com o código completo da página HTML, ou seja, a página HTML é totalmente recarregada com os dados fornecidos pelo servidor.

Já na requisição feita por AJAX (b), é instanciado, através do Javascript, um objeto XMLHttpRequest ainda no cliente e este faz a requisição assíncrona para o servidor, este processa os dados solicitados e ao invés de devolvê-los em forma de uma página HTML, monta-se um documento XML contendo apenas os dados solicitados e então este XML e apenas ele é devolvido para a página que através do objeto XMLHttpRequest está aguardando a resposta do servidor para sua requisição. Uma vez que o servidor devolve os dados solicitados, o AJAX processa essas informações e através do DOM, atualiza apenas o que é necessário com os novos dados, sem a necessidade de fazer um recarregamento completo da página.

Assim o AJAX é tido como uma revolução em termos de experiência do usuário em relação às aplicações Web pois proporciona agilidade e dinamismo no processamento de dados.

O desenvolvimento da uma aplicação Web 2.0, além das linguagens e tecnologias já citadas, envolve também outros assuntos que serão discutidos mais adiante como a metodologia de desenvolvimento do projeto da aplicação, um padrão de metadados para padronização das informações, um modelo arquitetural, entre outros.

CAPITULO 2 – METODOLOGIAS DE DESENVOLVIMENTO

Segundo Sommerville (2004), as metodologias de desenvolvimento são a representação simplificada do processo de software. Além disso, as metodologias de desenvolvimento ditam práticas e técnicas com a finalidade de obter melhores formas de gerenciar e ordenar o processo através de fases e/ou passos.

Todo projeto de software tem prazos e é comum ter como desafio no contexto do desenvolvimento de software a entrega do produto de forma que esteja dentro do prazo, de acordo com o orçamento previsto e que atenda as necessidades do cliente (FAGUNDES, 2005).

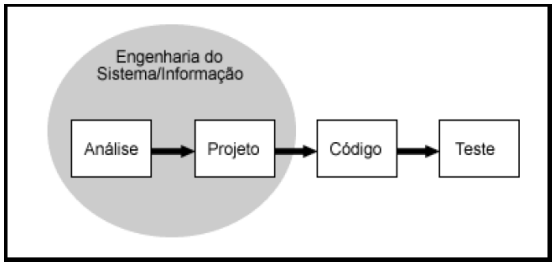
Dois tipos de metodologias de desenvolvimentos se destacam, sendo elas as metodologias tradicionais e metodologias ágeis.

2.1 – Metodologias Tradicionais

Na década de 70 o desenvolvimento de software era feito de maneira desorganizada, sem qualquer planejamento e desestruturadamente, gerando assim um produto final de má qualidade que não atendia as reais necessidades do cliente (PRESSMAN, 2001 apud FELIPPE, 2007). Fazia-se então necessária uma forma de estruturar o processo de software de modo padronizado e planejado. Esta padronização foi inicialmente embasada em conceitos de engenharia civil devido à sua sistemática de processo que se adaptava ao contexto da engenharia de software (NETO, 2004).

Metodologias tradicionais também podem ser conhecidas como metodologias orientadas à documentação e sua principal característica é dividir o processo de desenvolvimento em partes e/ou fases estritamente definidas. Um exemplo de metodologia tradicional é o Modelo em Cascata que pode ser visualizado de acordo com a Figura 4.

Figura 4 – Modelo em Cascata



Fonte: Adaptado de PRESSMAN, 2001

O Modelo em Cascata propõe suas fases de forma que seus requisitos sejam corretamente definidos antes de seguir para a próxima etapa e uma necessidade de mudança de requisitos indica uma falha no processo, ou seja, no Modelo em Cascata a mudança de requisitos não deve acontecer uma vez que estes são definidos e fixados da forma definida. Outra característica deste modelo é o fato de que o cliente somente terá uma versão funcional do produto após o tempo estabelecido para o fim do projeto.

As metodologias tradicionais são apropriadas em casos onde os requisitos são muito bem compreendidos e que não haja necessidades de possíveis mudanças (SOMMERVILLE, 2004 apud FELIPPE, 2007). Em outras palavras, a previsibilidade dos requisitos é necessária ao se utilizar este tipo de metodologia (NETO, 2004).

2.2 – Metodologias Ágeis

Segundo Fagundes (2005), as metodologias ágeis surgiram na década de 90 com o intuito de práticas de desenvolvimento de forma suscetível a mudanças e apoio à equipe de desenvolvimento. Além disso, as metodologias ágeis são uma reação aos modelos de metodologias tradicionais (FOWLER, 2005) com a finalidade de uma alternativa ao Modelo em Cascata (HILMAN, 2004).

Em 2001, dezessete especialistas criaram a Aliança Ágil e através do Manifesto Ágil, popularizou-se então o termo "Metodologia Ágil" (HIGHSMITH, 2002). O Manifesto Ágil é regido pela seguinte filosofia (AGILE MANIFESTO, 2001):

- **Indivíduos e interações** ao invés de processos e ferramentas.
- **Software executável** ao invés de documentação.
- **Colaboração do cliente** ao invés de negociação de contratos.
- **Respostas rápidas a mudanças** ao invés de seguir planos.

Além dos quatro princípios, os membros da Aliança Ágil definiram doze princípios como resultado do refinamento da filosofia do manifesto (FAGUNDES, 2005). Segundo Cockburn (2001) apud Felippe (2007), os doze princípios são:

1. A prioridade é satisfazer o cliente através de entregas de software de valor contínuas e freqüentes.

2. Entregar softwares em funcionamento com freqüência de algumas semanas ou meses, sempre na menor escala de tempo.
3. Ter o software funcionando é a melhor medida de progresso.
4. Receber bem as mudanças de requisitos, mesmo em uma fase avançada, dando aos clientes vantagens competitivas.
5. As equipes de negócio e de desenvolvimento devem trabalhar juntas diariamente durante todo o projeto.
6. Manter uma equipe motivada fornecendo ambiente, apoio e confiança necessários para a realização do trabalho.
7. A maneira mais eficiente da informação circular dentro da equipe é através de uma conversa face-a-face.
8. As melhores arquiteturas, requisitos e projetos provêm de equipes organizadas.
9. Atenção contínua a excelência técnica e a um bom projeto aumentam a agilidade.
10. Processos ágeis promovem o desenvolvimento sustentável. Todos os envolvidos devem ser capazes de manter um ritmo de desenvolvimento constante.
11. Simplicidade é essencial.
12. Em intervalos regulares, a equipe deve refletir sobre como se tornar mais eficaz e então se ajustar e adaptar seu comportamento.

Segundo Highsmith (2002), o Manifesto Ágil não é contra os modelos utilizados pela metodologia tradicional porém não segue os padrões propostos pela mesma. Não rejeita ferramentas, processos, documentação, contratos ou planejamentos mas prioriza indivíduos e iterações, a executabilidade do software, colaboração e feedbacks.

Metodologias ágeis são comumente aplicadas a pequenos projetos e/ou projetos com baixa complexidade utilizando ciclos iterativos, tolerância à mudança, proximidade da equipe, entre outros. Alguns exemplos de metodologias ágeis são o XP (eXtremente Programming), Scrum e OpenUP.

2.3 – OpenUP

Originalmente chamado de BUP (*Basic Unified Process*), o OpenUP foi criado pela IBM com o intuito de uma alternativa ágil ao RUP (*Rational Unified Process*). Em 2005, o BUP foi liberado para a Fundação Eclipse e então renomeado para OpenUP em 2006. Hoje o

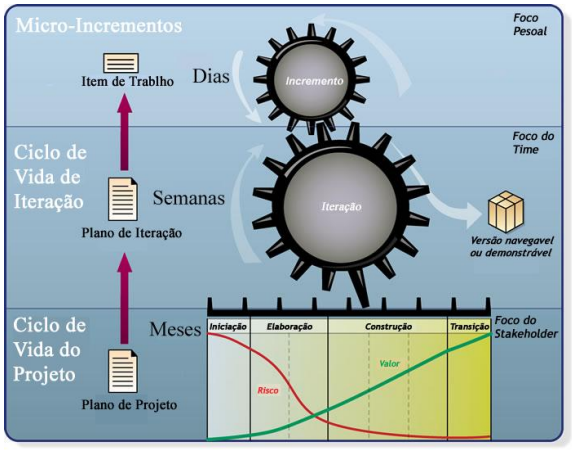
OpenUP é mantido pela mesma Fundação Eclipse e é parte do EPF (Eclipse *Process Framework*).

O OpenUP é um Processo Unificado que visa uma metodologia ágil de desenvolvimento, controle de riscos e melhoria do processo de desenvolvimento de um software através da colaboração. Baseado no Processo Unificado, o OpenUP funciona de forma iterativa e incremental dentro de um ciclo de vida definido sendo possível alterar a forma de seu funcionamento de acordo com as necessidades da empresa, ou seja, o OpenUP é customizável e também extensível podendo-se adicionar funcionalidades à metodologia.

O OpenUP é indicado para pequenas equipes onde a dispersão geográfica é mínima e a interação entre os membros da equipe é alta.

De uma forma geral, o OpenUP pode ser visualizado de acordo com a Figura 5.

Figura 5 - Camadas do OpenUP: micro-incrementos, ciclo de vida da iteração e ciclo de vida de projeto



Fonte: Adaptado de OpenUP, 2008 http://epf.eclipse.org/wikis/openup/publish.openup.base/guidances/supportingmaterials/resources/three_layers.jpg)

2.3.1 – Principios do OpenUP

O OpenUP tem como base quatro princípios básicos (OPENUP, 2008):

- **Equilibrar as prioridades concorrentes para maximizar o benefício aos Stakeholders:** É preciso ter um conhecimento como um todo sobre as necessidades dos stakeholders[1] e dessa forma, alinhar as prioridades que devem ser de total acordo entre as

[1] Stakeholders são, geralmente, pessoas que representam a empresa/instituição que solicitou o projeto.

partes. Além disso, projetar os cenários, casos de uso e escopo do projeto são itens importantes para a definição das prioridades.

- **Colaborar para alinhar os interesses e compartilhar o entendimento:** Em uma equipe cada membro tem seus próprios conhecimentos, habilidades e maneiras de fazer as coisas. Este princípio tem a finalidade de alinhar essas diferenças de forma que o projeto seja beneficiado bem como fazer com que todos os membros da equipe tenham um entendimento sobre o projeto. O continuo aprendizado também é estimulado por este princípio fazendo com que cada membro da equipe desenvolva mais habilidades e incremente seus conhecimentos.

- **Focar na arquitetura, o mais cedo possível, para reduzir o risco e organizar o desenvolvimento:** Uma arquitetura mal planejada muitas vezes é responsável pelos problemas de um sistema e até menos por complicações tanto de manutenção quanto do entendimento do mesmo que podem até levar ao fracasso do sistema e consequentemente do projeto.

Focar na arquitetura é de extrema importância pois muitas vezes os envolvidos no projeto tem visões de formas diferentes arquitetura do sistema. Para evitar tal problema, devem ser utilizados modelos de abstração da arquitetura para que todos os envolvidos tenham uma visualização comum entre si. Além disso, a flexibilidade e reutilização dos recursos são pontos de grande importância para este principio. Componentes de alto acoplamento tendem a dificultar o entendimento do sistema e tem impacto direto na reutilização dos mesmos. O ideal é projetar uma arquitetura onde o acoplamento entre componentes seja baixo, facilitando o reuso dos recursos fornecidos pelos componentes.

- **Evoluir para continuamente obter feedback e promover melhorias:** Conhecer toda a tecnologia e arquitetura do sistema são dois pontos importantes em um projeto porém mudanças podem e sempre ocorrerão em projetos. O objetivo deste princípio é fazer com que através de feedbacks, obtenha-se modos de melhorar o produto e também o processo da equipe envolvida. Através de feedbacks, pode-se identificar potenciais riscos e tratá-los mais cedo durante o projeto. É importante ter o objetivo do projeto de maneira clara ao próprio entendimento para que seja possível fazer uma medição do progresso e identificar possíveis melhorias no processo.

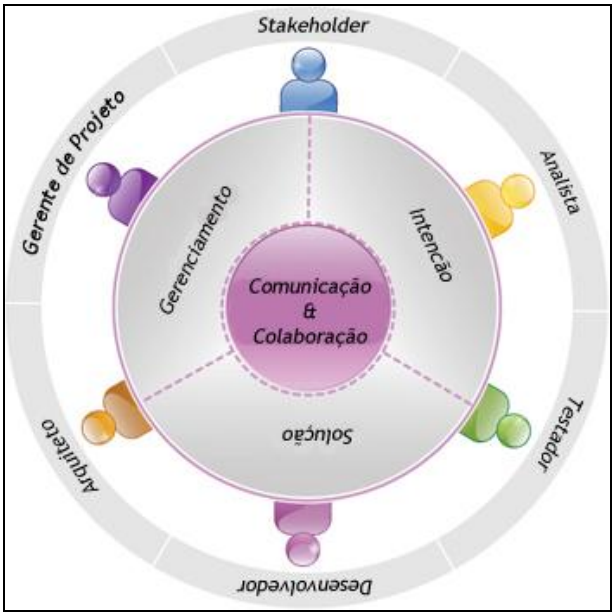
2.3.2 – Papéis do OpenUP

Em um projeto de software diferentes pessoas são envolvidas no processo e cada pessoa tem um papel definido no projeto de acordo com seus interesses e habilidades.

Segundo o OpenUP (2008), os papéis não tem significado de individualidade sobre alguma tarefa ou artefato, mas sim uma identificação de cada envolvido no projeto quando existe o trabalho em conjunto. Além disso, um papel não limita uma tarefa a apenas um indivíduo quando na verdade uma tarefa pode conter vários indivíduos adicionais e inclusive com outros papéis.

Os papéis do OpenUP podem ser visualizado segundo a Figura 6.

Figura 6 – Papéis do OpenUP e interações



Fonte: OpenUP, 2008

(http://epf.eclipse.org/wikis/openuppt/openup_basic/customcategories/resources/OpenUp1_350.jpg)

O OpenUP conta com 7 papéis (OPENUP, 2008):

- **Stakeholder:** Em um projeto, os Stakeholders são os interessados pelo produto final, ou seja, são as necessidades dos Stakeholders que devem ser satisfeitas pelo projeto. Este papel é comumente exercido por um representante da entidade que solicita o projeto.

- **Project Manager / Gerente de Projeto:** O gerente de projeto normalmente é exercido por apenas uma pessoa e esta é tida como líder da equipe envolvida

no projeto. Gerir o projeto, instruir a equipe para obter êxito, avaliação e controle de riscos são habilidades que o gerente de projeto deve ter.

- **Analyst / Analista:** Em constante contato com os Stakeholders, o Analista do projeto tem o objetivo de recolher informações do cliente e usuários finais a fim de compreender o problema a ser resolvido. Com base em informações recolhidas, o Analista deve ter a habilidade de transformá-las em requisitos e definir prioridades. Outra importante habilidade do Analista é a boa comunicação.

- **Architect / Arquiteto:** O arquiteto é o responsável pela definição da arquitetura de software bem como as decisões técnicas que orientam todo o design e implementação do projeto além de fornecer o raciocínio para essas decisões, reduzir riscos técnicos, balancear os interesses dos Stakeholders e assegurar que as decisões sejam devidamente comunicadas, validadas e seguidas.

- **Developer / Desenvolvedor:** Com base na arquitetura definida pelo Arquiteto e a tecnologia utilizada no projeto, o Desenvolvedor deve fazer o desenvolvimento de parte do sistema. Compreender a arquitetura e adaptar-se à ela é essencial para que o Desenvolvedor exerça seu papel com sucesso. O Desenvolvedor também deve ter a habilidade de identificar e gerar testes para que seja validado o comportamento esperado para o que foi desenvolvido.

- **Tester / Testador:** O testador é responsável por toda atividade de testes do projeto. É ele quem realiza os testes do que foi desenvolvido, registra e comunica os resultados. Conhecimento do sistema e arquitetura são desejáveis para este papel. No caso de utilização de ferramenta de testes, o testador deverá deter o conhecimento da ferramenta utilizada.

- **Any Paper / Qualquer Papel:** Apesar de não estar contido no gráfico de papéis acima, o papel "Qualquer Papel" é fornecido pelo OpenUP e este papel permite que qualquer membro da equipe execute tarefas gerais como por exemplo submeter solicitações de alterações no projeto, participar de revisões e avaliações, entre outros.

2.4 – Camadas do OpenUP

O OpenUP de modo geral conta com três camadas que interagem entre si:

1. Micro-Incrementos
2. Ciclo de Vida de Iteração
3. Ciclo de Vida do Projeto.

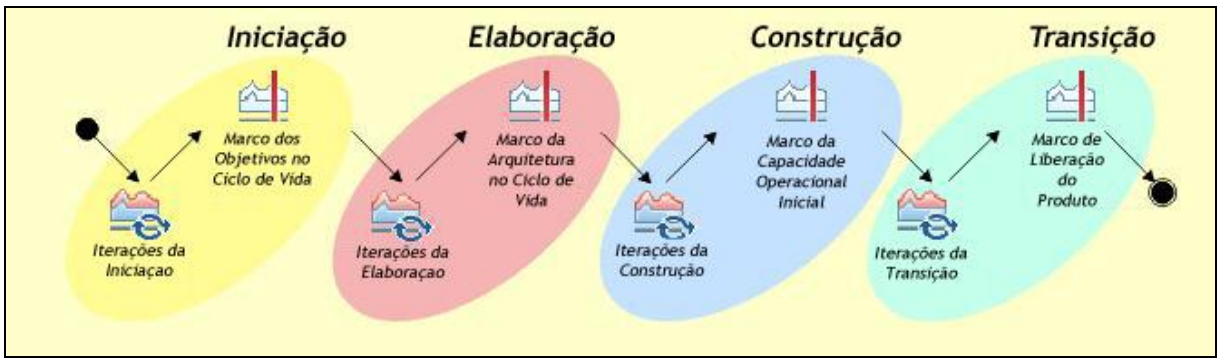
Através dessas interações das camadas, o projeto caminha desde pequenas ações até uma visualização macro do projeto.

2.4.1 – Ciclo de vida de projeto

O Ciclo de vida de projeto no OpenUP dá uma visão geral do projeto a ser desenvolvido bem como o que será realizado, como será realizado e por quem será realizado.

De modo geral, podem-se ver todas as partes do desenvolvimento de um projeto através das fases (Concepção, Elaboração, Construção e Transição) do ciclo de vida de projeto que pode ser visualizado na Figura 7.

Figura 7 – Ciclo de vida de projeto



Fonte: Adaptado de OpenUP, 2008
(http://epf.eclipse.org/wikis/openuppt/openup_basic/guidances/supportingmaterials/resources/openup-basic_lifecycle.jpg)

Cada fase tem um marco(milestone[2]) ao seu final e através deste, é possível entender o que foi realizado na fase. Os marcos tem como finalidade serem utilizados pelos stakeholders na forma de supervisão do projeto bem como tomadas de decisões sobre o mesmo.

[2] Milestones são os marcos de cada fase. São resultados apresentados no final de cada fase.

2.4.1.1 – Fase: Concepção

A primeira fase do ciclo de vida de projeto do OpenUP é a fase de concepção. Segundo o OpenUp (2008), nesta fase é onde são feitos os primeiros contatos entre os interessados no projeto, levantamentos de requisitos, projeções de caso de uso, levantamento e calculo de riscos, escopo do projeto, etc.

Ao final da fase de concepção, seu milestone(também conhecido como Marco dos Objetivos no Ciclo de Vida) deverá satisfazer a necessidade dos stakeholders em relação ao escopo do projeto, cronograma, prioridades, riscos e custos do projeto. Isso pode implicar na continuidade ou no cancelamento do projeto.

2.4.1.2 – Fase: Elaboração

A segunda fase é a fase de Elaboração. Nesta fase os requisitos levantados na fase de concepção são mais detalhados com a finalidade de projetar uma arquitetura base para o desenvolvimento do projeto (OPENUP, 2008). Além disso, nesta fase ainda são calculados e considerados os riscos do projeto assim como na primeira fase.

Alguns exemplos de projeção de arquitetura são: projeção de modelo entidade-relacionamento, projeção de diagrama de classes, projeção de diagrama de atividades, etc.

O marco desta fase (também conhecido como Marco da Arquitetura do Ciclo de Vida) será dado uma vez que a arquitetura for devidamente projetada e aprovada.

2.4.1.3 – Fase: Construção

A terceira fase é a fase de Construção. Esta fase tem como objetivo desenvolver o sistema baseado nos requisitos levantados na primeira fase, Concepção, baseando-se na arquitetura desenhada e diagramada na segunda fase, Elaboração.

Além do desenvolvimento do sistema, nesta fase também são feitos os testes do sistema após o desenvolvimento afim de detectar a maior quantidade possível de erros tanto de codificação quanto de conceito e consertá-los (OPENUP, 2008).

O marco desta fase é alcançado uma vez que o produto já tenha sido desenvolvido e devidamente testado (testes alfa). Além do produto, também pode ser considerado o desenvolvimento de um manual de uso do produto.

2.4.1.4 – Fase: Transição

A quarta e última fase é a Transição. Nesta fase o produto desenvolvido na terceira fase, Construção, estará em sua fase de testes Beta onde os stakeholders avaliarão seu funcionamento de acordo com as necessidades dos usuários. É possível que alguns novos ajustes devam ser feitos bem como correções.

Uma vez obtida a concordância dos stakeholders sobre o produto, o projeto poderá ser encerrado ou ser o início de um novo ciclo de vida do mesmo produto sob novo projeto, que também podem ser melhorias e/ou manutenção, solicitado pelos stakeholders.

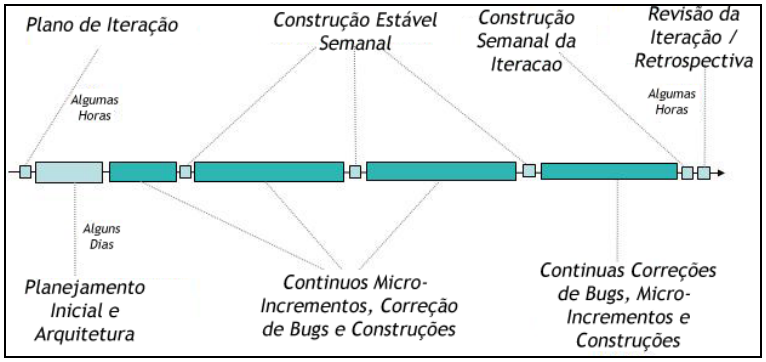
Uma vez que os usuários aceitem o produto e estejam satisfeitos com o mesmo bem como os stakeholders, o marco da fase será alcançado (OPENUP, 2008).

2.4.2 – Ciclo de Vida de Iteração

Segundo o OpenUP (2008): "*Uma iteração é um período de tempo definido dentro de um projeto em que você produz uma versão estável e executável do produto, junto com toda a documentação de apoio, scripts de instalação, artefatos e similares, necessários para usar a versão*.".

O ciclo de vida de uma iteração começa em uma reunião de curto tempo onde é planejada a iteração. Durante a reunião, o Plano de Iteração é construído e este conterá todos os prazos e atividades que deverão ser exercidos durante a iteração. As iterações em sua maioria são compostas de micro-incrementos, ou seja, artefatos validados, códigos executáveis e testados, etc. Ao final da iteração é realizada uma nova reunião com os stakeholders onde os mesmos avaliam o que foi feito e é então discutido se pode haver alguma melhoria no processo (OPENUP, 2008). Na Figura 8 é possível visualizar o ciclo de vida de Iteração.

Figura 8 – Ciclo de Vida de Iteração



Fonte: Adaptado de OpenUP, 2008 (http://epf.eclipse.org/wikis/openup/practice.mgmt.iterative_dev.base/guidances/concepts/resources/iteration_lifecycle.jpg)

2.4.3 – Micro-Incrementos

Quando um membro da equipe realiza um trabalho de curto tempo, este é definido de micro-incremento. Os micro-incrementos são considerados pequenos trabalhos que consomem pouco tempo (horas ou dias) e que contribuem para o crescimento da aplicação como um todo.

Segundo o OpenUP (2008):

> Um micro-incremento deve ser bem definido, e você deve ser capaz de acompanhar diariamente o progresso de cada micro-incremento. Os micro-incrementos são especificados e rastreados por um elemento trabalho. Os conjuntos de mudanças representam o resultado físico em termos dos arquivos que são modificados, como parte da realização de um item de trabalho.

Exemplos de micro-incrementos:

- Definição da visão/escopo.
- Projetar um caso de uso.
- Planejar uma iteração.
- Construir lista de itens de trabalho.

CAPÍTULO 3 – TECNOLOGIAS

Neste capítulo serão conceituadas as tecnologias utilizadas no desenvolvimento do projeto proposto.

3.1 – Frameworks

Segundo Fayad e Schimdt (1997), um *framework* é um conjunto de classes que colaboram para realizar uma responsabilidade para um domínio de um subsistema da aplicação.

Govoni (1999) diz que "Um *framework* é uma coleção de classes, interfaces e padrões com a finalidade de resolver uma série de problemas através de uma arquitetura flexível e extensível".

Sommerville (2004) define como um projeto de susbsistema constituído de um conjunto de partes abstratas e concretas e da interface entre elas.

Em outras palavras, pode-se dizer que um *framework* é um conjunto de classes e serviços que visam facilitar o desenvolvimento de aplicações. Suas classes tem, em geral, o propósito de reutilização, ou seja, não é necessário codificar as mesmas classes/métodos várias vezes.

Existem frameworks para diversas áreas da computação como por exemplo frameworks de arquitetura, frameworks de persistência de dados, frameworks que implementam plataformas de desenvolvimento, entre outros.

3.2 – .NET Framework

O .NET Framework trata-se de um *framework* da Microsoft que é uma camada de execução e desenvolvimento de aplicações de alto nível. Inicialmente funciona sobre o sistema operacional Windows, porém já existem projetos que fazem com que a plataforma funcione sobre ambientes Linux e Unix.

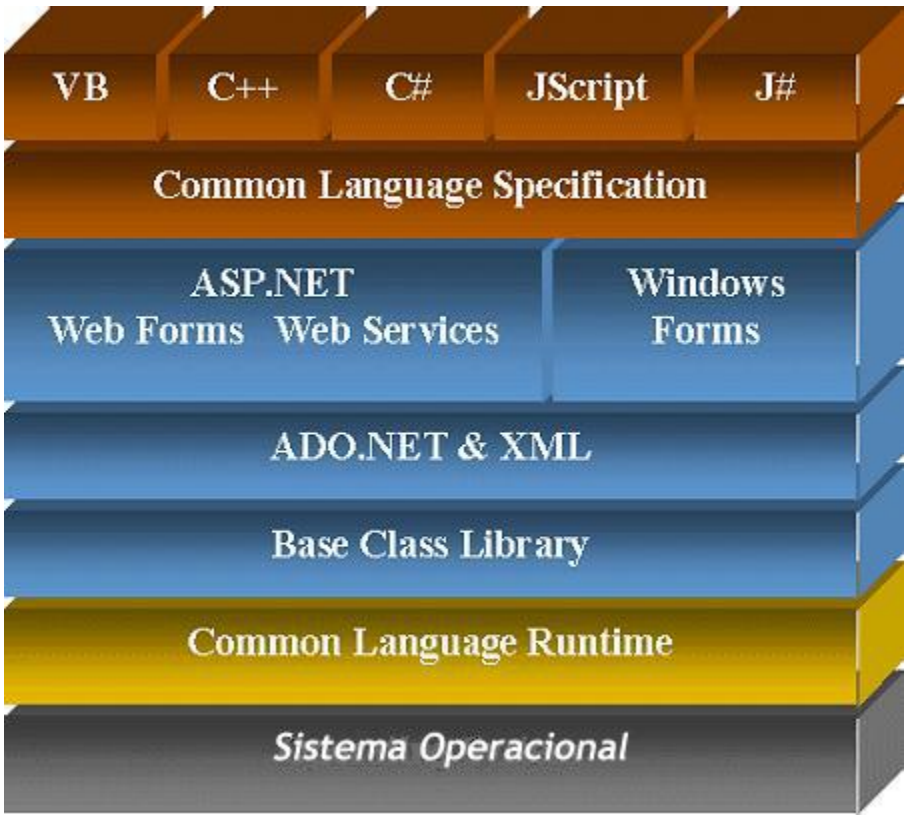
Lotar (2007) define o .NET Framework como sendo um "*componente integrado ao Windows que suporta a execução e o desenvolvimento de uma nova geração de aplicações e XML web services*".

De acordo com a documentação do .NET Framework, este foi projetado com os objetivos que se seguem:

- Disponibilizar um ambiente consistente para programação orientada a objetos de forma que possa ser executado tanto localmente quanto remotamente.
- Disponibilizar um ambiente de execução com a minimização de problemas de implantação e conflitos de versão.
- Disponibilizar um ambiente de execução de código de forma segura independente da origem do mesmo.
- Disponibilizar um ambiente de execução onde não haja problemas de desempenho oriundos de scripts ou ambientes interpretados.
- Aproveitar os conhecimentos do desenvolvedor independente do tipo de aplicação, seja ela Windows Forms ou Web.

Pode-se visualizar o .NET Framework da maneira mostrada na Figura 9.

Figura 9 – Estrutura do .NET Framework



Fonte:Adaptado de MSDN, 2005 (http://msdn.microsoft.com/en-us/library/ms973842.aspx)

A plataforma .NET tem como seus dois principais elementos a CLR (*Common Language Runtime*) e *Base Class Library* (LOTAR ,2007).

A CLR (*Common Language Runtime*) é a camada logo acima do Sistema Operacional, ou seja, é a última etapa do .NET *Framework* antes da execução da aplicação no Sistema Operacional. Assim, pode-se dizer que a CLR é a base do .NET *framework*.

A CLR é responsável por identificar erros no código, gerenciar a execução do mesmo e os recursos utilizados pelo código. Estes recursos fazem com que o ambiente de execução do .NET seja um ambiente gerenciado, ou seja, a CLR gerencia os recursos utilizados pela aplicação. Segundo Lotar (2007), a CLR trata dos seguintes pontos:

- Gerência de memória.
- Verificação de tipos.
- Gerência de exceções.
- Segurança da aplicação.
- Acesso a metadados.

A *Base Class Library* é uma biblioteca composta de interfaces, classes e tipos do .NET Framework onde toda aplicação, controle e componente desenvolvidos neste *framework* é embasada. Através dessa biblioteca pode-se criar aplicações para diversos seguimentos de software. Lotar (2007) cita as principais funcionalidades oferecidas pela *Base Class Library*:

- Tipos de dados e exceções.
- Encapsulamento de estruturas de dados.
- Operações de entrada e saída.
- Acesso às informações sobre tipos de dados carregados.
- Verificações de segurança.
- Acesso à internet.
- Desenvolvimento de interface de uma aplicação.
- Desenvolvimento de aplicações Windows Forms e ASP.NET.

Além disso, o .NET Framework suporta nativamente as linguagens: Visual Basic, C#, Visual C++, Jscript, Visual J# além de linguagens originadas de outras empresas como COBOL por exemplo.

3.2.1 – ASP.NET

De acordo com a documentação do .NET Framework, o ASP.NET trata-se de um modelo de desenvolvimento voltado para a Web. Ao desenvolver uma aplicação utilizando o ASP.NET tem-se acesso às classes do .NET Framework uma vez que o ASP.NET é parte do mesmo *framework*, ou seja, é possível utilizar todas as linguagens suportadas pelo .NET Framework em conjunto com o ASP.NET.

Lotar (2007) diz que a produtividade do ASP.NET é grande e com pouca codificação pode-se construir uma página com certa complexidade.

Através de web server controls, o ASP.NET tem a capacidade de construir complexos layouts com baixa dificuldade. Um exemplo é o controle *GridView* onde, através de um simples clique e arrasto do mouse seguido da configuração de poucas propriedades, é possível construir uma grade de resultados de consultas ao banco de dados. Além do *GridView*, o ASP.NET conta com outros controles como por exemplo controles de validação, controles de login, controles de navegação, controles de formulários, etc.

Outras características do ASP.NET:

- Seu código é compilado, aumentando assim a performance.
- Há gerenciamento de estado da aplicação.
- É configurável através de arquivos XML.
- Suporta depuração.
- Suporte web services.
- Utiliza extensão .aspx ao invés de .html.
- Pode-se utilizar diretivas como @ Page, @ Import, etc.
- Código executando no servidor.

O Quadro 5 mostra um exemplo de uma página ASP.NET.

Quadro 5 – Exemplo de uma página ASP.NET

```
<html>

        <body>

                <form runat="server">

                  <h3>
                        <asp:label id="lbl1" runat="server" />
                  </h3>

                </form>

        </body>

</html>
```

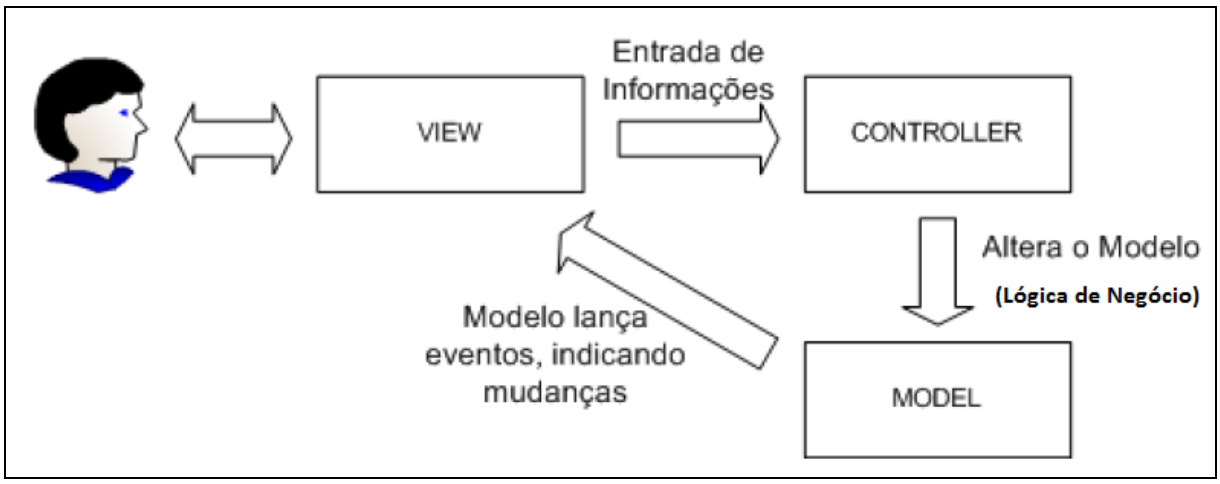
3.3 – MVC

O MVC (*Model-View-Controller*) é um padrão de arquitetura de software. Junior (2009) define padrões de arquitetura de software:

> Padrões para arquitetura de software são soluções de eficiência já comprovadas e amplamente utilizadas para a resolução de problemas comuns em projeto de software. Estas soluções são desenvolvidas e conhecidas por especialistas e tornam-se padrões por serem reutilizadas várias vezes em vários projetos e por terem eficácia comprovada.

Segundo Gamma *et al* (1999), o padrão MVC é composto de três camadas (*Model, View* e *Controller*) onde o Modelo (*Model*) é a camada responsável por controlar os estado da aplicação bem como seus dados, a Visão (*View*) é responsável por apresentar o estado e os dados da aplicação na tela e por fim o Controlador (*Controller*) é responsável por definir como o Modelo irá interagir com a Visão. Gamma *et al* (1999) também diz que o MVC separa essas camadas com o intuito de aumentar a flexibilidade e reutilização. Desta forma, pode-se visualizar o modelo MVC da forma representada na Figura 10.

Figura 10 – Arquitetura do MVC



Fonte: Adaptado de Aula de Arquitetura de Software / IESB, 2009
(http://nelson.junior.br.googlepages.com/aula05-padroes-estilosarquiteturais.pdf)

No entanto, o padrão MVC não é novidade dos últimos anos, Sanderson (2009) diz que este padrão surgiu em 1978 em projetos baseados em Smalltalk. Hoje o padrão MVC tem sido cada vez mais utilizado em aplicações Web provavelmente, ainda segundo Sanderson (2009), pelos dois seguintes motivos:

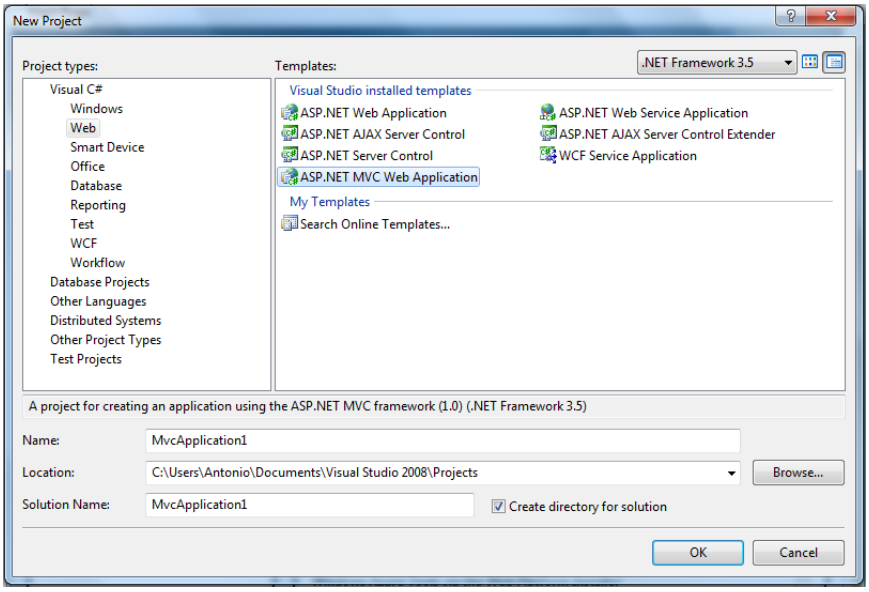
- A interação do usuário com aplicações MVC é um ciclo que geralmente funciona com uma ação do usuário que gera uma alteração nos dados e então o usuário recebe uma visualização alterada destes dados e este ciclo então se repete. Isto se torna conveniente para as aplicações Web que realizam muitas atividades de requisição e resposta.
- Muitas tecnologias são usadas em aplicações Web e, nos tempos atuais, muitos desenvolvedores estão adotando a prática de separação da aplicação em camadas, prática já conceituada pelo MVC.

3.3.1 – ASP.NET MVC

O ASP.NET MVC é um *framework* que implementa o padrão de arquitetura MVC na plataforma .NET. O *framework* foi apresentado em outubro de 2007 e, segundo Sanderson (2009), entre diversos benefícios, o ASP.NET MVC conta com testabilidade, extensibilidade, controle sobre o HTML, um poderoso sistema de roteamento e também este foi construído embasado nas melhores partes da plataforma ASP.NET.

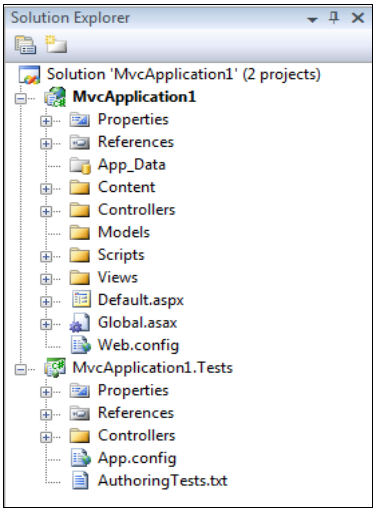
O ASP.NET MVC pode ser baixado no site oficial, http://www.asp.net/mvc, e ao ser instalado, criará um novo *template* para o Visual Studio como mostra a Figura 11.

Figura 11 – Template do ASP.NET MVC no Visual Studio 2008



Ao criar um novo projeto com a opção “ASP.NET MVC Web Application”, já são criados automaticamente os diretórios *Model*, *View* e *Controller* além dos testes unitários. A estrutura deverá ficar da seguinte forma:

Figura 12 – Estrutura do projeto ASP.NET MVC



Já as Visões implementam a visualização dos dados para o usuário, ou seja, nas aplicações Web as Visões são as páginas HTML que exibem os dados em forma de informação.

No Quadro 6 está um exemplo de uma *View* (Visão):

Quadro 6 – Exemplo de uma *View*

```
%@ Page Title="" Language="C#"
MasterPageFile="~/Views/Shared/Site.Master"
Inherits="System.Web.Mvc.ViewPage<List<SubSonic.Web.Models.ItemBi
bliografico>>" %>

<asp:Content ID="Content1" ContentPlaceHolderID="TitleContent"
runat="server">
      Inicio
</asp:Content>


<asp:Content ID="Content2" ContentPlaceHolderID="MainContent"
runat="server">

<form action="<%=Url.Action("index") %>" method="post">
    <input type="text" size="40" id="search" name="q" /><input
type="submit" value="Buscar" />
</form>

</asp:Content>
```

Os Controladores são responsáveis por fazer a ligação entre Modelos e Visões, ou seja, fazer com que os dados fornecidos pelos Modelos sejam exibidos pelas Visões e também que os dados fornecidos pelas Visões sejam armazenados pelos Modelos.

O Quadro 7 exibe um exemplo de um *Controller* (Controlador).

Quadro 7 – Exemplo de um *Controller*

```
public ActionResult Editar()
        {
                var item = _repository.GetByKey(id);
                return View(item);
       }
```

No MVC, os Modelos são encarregados de fazer a ligação da aplicação Web com a base de dados e também realizar toda a regra de negócio. A camada Modelo geralmente é composta de classes que referenciam um banco de dados, classes de mapeamento objeto-relacional e classes que implementam regras de negócio.

No Quando 8 é mostrado um exemplo de um trecho de código de um *Model* (Modelo) do ASP.NET MVC.

Quadro 8 – Exemplo de um *Model*

```
public partial class Busca: INotifyPropertyChanging,
INotifyPropertyChanged {
          public Busca(){
              OnCreated();
          }
        public int KeyValue (){
            return _BuscaID;
        }
            int _BuscaID;
            public int BuscaID {
                get{
                    return _BuscaID;
                }
                set{
                    this.OnBuscaIDChanging(value);
                this.SendPropertyChanging();
                this._BuscaID = value;
                this.SendPropertyChanged("BuscaID");
                this.OnBuscaIDChanged();
                }
            }
```

É na camada de Modelo que os *frameworks* de persistência de dados são utilizados como por exemplo o NHibernate e o Subsonic.

3.3 – Framework de Persistência de Dados

O conceito de persistência no contexto de armazenamento de dados é utilizado no intuito da reutilização dos dados, ou seja, os dados são persistidos, preservados e conservados para serem utilizados novamente em um momento futuro.

Segundo Rollof *et al*:

> No desenvolvimento de software, a persistência pode ficar em uma camada separada, com isso reduzindo a complexidade de desenvolvimento do software em si. Em uma programação orientada a objetos, essa camada de persistência fica responsável pelo recebimento de objetos e o seu armazenamento, de forma que possam ser futuramente recuperados. A forma de armazenamento não importa para a aplicação, quem vai se preocupar com isso é a camada de persistência. Essa camada de persistência pode ser desenvolvida na forma de um framework, ou seja, um conjunto de classes que realizam determinadas tarefas, permitindo a sua fácil reutilização.

Existem muitos *frameworks* de persistência de dados para várias linguagens e/ou plataformas de desenvolvimento, entre eles pode-se citar:

- Hibernate: *Framework* de persistência de dados para aplicações desenvolvidas em Java.
- NHibernate: Desenvolvido pelos mesmos desenvolvedores do Hibernate porém para a plataforma .NET.
- ADO.NET Entity Framework: Realiza a persistência de dados na plataforma .NET.
- Floggy: *Framework* de persistência de dados para a J2ME (Java Micro Edition) em dispositivos móveis.
- Subsonic: Desenvolvido por Rob Conery para a plataforma .NET.

3.3.1 – Subsonic

Com o lema "*All your database belongs to us*" (Todo seu banco de dados pertence a nós), Rob Conery criou o Subsonic.

Trata-se de um *framework* de persistência de dados que realiza o mapeamento objeto-relacional na plataforma .NET. Segundo Ambler (2000), o mapeamento objeto-relacional é a determinação de como os objetos e seus relacionamentos serão persistidos em um mecanismo de armazenamento de dados seguindo um modelo relacional.

Segundo o Subsonic, o foco é que as pessoas tem coisas melhores para fazer com o próprio tempo do que ficar se preocupado com o acesso aos dados e que isso é um problema. Portanto o objetivo do Subsonic é facilitar a vida dos desenvolvedores em relação ao acesso a dados.

3.3.1.1 – Templates Active Record e Simple Repository

O Subsonic pode ser utilizado de duas formas através de dois *templates*: *Active Record* e *Simple Repository*.

O *Active Record* é utilizado pelo próprio Subsonic em seu núcleo como padrão. Este *template* utiliza uma abordagem onde cada objeto representa exatamente uma tabela do banco de dados e cada instância deste objeto representa uma ou mais linhas destas tabelas. A instância é responsável pela persistência dos dados com o banco de dados. Os objetos que referenciam as tabelas são gerados automaticamente.

Algumas características e convenções do *Active Record:*

- Os nomes das tabelas devem estar no singular.
- Existem métodos de auditoria em cada tabela gerada: *CreatedOn*, *CreatedBy*, *ModifiedOn* e *ModifiedBy*.

Figura 13 – Exemplo do *Active Record*

```
//busca um único produto pelo código
var product = Product.SingleOrDefault(x => x.ProductID == 1);

//busca uma lista de produtos baseado em um critério
var products = Product.Find(x => x.ProductID <= 10);

//busca uma lista de produtos páginados pelo servidor
var products = Product.GetPaged(0, 10);

//busca usando o LINQ
var products = from p in Product.All()
               join od in OrderDetail.All() on p.ProductID equals od.ProductID
               select p;
```

Fonte: Adaptado de Subsonic, 2009 (http://subsonicproject.com/docs/Using_ActiveRecord)

O *Simple Repository* utiliza uma abordagem menos direta em relação ao *Active Record* uma vez que não gera todo o código automaticamente. Quando há uma necessidade de aplicações leves, o *Simple Repository* é ideal. Além disso, este *template* tem a capacidade de alterar a estrutura física de tabelas ou até mesmo criá-las em tempo de execução através de seu recurso chamado "*Auto Migrations*".

A única convenção em relação a este *template* é que as tabelas geradas terão seus nomes no plural.

Figura 14 – Exemplo de *Simple Repository*

```
bool exists = repo.Exists<Post>(x => x.Title == "My Title");

//utiliza IQueryable
var qry = from p in repo.All<Post>()
          where p.Title == "My Title"
          select p;

//busca uma postagem
var post = repo.Single<Post>(x => x.Title == "My Title");
var post = repo.Single<Post>(key);

//várias postagens
var posts = repo.Find<Post>(x => x.Title.StartsWith("M"));

//Uma PagedList (lista paginada) de postagens usando 10 por página
var posts = repo.GetPaged<Post>(0, 10);
//ordena pelo título
var posts = repo.GetPaged<Post>("Title", 0, 10);

//adiciona uma postagem
var newKey = repo.Add(post);

//Adiciona várias postagens usando transação
IEnumerable<Post> posts = GetABunchOfNewPosts();
repo.AddMany(posts);

//atualiza postagem
repo.Update(post);
```

Fonte: Adaptado de Subsonic, 2009 (http://subsonicproject.com/docs/Using_SimpleRepository)

3.3.1.2 – T4 Templates

A existência dos *templates* T4 do Visual Studio 2008 é desconhecida por muitos e é um dos grandes "segredos" do Visual Studio 2008, segundo Rob Conery.

T4 significa "*Text Template Transformation Toolkit*", em outras palavras, os templates T4 tem a finalidade de servirem como um modelo em forma de texto para serem posteriormente transformados em código.

Um exemplo de *Template* T4 pode ser visualizado na Figura 15.

Figura 15 – *Template* T4

```
<#@ template language="C#" debug="True" hostspecific="True" #>
<#@ assembly name="System.Data" #>
<#@ assembly name="SubSonic.Mvc.dll" #>
<#@ assembly name="SubSonic.dll" #>
<#@ import namespace="System.Diagnostics" #>
<#@ import namespace="System.IO" #>
<#@ import namespace="System.Data" #>
<#@ import namespace="SubSonic" #>
<#@ import namespace="SubSonic.Mvc" #>

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace Test{

    class Program{

        static void Main(string[] args){



            <#for(int i=0;i<10;i++){#>
            Console.WriteLine("<#=i.ToString()#>");
            <#}#>



        }
    }

}
```

Após a execução do Visual Studio 2008, é gerado então o arquivo exibido na Figura 16.

Figura 16 – Código gerado pelo *template* T4

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace Test{

    class Program{

        static void Main(string[] args){
            Console.WriteLine("0");
            Console.WriteLine("1");
            Console.WriteLine("2");
            Console.WriteLine("3");
            Console.WriteLine("4");
            Console.WriteLine("5");
            Console.WriteLine("6");
            Console.WriteLine("7");
            Console.WriteLine("8");
            Console.WriteLine("9");
        }
    }

}
```

3.3.1.3 – Configuração

A configuração do Subsonic é feita da seguinte forma:

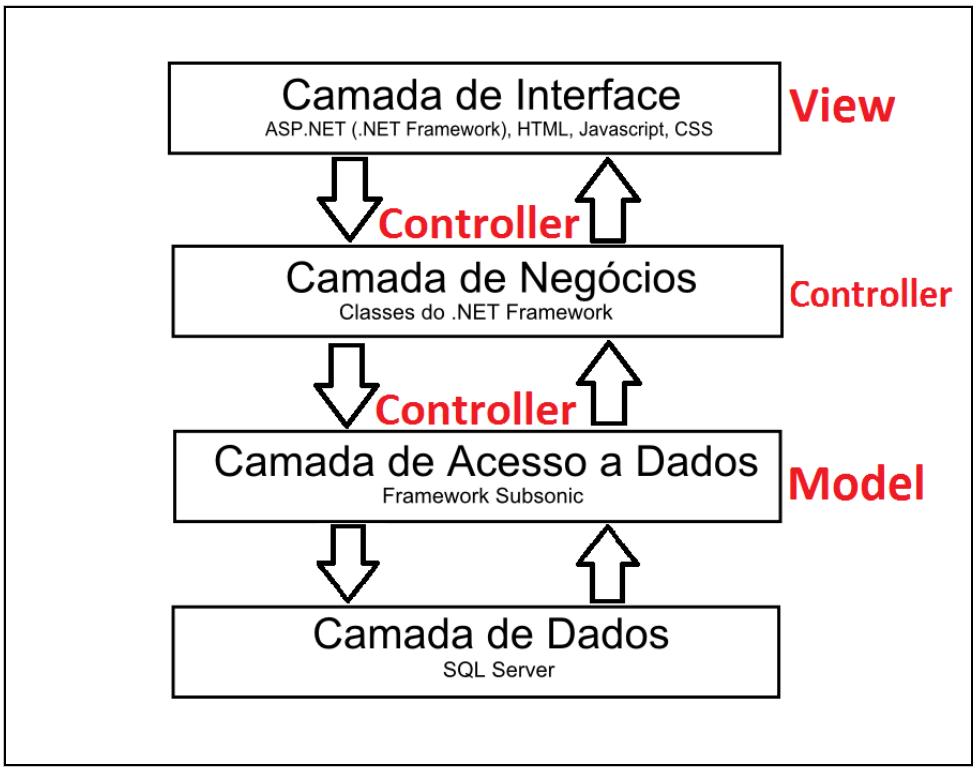
1. Crie um projeto no Visual Studio 2008.
2. Adicione uma *string* de conexão no arquivo Web.config, dê-lhe um nome e aponte-a para um banco de dados válido. Exemplo:

```
<connectionStrings>
    <add name="bbdnet" connectionString="Data
    Source=SSBAAL;Initial Catalog=bbdnet;User
    ID=sa;Password=123"
    providerName="System.Data.SqlClient" />
</connectionStrings>
```

3. Procure a pasta contendo os templates T4 (arquivos com a extensão “tt”) e abra o arquivo “_Settings.tt” e ajuste o valor de “ConnectionStringName” com o nome da *string* de conexão criada anteriormente.
4. Arraste a pasta contendo os templates T4 para o projeto criado e o Visual Studio 2008 irá automaticamente executar os *templates* T4 e criar as classes necessárias.

Apresentadas as tecnologias, na Figura 17 é apresentada a arquitetura do serviço proposto.

Figura 17 – Arquitetura do Serviço



Com todas as tecnologias reunidas e devidamente compreendidas, o próximo capítulo relata e demonstra o desenvolvimento do serviço web proposto.

CAPÍTULO 4 – ESTUDO DE CASO

Neste capítulo, todos os conceitos anteriormente apresentados e discutidos serão aplicados no desenvolvimento de um software como serviço web denominado Base Bibliográfica Digital constituindo assim o estudo de caso deste trabalho.

O Software como Serviço servirá como um portal colaborativo onde qualquer pessoa poderá colaborar adicionando bibliografias através da criação de sua própria base bibliográfica ou colaborando em bases bibliográficas de outras pessoas. Desta forma o serviço atenderá os requisitos que estão relacionados no Escopo do Projeto que é encontrado no apêndice A.

Acompanhando a metodologia de desenvolvimento OpenUP, este projeto foi dividido em fases, sendo elas: Iniciação, Elaboração, Construção e Transição.

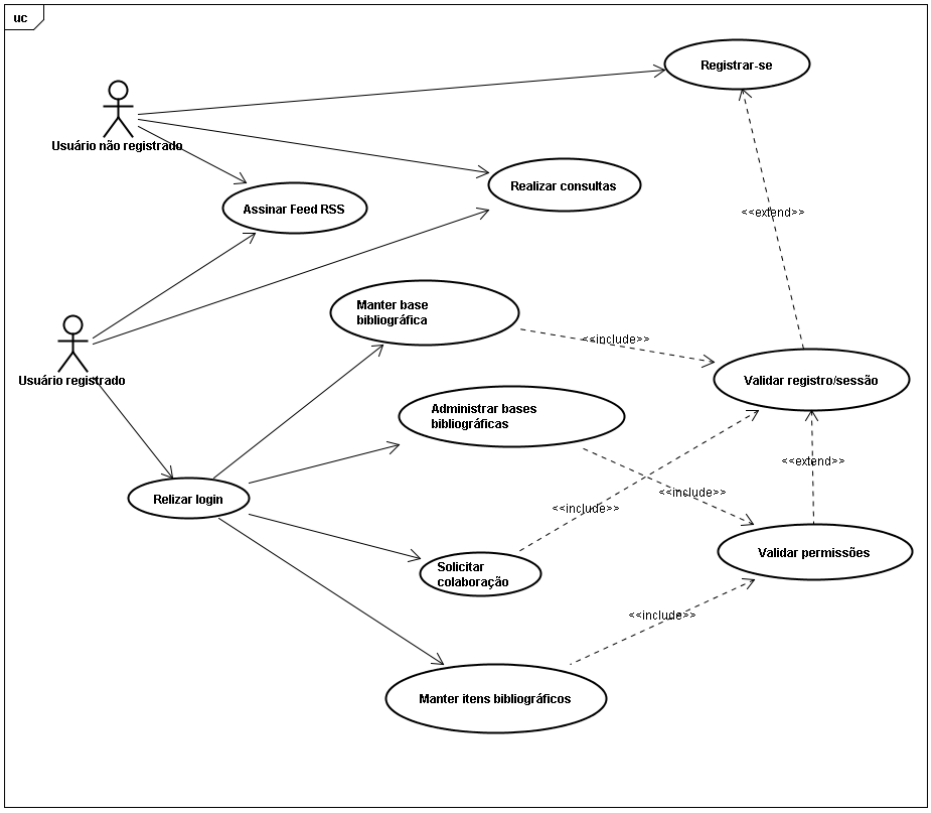
4.1 – Fase 1: Iniciação

Na primeira fase do projeto, foram feitos levantamentos das necessidades juntamente com os stakeholders e foram gerados quatro artefatos: diagrama de casos de uso, descrição dos casos de uso, escopo do projeto e por fim, plano de projeto. A funcionalidade de cada um deles é dada a seguir:

- Escopo do projeto: O documento de escopo do projeto é o primeiro documento a ser criado no projeto, este contém as informações gerais em relação às necessidades dos stakeholders bem como premissas e restrições do mesmo. O documento de escopo do projeto do serviço proposto é encontrado no apêndice A.
- Plano de projeto: O plano de projeto tem como finalidade fornecer uma visão de como será executado o projeto bem como sua estrutura. Alguns exemplos de informações contidas no plano de projeto são a lista de iterações do projeto, a definição dos papéis de cada membro da equipe, os objetivos do projeto, entre outros. O plano de projeto do serviço proposto é encontrado no apêndice B.
- Diagrama de casos de uso: Diagramas de casos de uso são uma representação de como funcionará o produto a ser desenvolvido. Envolve atores, relacionamentos, casos de uso, entre outros. São comumente utilizados para

demonstrar aos interessados no projeto as funcionalidades solicitadas por eles no contexto do projeto. A Figura 18 mostra o diagrama de casos de uso para o projeto proposto.

Figura 18 – Diagrama de casos de uso do projeto



- Descrição dos casos de uso: O documento de descrição dos casos de uso do projeto é uma narrativa e também um detalhamento em texto sobre os casos de uso do mesmo. O documento de descrição dos casos de uso é encontrado no apêndice C.

Com a estruturação destes documentos, o projeto foi mensurado e sua viabilidade aprovada. Pode-se então dar continuidade ao mesmo.

4.2 – Fase 2: Elaboração

A segunda fase do projeto, Elaboração, é a fase que fatores de risco são identificados e eliminados. Além disso, nesta fase é elaborada toda a arquitetura que servirá de base para a implementação do serviço proposto.

4.2.1 – Padrão de Metadados Dublin Core

Nesta etapa do projeto há uma necessidade de extrema importância para o contexto do projeto que é a definição de um padrão de metadados para a arquitetura do projeto.

O conceito de metadados é definido como “dados sobre dados”. É uma informação estruturada, usada para descrever recursos/objetos de informação.

Weibel e Lagoze (1997) disseram:

> A associação de metadados descritivos padronizados com objetos em rede tem o potencial de melhorar substancialmente a descoberta de capacidade de recursos por habilitar a busca baseada em campos (por exemplo, autor, título), que permite a indexação de objetos não-textuais, e permite o acesso ao conteúdo substituto que é distinto do acesso ao conteúdo do recurso em si.

Os metadados são importantes principalmente em questões de padronização de software para a interoperabilidade. Seguindo um padrão de metadados, um *software* poderá se comunicar com outro sem maiores problemas, uma vez que ambos utilizam o mesmo padrão.

O uso de metadados para este projeto tem uma importância como já citado anteriormente pela padronização dos elementos, ou seja, com a padronização de campos de acordo com o padrão Dublin Core, o sistema ficará mais suscetível à interoperabilidade com outros sistemas sendo possível o intercâmbio de informações sem a necessidade de grandes conversões de dados.

O padrão de metadados utilizado para este projeto é o *Dublin Core*, proposto pela *Dublin Core Metadata Initiative* (DCMI).

Este padrão foi proposto pensando na simplicidade e capacidade de auto-explicação do mesmo fazendo com que seja fácil descrevê-lo e publicá-lo por um autor de um documento que utilize este padrão.

O padrão DCMI utiliza apenas 15 elementos, conhecido como *Dublin Core Metadata Element Set* (DCMES) onde ao publicar um documento, o autor deverá preencher estes elementos e submeter seu documento. Os elementos são:

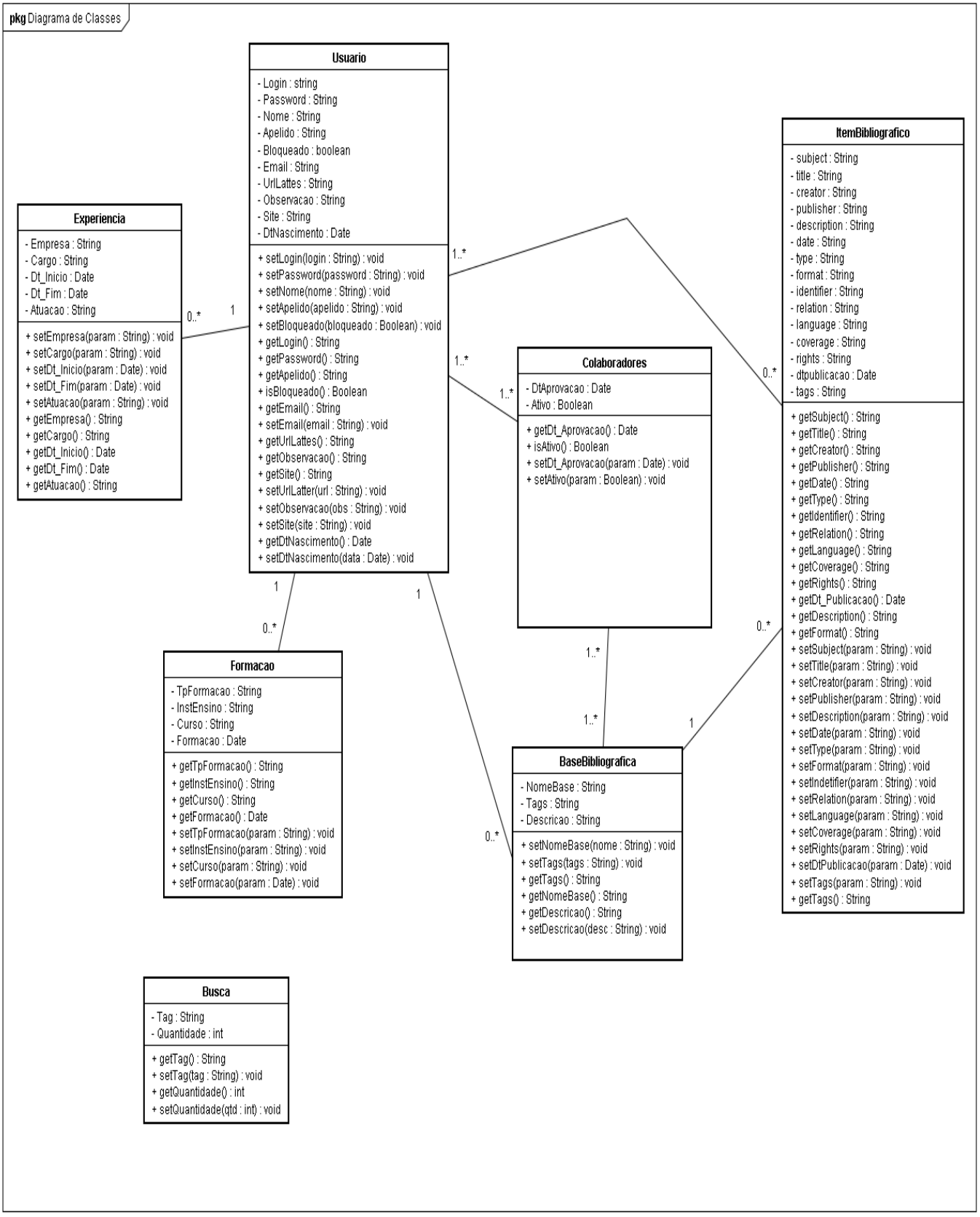
- *Subject* (assunto)
- *Title* (titulo)
- *Creator* (autor do documento)
- *Publisher* (publicador)
- *Contributor* (editores, tradutores, etc)
- *Description* (descrição, resumo)
- *Date* (data de publicação)
- *Type* (tipo de texto: poesia, dicionário, etc)
- *Format* (formato do arquivo submetido)
- *Identifier* (identificação única para o documento)
- *Relation* (fonte relacionada)
- *Source* (fonte do documento)
- *Language* (idioma)
- *Coverage* (época, lugar, cobertura espacial ou temporal do documento)
- *Rights* (direitos autorais)

4.2.2 - Elaboração

Diagrama de Classes, Modelo Lógico do Banco de Dados, Diagramas de Atividade e protótipo foram os artefatos gerados nesta fase de Elaboração. As funcionalidades de cada item são:

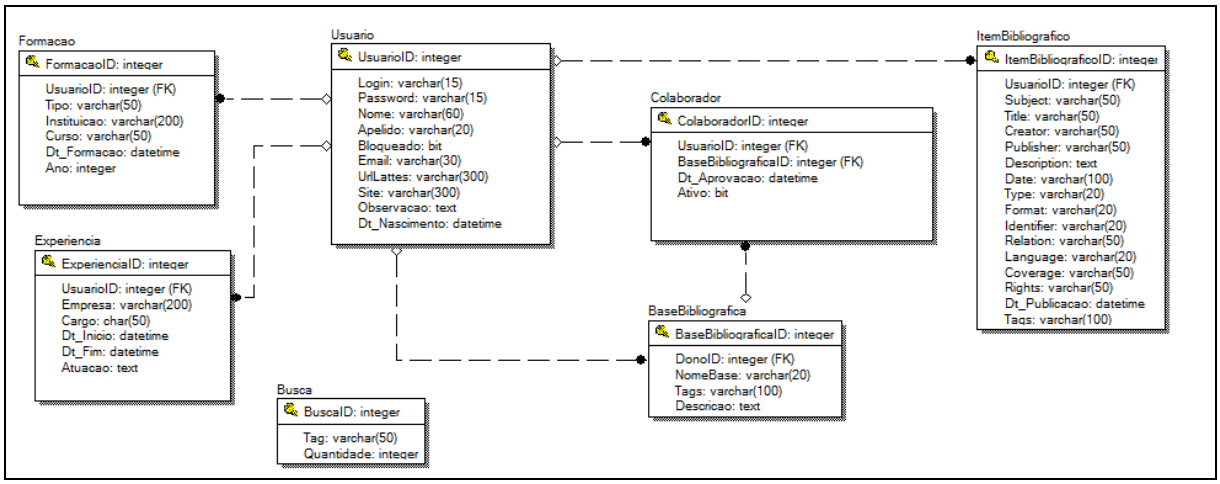
- Diagrama de Classes: O Diagrama de Classes no contexto de um projeto tem como objetivo ser uma fonte de orientação em relação às classes de um projeto desenvolvido sob os conceitos e padrões da orientação a objetos. Em um projeto de software, nem sempre todos os envolvidos com a arquitetura, desenvolvimento ou testes detém amplo conhecimento sobre a estrutura de classes do software ou ainda pode haver dúvidas sobre a mesma e são nesses casos que a importância do diagrama de classes é notada. Na figura 19 está o Diagrama de Classes do projeto proposto.

Figura 19 – Diagrama de Classes do Projeto



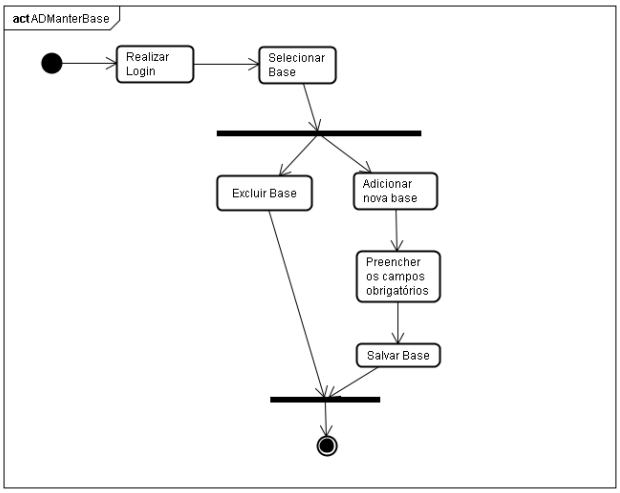
- Modelo Lógico do Banco de Dados: O Modelo Lógico de Banco de Dados é uma representação lógica da estrutura física do banco de dados, apresentando seus relacionamentos, estruturas e regras de integridade. A Figura 20 mostra o Modelo Lógico do Banco de Dados do projeto proposto.

Figura 20 – Modelo Lógico do Banco de Dados do Projeto



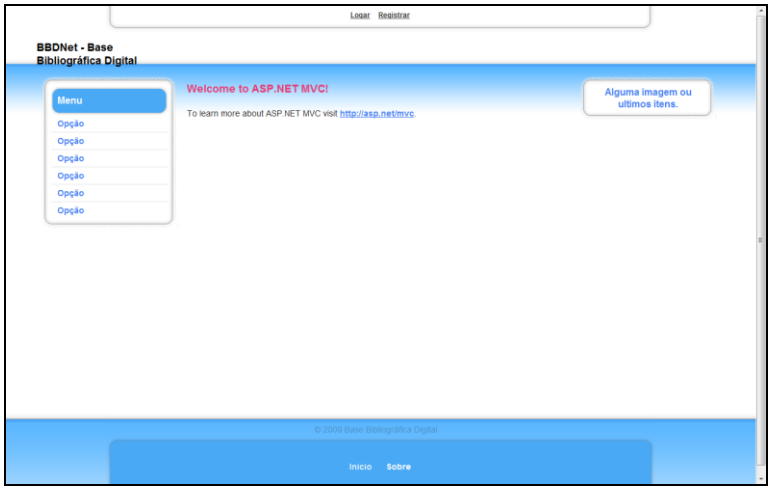
- Diagramas de Atividade: São diagramas que representam o fluxo de atividades entre operações e/ou processos de software e/ou negócios de um software (AMBLER, 2005). A Figura 21 exibe um diagrama de atividades deste projeto.

Figura 21 – Fluxo de Atividades do Projeto



- Protótipo: Um protótipo pode ser feito como um esboço em um papel, uma implementação de uma interface gráfica do software ou qualquer documento que demonstre como será construída a interface gráfica do software. Na Figura 22 é exibido o primeiro protótipo do projeto utilizando-se um *template* do ASP.NET MVC.

Figura 22 – Protótipo do Projeto



Desta forma, a arquitetura do serviço estava toda projetada. Não foi considerado nenhum fator de risco para o projeto além da necessidade de projeção arquitetural. Uma vez definida a arquitetura do serviço, é dada a continuação do projeto.

4.3 – Fase 3: Construção

Na terceira fase do projeto, Construção, todos os requisitos levantados anteriormente juntamente com toda a arquitetura projetada servirão como embasamento para a implementação do serviço.

A fase de construção começou com a instalação das ferramentas e *frameworks* necessários para o desenvolvimento do serviço:

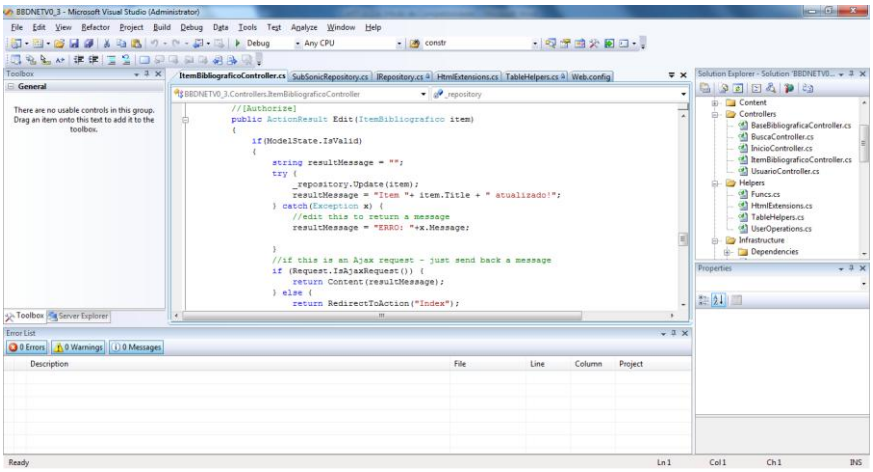
- Microsoft Visual Studio 2008: É a IDE (*Integrated Development Eviroment*) utilizada para a codificação do serviço. Disponível em: http://msdn.microsoft.com/en-us/vstudio/default.aspx
- Microsoft SQL Server 2005: É o SGBD (Sistema Gerenciador de Banco de Dados) utilizado para o armazenamento de dados do serviço. Disponível em:

http://www.microsoft.com/downloads/details.aspx?familyid=220549b5-0b07-4448-8848-dcc397514b41&displaylang=en

- ASP.NET MVC: *Framework* responsável por integrar a arquitetura MVC com o *framework* .NET. Disponível em: http://www.asp.net/mvc
- Subsonic: *Framework* de persistência de dados. Disponível em: http://subsonicproject.com
- MVC Starter Template: *Template* que faz toda a ligação entre o *framework* Subsonic e o *framework* ASP.NET MVC. Disponível em: http://subsonicproject.com/docs/MVC_Starter_Template

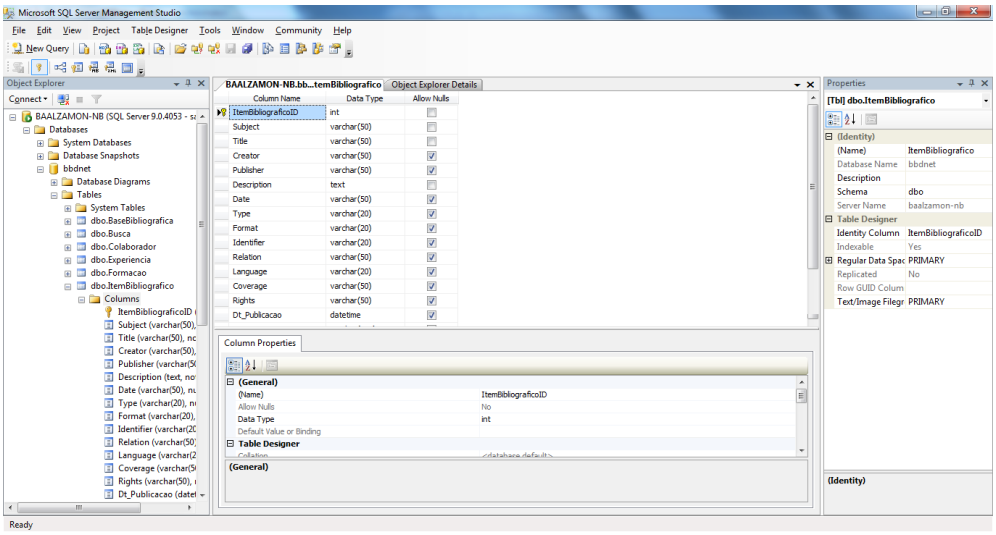
Uma vez que todas as ferramentas e *frameworks* foram instalados foi iniciada então a implementação do serviço propriamente dito. A Figura 23 mostra o ambiente de desenvolvimento do Visual Studio 2008.

Figura 23 – Ambiente de Desenvolvimento do Visual Studio 2008



Além do ambiente de desenvolvimento do Visual Studio 2008, também há o ambiente de gerenciamento do SQL Server 2005 utilizado para a construção do banco de dados do serviço. O script de criação do banco de dados do serviço é encontrado no apêndice D. A Figura 24 mostra o ambiente de gerenciamento do SQL Server 2005.

Figura 24 – Ambiente de Gerenciamento do SQL Server 2005



Após o desenvolvimento do serviço, foram feitos os testes do desenvolvedor (também conhecidos como testes alfa) onde o próprio desenvolvedor testou as funcionalidades do serviço e fez os reparos necessários ao serviço. A Figura 25 exibe uma funcionalidade do sistema em funcionamento.

Figura 25 – Solicitação de Colaboração na Base Bibliográfica

Nesta etapa, o serviço já passou pelos testes do desenvolvedor e foram feitas as correções e ajustes necessários. O projeto então terá sua continuidade para a próxima fase.

4.4 – Fase 4: Transição

A última fase do projeto é a fase de Transição. Nesta fase são executados os testes pela equipe de testes (também conhecidos como testes beta). Os testes são feitos através de *scripts* de teste, um exemplo de um *script* de teste executado nesta fase é encontrado no apêndice E. Caso seja encontrada alguma desconformidade e/ou erro, o mesmo será encaminhado para a equipe de desenvolvimento para correção e então novos testes de desenvolvedor serão feitos seguidos novamente pelos testes realizado pelos Testadores.

Por se tratar de uma aplicação puramente baseada nos pilares da Web 2.0, não há a implantação da mesma é uma empresa ou instituição especifica, pois esta aplicação é um software como serviço e poderá ser utilizada por todos os interessados. Seu funcionamento será o chamado “Beta Eterno”, ou seja, estará sempre em mudanças, sejam elas solicitadas e/ou sugeridas por alguém ou necessárias para o funcionamento do serviço.

Uma vez que os interessados no projeto aprovem o que foi desenvolvido no projeto e/ou suas possíveis correções, o serviço é então disponibilizado em ambiente de produção, ou seja, é disponibilizado através da internet para o resto do mundo de forma que qualquer um pode utilizá-lo e também colaborar.

CONCLUSÃO

Baseando-se nas atuais aplicações web cada vez mais sofisticadas, a web caminha, a largos passos, para um rumo onde se tornará a maior plataforma de execução de aplicativos. *Cloud Computing* (Computação nas Nuvens), *Web Storage* (Armazenamento na Web) e outros termos tendem a, em muito breve, serem muito mais comuns do que atualmente já são. A Web 2.0 está sendo responsável por essa grande e totalmente perceptível mudança e evolução de modelo de aplicação e ela própria, Web 2.0, já está sob discussão de evolução propondo-se sua versão 3.0 visando uma melhor utilização dos conteúdos disponível na web e de forma mais inteligente.

O software como serviço web desenvolvido neste trabalho é um grande exemplo da Web 2.0 uma vez que conceitos importantes da Web 2.0 como "Software Como Serviço", "Inteligência Coletiva", "Tagsonomia" e "Colaboratividade" são características predominantes do serviço desenvolvido além do conceito "Beta Eterno".

Emabasando-se no conceito de "Beta Eterno" das aplicações Web 2.0, é possível concluir que as metodologias ágeis de desenvolvimento são a melhor opção para este tipo de aplicação devido ao seu constante estado de mudanças que é uma das preocupações das metodologias ágeis. O OpenUP, graças à possibilidade de ser estendido, é capaz de moldar-se de acordo com as necessidades apresentadas por cada nova aplicação web proposta. O modelo de desenvolvimento iterativo e incremental do OpenUP possibilitou a divisão do desenvolvimento em etapas fazendo com que as preocupações de desenvolvimento fossem totalmente divididas de acordo com as iterações.

Em termos de agilidade de desenvolvimento em nível de codificação, a produtividade da utilização do *framework* ASP.NET MVC em conjunto com o *framework* Subsonic é grande devido à integração que o MVC Starter Template realiza fazendo com que a integração entre Subsonic e ASP.NET MVC gere benefícios (maiores detalhes na geração de interfaces, forte tipagem de dados nas interfaces, *templates* T4 pré-configurados para um melhor aproveitamento das classes do Subsonic, entre outros) para os desenvolvedores. A geração de interfaces torna-se simples uma vez que o próprio *framework* ASP.NET MVC se encarrega de gerá-las baseando-se nas classes geradas automaticamente pelo *framework* Subsonic baseado nas tabelas e visões do banco de dados.

REFERÊNCIAS

AGILE MANIFESTO. O manifesto ágil. Disponível em: <http://agilemanifesto.org/>. Acessado em: 18 out 2009.

AMBLER, Scott W. **The Elements of the UML 2.0 Style**. Cambridge University Press. Maio, 2005.

COCKBURN, Alistair. **Agile Software Development**. Adisson-Wesley, 2001.

DUBLIN CORE METADATA INITIATIVE. Apresenta informações sobre o padrão Dublin Core de Metadados. Disponível em <http://dublincore.org/>. Acesso em 25 de maio de 2009.

ECLIPSE PROCESS FRAMEWORK. **OpenUP**. 2008. Disponível em: <http://epf.eclipse.org/wikis/openuppt/>. Acessado em: 10 out 2009.

FAGUNDES, Priscila Bastos. **Framework para Comparação e Análise de Métodos Ágeis**. Dissertação de Mestrado. Universidade Federal de Santa Catarina. Florianópolis: 2005.

FOWLER, Martin. **The New Methodology**. 2005. Disponível em: <http://www.martinfowler.com/articles/newMethodology.html>. Acessado em: 20 out 2009.

FAYAD Mohamed E.; SCHMIDT, Douglas C.. **Object-Oriented Application Framework**. 1997.

GAMMA E.; HEML R.; JOHNSON R; VLISSIDES J.. **Padrões de Projeto**. Bookman, 2000.

GOVONI, Darren. Java **Application Frameworks**. 1999.

HIGHSMITH, Jim. **Agile Software Development Ecosystems**. Adisson-Wesley, 2002.

HILMAN. **Metodologias Ágeis**. 2004. Disponível em: <http://www.redes.unb.br/material/ESOO/Metodologias%20%c1geis.pdf>. Acessado em: 20 out 2009.

JUNIOR, Nelson Alves da Nóbrega. **Padrões/Estilos Arquiteturais**. Brasília: IESB (Apostila utilizada para aula de Arquitetura de Software), 2009. Disponível em: <http://nelson.junior.br.googlepages.com/aula05-padroes-estilosarquiteturais.pdf >. Acessado em: 20 out 2009.

KAPLAN, Jeffrey M. **SaaS: Friend or Foe?**. Business Communication Review. p.48-53. jun. 2007. Disponível em: < http://www.wyvilsystems.com/newsap/pdf/saas-friend-or-foe.pdf>. Acesso em 25 out 2009.

LÉVY, Pierre. **A Inteligência Coletiva por uma antropologia no ciberespaço**. 5ª Edição. Edições Loyola. São Paulo. 2005.

LOTAR, Alfredo. **Como Programar com ASP.NET e C#**. Novatec. São Paulo. 2007.

MICROSOFT CORPORATION, MSDN. **Moving Java Applications to .NET**. Disponível em: <http://msdn.microsoft.com/en-us/library/ms973842.aspx>. Acesso em 26 out 2009.

NETO, Oscar Nogueira de Souza. **Análise Comparativa das Metodologias de Desenvolvimento de Softwares Tradicionais e Ágeis**. Trabalho de Conclusão de Curso.Universidade da Amazônia. Belém: 2004.

O'REILLY, Tim. **What Is Web 2.0**. 2005. Disponível em <http://oreilly.com/web2/archive/what-is-web-20.html>. Acessado em: 25/10/2009.

PAULSON, Linda Dailey. **Building Rich Web Applications with Ajax**. Computer. p.14-17. out. 2005. Disponível em: <http://xml.csie.ntnu.edu.tw/JSPWiki/attach/Vicky/Building%20Rich%20Web%20Applications%20with%20Ajax.pdf >. Acessado em: 25 out 2009.

PRESSMAN, Roger. Software **Engineering – A Pratitictioner's Approach**. McGraw-Hill, 5th Edition, 2001.

SANDERSON, Steve. **Pro Asp.Net Mvc Framework**. 1st Edition, 2009.

SOMMERVILLE, Ian. **Software Engineering**. 7th Edition, 2004.

WEIBEL Stuart, LAGOZE, Carl. **An Element Set to Support Resource Discovery - The State of the Dublin Core: January 1997**. International. Journal on Digital Libraries, 1997.

APÊNDICE A – Escopo do Projeto

Escopo do Projeto Catálogo Digital

1-Justificativa

As informações de pesquisa e produção técnica e cientifica, geradas no mundo atual têm aumentado exponencialmente. São novos artigos todos os dias, novos livros, novos trabalhos de conclusão de graduação, mestrado, doutorado, etc.

Muitas vezes um artigo de extrema importância para um pesquisador não chega ao conhecimento do mesmo devido a empecilhos geográficos como, por exemplo, um artigo catalogado em São Paulo será dificilmente visualizado no Rio Grande de Norte a menos que alguém o apresente naquela região.

A necessidade de uma base central de referências é algo perceptível em casos como o exemplo citado, pois facilitaria e muito a proliferação de informação entre pesquisadores e entusiastas de diversos temas e linhas de pesquisa.

2-Objetivos Gerais

Desenvolver um serviço web que funcionará como um portal web colaborativo com o intuíto de uma base centralizada de bibliografias e acervo de artigos, teses, e trabalhos onde usuários poderão fazer consultas, se cadastrar e colaborar alimentando a base de dados.

3-Restrições do Projeto

-Os metadados dos itens bibliográficos obedecerão os requisitos dos pesquisadores de catalogação consultados.

-A modelagem do banco de dados relativa aos itens bibliográficos será feita seguindo o padrão de metadados Dublin Core.

-A equipe do projeto será composta por um stakeholder e um desenvolvedor. O primeiro será responsável pela descrição do problema e suas necessidades para o segundo (desenvolvedor) que será responsável pela modelagem de dados e passos necessários para o desenvolvimento do serviço que atenda às necessidades descritas pelo stakeholder.

4-Premissas do Projeto

-Disponibilidade de um datacenter para a hospedagem de serviço com alta capacidade de armazenamento e disponibilidade.

-Disponibilidade dos pesquisadores da área de catalogação para fornecimento de informações relevantes para o desenvolvimento do projeto.

5-Produtos a serem entregues

-Cadastro de colaboradores.

-Cadastro de bases bibliográficas.

-Cadastro de itens bibliográficos com opção de envio de arquivo.

-Cadastro de recebimento de feed RSS.

-Recuperação dos itens bibliográficos.

-Módulo de administração de bases bibliográficas.

-Controle de acesso.

6-Não-Escopo

-Configuração do ambiente(servidor) do serviço (Banco de dados, servidor web, etc.).

-Compra de domínio da aplicação.

-Divulgação e publicação do portal.

7-Regras de alteração de escopo

Qualquer alteração feita neste escopo aqui apresentado deverá ser devidamente documentado mantendo-se sempre um arquivo de solicitação e o novo escopo deverá ser gerado em um novo arquivo com a finalidade de versões distintas deste escopo em caso de alterações.

As alterações deverão ser discutidas entre os envolvidos no projeto antes de qualquer alteração de fato.

APÊNDICE B – Plano de Projeto

BBDNet

Plano de Projeto

Introdução

Este plano de projeto tem como objetivo gerar uma visão macro do desenvolvimento do projeto BBDNet.

Organização do Projeto

O projeto será composto por duas pessoas com seus papéis definidos da seguinte forma:

Membro da Equipe	Papéis
Antonio Dourado	Analista, Gerente de projeto, Arquiteto, Testador, Desenvolvedor.
Elvis Fusco	Stakeholder.

O membro Antonio Dourado estará em constante contato com o stakeholder Elvis Fusco com a finalidade de alinhar interesses e aprovações para o andamento do projeto.

Processo de Desenvolvimento e Métricas

O projeto será desenvolvido em cima do OpenUP seguindo os passos significantes do mesmo e gerando os artefatos indicados que sejam essenciais para o projeto.

O processo será divido em iterações sendo que no final de cada iteração, será gerado algo (marcos de projeto) relevante para a continuidade do projeto, seja um artefato, protótipo, código, etc.

Marcos de Projeto e Objetivos

Os objetivos do projeto são:

- Definir as necessidades do stakeholder e gerar documentos e casos de uso.
- Definir a arquitetura do serviço e realizar a prototipagem.
- Desenvolver o serviço embasado na arquitetura proposta e realizar testes .
- Colocar o serviço em produção, fazer ajustes e manutenções necessárias.

Fase	Iteração	Objetivos	Data Inicio	Data Fim
Concepção	I1	➢ Levantamento de Requisitos. ➢ Gerar Escopo do Projeto ➢ Gerar Casos de Uso do Projeto ➢ Gerar descrição dos casos	01/07/2009	01/08/2009

		de uso do projeto		
Concepção	I2	➢ Gerar plano de projeto	01/09/2009	03/09/2009
Elaboração	I3	➢ Gerar diagrama de classes ➢ Gerar diagrama de atividades ➢ Gerar modelo lógico do banco de dados. ➢ Gerar protótipo	04/09/2009	10/09/2009
Construção	I4	➢ Preparar o ambiente de desenvolimento ➢ Gerar a estrutura do banco de dados ➢ Implementar as classes que serão utilizadas	11/09/2009	16/09/2009
Construção	I5	➢ Implementação do portal principal ➢ Implementação da área de registro ➢ Implementação da área de login ➢ Implementação da área de configuração de usuário	17/09/2009	22/09/2009
Construção	I6	➢ Implementação das bases bibliográficas ➢ Implementação dos itens bibliográficos ➢ Impementação de Feed RSS nas bases bibliográficas	23/09/2009	30/09/2009
Construção	I7	➢ Revisão e testes do serviço ➢ Correções	01/10/2009	10/10/2009
Transição	I8	➢ Demonstração do serviço para o stakeholder ➢ Disponibilização da primeira versão	11/10/2009	30/10/2009
Transição	I9	➢ Feedback	31/10/2009	15/11/2009

Implantação

O serviço será disponibilizado em um servidor web com suporte à plataforma .NET e banco de dados SQL Server 2005. Nos casos de atualizações, um pacote de atualizações será entregue ao responsável pelo servidor web que por sua vez deverá fazer a atualização do serviço.

Solicitações de correções e melhorias serão recebidos através de e-mails.

APÊNDICE C – Descrição dos Casos de Uso

Descrição dos Casos de Uso do Projeto Catálogo Digital

Caso de uso:	Registrar-se
Descrição:	1-O usuário preenche os campos requeridos. 2-O usuário aceita os termos de uso. 3-O registro estará feito
Alternativas:	
Exceções:	-Caso não sejam informados todos os dados solicitados, será emitida uma mensagem. -Caso o usuário não aceite os termos de uso, será emitida uma mensagem.

Caso de uso:	Assinar Feed RSS
Descrição:	1-O usuário seleciona a base que deseja receber o Feed. 2-O usuário adiciona o Feed em seu programa RSS ou navegador que suporte o mesmo. 3-O Feed estará assinado.
Alternativas:	
Exceções:	

Caso de uso:	Realizar consultas
Descrição:	1-O usuário informa no campo de buscas o conteúdo a ser buscado 2-Caso seja encontrado resultados, serão exibidos em uma lista 3-O usuário poderá clicar em um resultado e visualizar o registro.
Alternativas:	-Caso não seja encontrado registro, será exibida uma mensagem.
Exceções:	

Caso de uso:	Realizar login
Descrição:	1-O usuário deverá informar o nome de login e senha
Alternativas:	-O usuário terá a opção “Esqueceu a senha?” que enviará para o e-mail de registro a sua senha.
Exceções:	-Caso não sejam informados um dos dados solicitados, será exibida uma mensagem de erro. -Caso não seja encontrado registro, será

	exibida uma mensagem de erro.

Caso de uso:	Manter base bibliográfica
Descrição:	1-O usuário realiza login no sistema 2-O usuário abre o cadastro de bases bibliográficas 3-O usuário informa os dados requeridos para a criação da base ou seleciona a base desejada para exclusão
Alternativas:	
Exceções:	-Caso não seja informado algum dado requerido, será emitida uma mensagem de erro no momento da criação. -Caso já exista uma base com o mesmo nome, será emitida uma mensagem de erro no momento da criação. -Caso a sessão tenha expirado, será emitida uma mensagem de erro. -Caso haja algum item bibliográfico na base a ser excluída, será emitida uma mensagem de erro.

Caso de uso:	Administrar bases bibliográficas
Descrição:	1-O usuário realiza login no sistema 2-O usuário acessa a base bibliográfica desejada 3-O usuário poderá aceitar novos colaboradores, excluir colaboradores ou inativar colaboradores.
Alternativas:	
Exceções:	-Caso a sessão tenha expirado, será emitida uma mensagem de erro.

Caso de uso:	Manter itens bibliográficos
Descrição:	1-O usuário realiza login no sistema 2-O usuário seleciona a base bibliográfica desejada 3-O usuário abre o cadastro de novo item bibliográfico, edita ou exclui item bibliográfico. 4-O item será salvo ou excluído.
Alternativas:	-O usuário poderá enviar um arquivo de texto/pdf para disponilizar para download no momento da criação e/ou edição do item.
Exceções:	-Caso o usuário não seja um colaborador, uma mensagem de erro será exibida.

	-Caso a sessão tenha expirado, será emitida uma mensagem de erro.

Caso de uso:	Solicitar colaboração
Descrição:	1-O usuário realiza login no sistema 2-O usuário seleciona a base bibliográfica desejada 3-O usuário solicita a colaboração
Alternativas:	
Exceções:	-Caso a sessão tenha expirado, será emitida uma mensagem de erro.

APÊNDICE D – Script de Criação de Banco de Dados

```
USE [bbdnet]
GO
/****** Object:  Table [dbo].[Busca]    Script Date: 11/15/2009 13:34:09
******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
SET ANSI_PADDING ON
GO
CREATE TABLE [dbo].[Busca](
      [BuscaID] [int] IDENTITY(1,1) NOT NULL,
      [Tag] [varchar](50) NOT NULL,
      [Quantidade] [bigint] NOT NULL,
 CONSTRAINT [PK_Busca] PRIMARY KEY CLUSTERED
(
      [BuscaID] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
SET ANSI_PADDING OFF
GO
/****** Object:  Rule [dbo].[CheckConstraint]    Script Date: 11/15/2009
13:34:04 ******/
CREATE RULE [dbo].[CheckConstraint]
      AS @col BETWEEN 0 AND 1
GO
/****** Object:  Default [dbo].[DEFAULT_N]    Script Date: 11/15/2009
13:34:04 ******/
CREATE DEFAULT [dbo].[DEFAULT_N]
      AS 0
GO
/****** Object:  Table [dbo].[Experiencia]    Script Date: 11/15/2009
13:34:15 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
```

```
GO
SET ANSI_PADDING ON
GO
CREATE TABLE [dbo].[Experiencia](
	[ExperienciaID] [int] IDENTITY(1,1) NOT NULL,
	[Empresa] [varchar](200) NOT NULL,
	[Cargo] [char](50) NOT NULL,
	[Dt_Inicio] [datetime] NOT NULL,
	[Dt_Fim] [datetime] NULL,
	[Atuacao] [text] NULL,
	[UsuarioID] [int] NULL,
 CONSTRAINT [XPKExperiencia] PRIMARY KEY NONCLUSTERED
(
	[ExperienciaID] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO
SET ANSI_PADDING OFF
GO
/****** Object:  Table [dbo].[Formacao]    Script Date: 11/15/2009 13:34:18
******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
SET ANSI_PADDING ON
GO
CREATE TABLE [dbo].[Formacao](
	[FormacaoID] [int] IDENTITY(1,1) NOT NULL,
	[Instituicao] [varchar](200) NOT NULL,
	[Curso] [varchar](50) NOT NULL,
	[Tipo] [varchar](50) NOT NULL,
	[Ano] [int] NULL,
	[UsuarioID] [int] NULL,
 CONSTRAINT [XPKFormacao] PRIMARY KEY NONCLUSTERED
(
	[FormacaoID] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
```

```
GO
SET ANSI_PADDING OFF
GO
/****** Object:  Table [dbo].[BaseBibliografica]    Script Date: 11/15/2009
13:34:07 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
SET ANSI_PADDING ON
GO
CREATE TABLE [dbo].[BaseBibliografica](
      [BaseBibliograficaID] [int] IDENTITY(1,1) NOT NULL,
      [NomeBase] [varchar](100) NOT NULL,
      [Tags] [text] NOT NULL,
      [DonoID] [int] NULL,
      [Descricao] [text] NOT NULL,
 CONSTRAINT [XPKBaseBibliografica] PRIMARY KEY NONCLUSTERED
(
      [BaseBibliograficaID] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO
SET ANSI_PADDING OFF
GO
/****** Object:  Table [dbo].[ItemBibliografico]    Script Date: 11/15/2009
13:34:26 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
SET ANSI_PADDING ON
GO
CREATE TABLE [dbo].[ItemBibliografico](
      [ItemBibliograficoID] [int] IDENTITY(1,1) NOT NULL,
      [Subject] [varchar](50) NOT NULL,
      [Title] [varchar](50) NOT NULL,
      [Creator] [varchar](50) NULL,
      [Publisher] [varchar](50) NULL,
      [Description] [text] NOT NULL,
```

```
	[Date] [varchar](50) NULL,
	[Type] [varchar](20) NULL,
	[Format] [varchar](20) NULL,
	[Identifier] [varchar](20) NULL,
	[Relation] [varchar](50) NULL,
	[Language] [varchar](20) NULL,
	[Coverage] [varchar](50) NULL,
	[Rights] [varchar](50) NULL,
	[Dt_Publicacao] [datetime] NULL,
	[Tags] [varchar](100) NOT NULL,
	[UsuarioID] [int] NULL,
	[BaseBibliograficaID] [int] NULL,
 CONSTRAINT [XPKItemBibliografico] PRIMARY KEY NONCLUSTERED
(
	[ItemBibliograficoID] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO
SET ANSI_PADDING OFF
GO
/****** Object:  Table [dbo].[Colaborador]    Script Date: 11/15/2009
13:34:11 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[Colaborador](
	[UsuarioID] [int] NOT NULL,
	[BaseBibliograficaID] [int] NULL,
	[ColaboradorID] [int] IDENTITY(1,1) NOT NULL,
	[Dt_Aprovacao] [datetime] NULL,
	[Ativo] [bit] NOT NULL,
 CONSTRAINT [XPKColaborador] PRIMARY KEY NONCLUSTERED
(
	[ColaboradorID] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
```

```
EXEC sys.sp_bindefault @defname=N'[dbo].[DEFAULT_N]',
@objname=N'[dbo].[Colaborador].[Ativo]' , @futureonly='futureonly'
GO
EXEC sys.sp_bindrule @rulename=N'[dbo].[CheckConstraint]',
@objname=N'[dbo].[Colaborador].[Ativo]' , @futureonly='futureonly'
GO
/****** Object:  Table [dbo].[Usuario]    Script Date: 11/15/2009 13:34:32
******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
SET ANSI_PADDING ON
GO
CREATE TABLE [dbo].[Usuario](
      [UsuarioID] [int] IDENTITY(1,1) NOT NULL,
      [Login] [varchar](15) NOT NULL,
      [Password] [varchar](15) NOT NULL,
      [Nome] [varchar](60) NOT NULL,
      [Apelido] [varchar](20) NOT NULL,
      [Bloqueado] [bit] NOT NULL,
      [Email] [varchar](30) NOT NULL,
      [UrlLattes] [varchar](300) NULL,
      [Site] [varchar](300) NULL,
      [Dt_Nascimento] [datetime] NULL,
      [Observacao] [text] NULL,
 CONSTRAINT [XPKUsuario] PRIMARY KEY NONCLUSTERED
(
      [UsuarioID] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY],
 CONSTRAINT [UNIQUE_LOGIN] UNIQUE NONCLUSTERED
(
      [Login] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO
SET ANSI_PADDING OFF
GO
```

```
EXEC sys.sp_bindefault @defname=N'[dbo].[DEFAULT_N]',
@objname=N'[dbo].[Usuario].[Bloqueado]' , @futureonly='futureonly'
GO
EXEC sys.sp_bindrule @rulename=N'[dbo].[CheckConstraint]',
@objname=N'[dbo].[Usuario].[Bloqueado]' , @futureonly='futureonly'
GO
/****** Object:  View [dbo].[View_ColabUser]    Script Date: 11/15/2009
13:34:32 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE VIEW [dbo].[View_ColabUser]
AS
SELECT     dbo.Colaborador.Ativo, dbo.Usuario.Nome,
dbo.Colaborador.BaseBibliograficaID, dbo.Colaborador.UsuarioID,
dbo.Colaborador.Dt_Aprovacao
FROM         dbo.Colaborador INNER JOIN
                      dbo.Usuario ON dbo.Colaborador.UsuarioID =
dbo.Usuario.UsuarioID
GO
EXEC sys.sp_addextendedproperty @name=N'MS_DiagramPane1',
@value=N'[0E232FF0-B466-11cf-A24F-00AA00A3EFFF, 1.00]
Begin DesignProperties =
   Begin PaneConfigurations =
      Begin PaneConfiguration = 0
         NumPanes = 4
         Configuration = "(H (1[40] 4[20] 2[20] 3) )"
      End
      Begin PaneConfiguration = 1
         NumPanes = 3
         Configuration = "(H (1 [50] 4 [25] 3))"
      End
      Begin PaneConfiguration = 2
         NumPanes = 3
         Configuration = "(H (1 [50] 2 [25] 3))"
      End
      Begin PaneConfiguration = 3
         NumPanes = 3
         Configuration = "(H (4 [30] 2 [40] 3))"
      End
```

```
Begin PaneConfiguration = 4
   NumPanes = 2
   Configuration = "(H (1 [56] 3))"
End
Begin PaneConfiguration = 5
   NumPanes = 2
   Configuration = "(H (2 [66] 3))"
End
Begin PaneConfiguration = 6
   NumPanes = 2
   Configuration = "(H (4 [50] 3))"
End
Begin PaneConfiguration = 7
   NumPanes = 1
   Configuration = "(V (3))"
End
Begin PaneConfiguration = 8
   NumPanes = 3
   Configuration = "(H (1[56] 4[18] 2) )"
End
Begin PaneConfiguration = 9
   NumPanes = 2
   Configuration = "(H (1 [75] 4))"
End
Begin PaneConfiguration = 10
   NumPanes = 2
   Configuration = "(H (1[66] 2) )"
End
Begin PaneConfiguration = 11
   NumPanes = 2
   Configuration = "(H (4 [60] 2))"
End
Begin PaneConfiguration = 12
   NumPanes = 1
   Configuration = "(H (1) )"
End
Begin PaneConfiguration = 13
   NumPanes = 1
   Configuration = "(V (4))"
End
Begin PaneConfiguration = 14
```

```
      NumPanes = 1
      Configuration = "(V (2))"
   End
   ActivePaneConfig = 0
End
Begin DiagramPane =
   Begin Origin =
      Top = 0
      Left = 0
   End
   Begin Tables =
      Begin Table = "Colaborador"
         Begin Extent =
            Top = 6
            Left = 38
            Bottom = 114
            Right = 209
         End
         DisplayFlags = 280
         TopColumn = 0
      End
      Begin Table = "Usuario"
         Begin Extent =
            Top = 16
            Left = 446
            Bottom = 124
            Right = 598
         End
         DisplayFlags = 280
         TopColumn = 0
      End
   End
End
Begin SQLPane =
End
Begin DataPane =
   Begin ParameterDefaults = ""
   End
End
Begin CriteriaPane =
   Begin ColumnWidths = 11
```

```
         Column = 1440
         Alias = 900
         Table = 1170
         Output = 720
         Append = 1400
         NewValue = 1170
         SortType = 1350
         SortOrder = 1410
         GroupBy = 1350
         Filter = 1350
         Or = 1350
         Or = 1350
         Or = 1350
      End
   End
End
' , @level0type=N'SCHEMA',@level0name=N'dbo',
@level1type=N'VIEW',@level1name=N'View_ColabUser'
GO
EXEC sys.sp_addextendedproperty @name=N'MS_DiagramPaneCount', @value=1 ,
@level0type=N'SCHEMA',@level0name=N'dbo',
@level1type=N'VIEW',@level1name=N'View_ColabUser'
GO
/****** Object:  ForeignKey [R_1]    Script Date: 11/15/2009 13:34:07
******/
ALTER TABLE [dbo].[BaseBibliografica]  WITH CHECK ADD  CONSTRAINT [R_1]
FOREIGN KEY([DonoID])
REFERENCES [dbo].[Usuario] ([UsuarioID])
ON UPDATE CASCADE
GO
ALTER TABLE [dbo].[BaseBibliografica] CHECK CONSTRAINT [R_1]
GO
/****** Object:  ForeignKey [FK_Colaborador_BaseBibliografica]    Script
Date: 11/15/2009 13:34:12 ******/
ALTER TABLE [dbo].[Colaborador]  WITH CHECK ADD  CONSTRAINT
[FK_Colaborador_BaseBibliografica] FOREIGN KEY([BaseBibliograficaID])
REFERENCES [dbo].[BaseBibliografica] ([BaseBibliograficaID])
ON DELETE CASCADE
GO
ALTER TABLE [dbo].[Colaborador] CHECK CONSTRAINT
[FK_Colaborador_BaseBibliografica]
```

```
GO
/****** Object:  ForeignKey [R_2]    Script Date: 11/15/2009 13:34:12
******/
ALTER TABLE [dbo].[Colaborador]  WITH CHECK ADD  CONSTRAINT [R_2] FOREIGN
KEY([UsuarioID])
REFERENCES [dbo].[Usuario] ([UsuarioID])
GO
ALTER TABLE [dbo].[Colaborador] CHECK CONSTRAINT [R_2]
GO
/****** Object:  ForeignKey [R_7]    Script Date: 11/15/2009 13:34:15
******/
ALTER TABLE [dbo].[Experiencia]  WITH CHECK ADD  CONSTRAINT [R_7] FOREIGN
KEY([UsuarioID])
REFERENCES [dbo].[Usuario] ([UsuarioID])
ON UPDATE CASCADE
ON DELETE CASCADE
GO
ALTER TABLE [dbo].[Experiencia] CHECK CONSTRAINT [R_7]
GO
/****** Object:  ForeignKey [R_6]    Script Date: 11/15/2009 13:34:18
******/
ALTER TABLE [dbo].[Formacao]  WITH CHECK ADD  CONSTRAINT [R_6] FOREIGN
KEY([UsuarioID])
REFERENCES [dbo].[Usuario] ([UsuarioID])
ON UPDATE CASCADE
ON DELETE CASCADE
GO
ALTER TABLE [dbo].[Formacao] CHECK CONSTRAINT [R_6]
GO
/****** Object:  ForeignKey [FK_ItemBibliografico_BaseBibliografica]
Script Date: 11/15/2009 13:34:27 ******/
ALTER TABLE [dbo].[ItemBibliografico]  WITH CHECK ADD  CONSTRAINT
[FK_ItemBibliografico_BaseBibliografica] FOREIGN KEY([BaseBibliograficaID])
REFERENCES [dbo].[BaseBibliografica] ([BaseBibliograficaID])
ON DELETE CASCADE
GO
ALTER TABLE [dbo].[ItemBibliografico] CHECK CONSTRAINT
[FK_ItemBibliografico_BaseBibliografica]
GO
/****** Object:  ForeignKey [R_5]    Script Date: 11/15/2009 13:34:27
******/
```

```
ALTER TABLE [dbo].[ItemBibliografico]  WITH CHECK ADD  CONSTRAINT [R_5]
FOREIGN KEY([UsuarioID])
REFERENCES [dbo].[Usuario] ([UsuarioID])
GO
ALTER TABLE [dbo].[ItemBibliografico] CHECK CONSTRAINT [R_5]
GO
```

APÊNDICE E – Script de Teste

Nome do Teste	Solicitar Colaboração
Caso de uso testado:	Solicitar Colaboração
Descrição:	Este teste tem a finalidade de testar a solicitação de colaboração em uma base bibliográfica
Pré Condições	Ter um login e senha no serviço
Pós Condições	O dono da base bibliográfica deverá receber um e-mail de solicitação
Notas adicionais:	
Resultado (Passou/Falhou/Alerta/ Incompleto)	**Passou**

	PASSOS DO TESTE	RESULTADOS ESPERADOS	P	F
1.	Realizar login	Entrar no service	X	
2.	Selecionar a base	Acessar a base e listar os itens da mesma	X	
3.	Solicitar a colaboração	Receber uma mensagem de solicitação e o dono da base receber um e-mail de solicitação	X	