UNIVERSIDADE FEDERAL DO ESTADO DO RIO DE JANEIRO
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
ESCOLA DE INFORMÁTICA APLICADA

Redes sociais para desenvolvedores

Diego Felipe Soares Pereira dos Santos
Ricardo de Oliveira dos Santos

Orientador
Márcio de Oliveira Barros

RIO DE JANEIRO, RJ – BRASIL
JULHO DE 2010

Redes sociais para desenvolvedores

Projeto de Graduação apresentado à Escola de Informática Aplicada da Universidade Federal do Estado do Rio de Janeiro (UNIRIO) para obtenção do título de Bacharel em Sistemas de Informação.

Diego Felipe Soares Pereira dos Santos
Ricardo de Oliveira dos Santos

Orientador
Márcio de Oliveira Barros

Redes sociais para desenvolvedores.

Aprovado em ____/_____/_____

BANCA EXAMINADORA

_____

Prof. Márcio de Oliveira Barros, DSc. (UNIRIO)

_____

Prof. Leonardo Guerreiro Azevedo, DSc. (UNIRIO)

_____

Prof. Alexandre Luís Correa, DSc. (UNIRIO)

Os autores deste Projeto autorizam a ESCOLA DE INFORMÁTICA APLICADA da UNIRIO a divulgá-lo, no todo ou em parte, resguardando os direitos autorais conforme legislação vigente.

Rio de Janeiro, ____ de _____ de _____

_____

Diego Felipe Soares Pereira dos Santos

_____

Ricardo de Oliveira dos Santos

Agradecimentos

Ao professor Márcio Barros pela orientação, dedicação e paciência durante todo o processo do projeto.

Aos professores Leonardo Azevedo e Alexandre Correa por terem aceitado o convite para fazer parte da banca, mesmo com um prazo curto para leitura da monografia. Esta atitude reafirmou todo o respeito que temos por ambos.

Às nossas mães que viabilizaram a conclusão deste projeto com o café da manhã após noites em claro.

Aos amigos e familiares que apoiaram esta empreitada e nos deram forças para que esse projeto fosse concluído em tão pouco tempo.

RESUMO

Santos, Diego Felipe S. P. dos; Santos, Ricardo de O. dos; Barros, Márcio de Oliveira. **Redes sociais para desenvolvedores**. Rio de Janeiro, 2010. 58p. Projeto de Graduação – Departamento de Informática Aplicada, Universidade Federal do Estado do Rio de Janeiro.

Muitos problemas encontrados durante o processo de desenvolvimento de software possuem relação direta com a falta de compartilhamento dos conhecimentos entre os desenvolvedores. Este conhecimento, que geralmente é mantido de forma tácita, transforma os desenvolvedores na fonte mais fiel de informações para resolução de dúvidas ligadas ao desenvolvimento do software. Isto torna os desenvolvedores indispensáveis e lhes ocupa um tempo precioso, que poderia ser dedicado aos seus novos projetos. Esta monografia apresenta uma proposta para viabilizar o aumento da comunicação entre os desenvolvedores, fazendo com que o conhecimento mantido por estes seja publicado por vias em que possa ser consultado por outros integrantes da equipe. Conceitos como percepção de projeto (*design awareness*), gestão do conhecimento e redes sociais são explorados em uma solução que tem como objetivo a disseminação do conhecimento contido nas mentes de cada desenvolvedor, de forma simples e intuitiva. A implementação da solução consiste em uma rede social composta por um *website* e um conjunto de *add-ins*.

Palavras-chave: Construção de Software, Redes Sociais, Design Awareness, Gestão do Conhecimento.

ABSTRACT

Santos, Diego Felipe S. P. dos; Santos, Ricardo de O. dos; Barros, Márcio de Oliveira. **Social Networks for developers**. Rio de Janeiro, 2010. 58p. Graduation Project – Department of Applied Computer Science, Universidade Federal do Estado do Rio de Janeiro.

Many problems encountered during the software development process are directly related to the lack of knowledge sharing among developers. This knowledge, which is usually held tacitly by the development team, turns the developers the most accurate source of information for resolving questions related to software development. This makes some developers indispensable, and occupy their valuable time, which could be dedicated to new projects. This monograph presents a proposal to allow increased communication among developers, ensuring that the knowledge held by them is published in such a way such that it can be referred by other team members. Concepts such as design awareness, knowledge management, and social networks are explored in a solution that aims to spread the knowledge contained in the mind of each developer, in a simple and intuitive way. The proposed solution is a social network consisting of a website and a set of add-ins to well-known software development environments.

Keywords: Software Development, Social Networks, Design Awareness, Knowledge Management.

Índice

Índice de Figuras

Índice de Tabelas

1 *Introdução*

1.1 Motivação

Desenvolver software é uma tarefa cara e demorada. Apesar de todos os esforços dos desenvolvedores, os projetos quase sempre atrasam e acumulam defeitos. Por consequência, o que era um projeto simples se torna complexo.

O desenvolvimento de um software de qualidade requer a colaboração de muitas pessoas, principalmente dos desenvolvedores, que precisam compartilhar informações entre si, de forma que fique fácil de entender as dependências e o comportamento de um sistema [KO *et al.,* 2007].

Tendo em vista que este compartilhamento de informações, na maioria das vezes, não ocorre da melhor forma possível e sabendo que grande parte do esforço de desenvolvimento de software é constituído pela codificação, buscou-se criar uma maneira simples de possibilitar a disseminação do conhecimento individual entre os desenvolvedores em seu ambiente de trabalho.

1.2 Objetivos

Neste trabalho é proposta a criação de uma rede social entre os desenvolvedores de um projeto de software, a fim de aumentar o compartilhamento das informações que são necessárias para o cumprimento das atividades do projeto, assim como aumentar a percepção das atividades realizadas e soluções aplicadas pelos desenvolvedores.

Esta rede é composta por um *website*, um conjunto de *add-ins* e w*eb services*. O *website* armazena informações que devem ser compartilhadas entre os desenvolvedores. Estas informações são acessadas por *add-ins* que estendem as funcionalidades do ambiente de desenvolvimento de software utilizado no projeto. Os *add-ins* se comunicam com o *website* através de *web services*.

Espera-se que, pelo uso destas ferramentas, os desenvolvedores obtenham as informações necessárias para apoiar suas tarefas no contexto de um projeto de desenvolvimento de software de forma rápida e objetiva, gerando um aumento na qualidade do produto, bem como a diminuição dos problemas apresentados durante o seu processo de desenvolvimento.

1.3 Organização do texto

O presente trabalho está estruturado em capítulos e será desenvolvido da seguinte forma:

- Capítulo II: *Design Awareness* – apresenta ao leitor os principais temas relacionados ao projeto aqui desenvolvido, definindo-os e explicando de forma sucinta seus pontos principais. Dessa forma, o leitor estará apto a analisar a solução proposta para o problema de compartilhamento de informações entre desenvolvedores de software;

- Capítulo III: ASP.NET MVC – introduz a arquitetura de MVC (*Model-View-Controller*) e, em seguida, se concentra em detalhes técnicos da plataforma ASP.NET MVC, utilizada na construção do *website* desse projeto;

- Capítulo IV: Solução Proposta – explica, detalhadamente, as funcionalidades e a estrutura da solução proposta para o problema de compartilhamento de

informações. O capítulo está dividido em diversas seções, que descrevem cada um dos componentes da solução, explicitando o porquê e como foi feito;

- Capítulo V: Exemplo de Uso – exibe a solução implementada, fazendo uso de imagens da própria aplicação para facilitar o entendimento. Um caso de uso real do sistema é reproduzido detalhadamente;

- Capítulo VI: Conclusão – Reúne as considerações finais, assinala as contribuições da pesquisa e sugere possibilidades de aprofundamento posterior.

2 *Design Awareness*

Nesse capítulo são apresentados os conceitos necessários para o bom entendimento do projeto. Uma visão geral de construção de software é apresentada (Seção 2.1). Identificamos quais os possíveis problemas que podem acontecer durante esse processo (Seção 2.1.1). A causa desses problemas está, na maioria dos casos, relacionada não só à falta de *awareness* (Seção 2.2), mas também a um ineficiente, ou inexistente, processo de gestão do conhecimento (Seção 2.3). As redes sociais (Seção 2.4) são uma solução proposta para tentar amenizar esses problemas.

2.1 Construção de Software

Segundo McConnell (2004), a construção é uma parte extensa do processo de desenvolvimento de software, que, por sua vez, pode ser considerado um processo complexo. Nos últimos 25 anos, pesquisadores da área de desenvolvimento de software identificaram diversas atividades distintas relacionadas a essa prática. Entre elas estão: definição do problema, desenvolvimento dos requisitos, planejamento da construção, arquitetura do software, projeto detalhado, codificação e depuração, testes unitários, testes de integração, integração, testes de sistema e manutenção corretiva.

Segundo Barros (2010), a construção de software trata principalmente da codificação e depuração, ocupando cerca de 60% a 70% do tempo total de desenvolvimento, e sendo o planejamento anterior, composto por pelo menos

“Documento de visão”, “Requisitos” e “Arquitetura”. Detalhando cada um dos pré-requisitos da construção, obtém se:

- Documento de visão: apresenta uma visão geral dos resultados esperados de um projeto. Em geral, é utilizado como base para as primeiras estimativas gerenciais acerca do projeto e como guia para o início da análise de requisitos;
- Requisitos: conjunto organizado de documentos descrevendo as funcionalidades esperadas do projeto. Descrevem como as entradas do processo se transformam em seus resultados e garantem que o usuário, e não o desenvolvedor, determine as funcionalidades presentes no sistema;
- Arquitetura: a arquitetura é o centro de projetos modernos de software. Ela registra as principais decisões sobre como será implementada a solução para o problema identificado durante a análise de requisitos. Ela descreve o sistema a partir de unidades de alto nível, como seus principais módulos, e serve de base para o projeto detalhado e, posteriormente, para a codificação.

O resultado de uma boa arquitetura é a integridade conceitual do sistema em desenvolvimento, que indica que o sistema foi construído de forma uniforme, seguindo padrões. Desta forma, com códigos mais legíveis, simples de entender, corrigir e expandir, os custos de manutenção do sistema tendem a ser reduzidos.

Tais pré-requisitos são importantes porque os documentos gerados a partir destes servem como “plantas base” para a construção, evitando que se prossiga às escuras pelas fases seguintes de desenvolvimento e facilitando a percepção de erros no decorrer do processo. Estudos indicam que corrigir um erro nas etapas iniciais de desenvolvimento é de dez a cem vezes mais barato do que corrigir o mesmo erro durante os testes [BARROS 2010].

Alguns outros fatores demonstram a importância da construção de software. O código é muitas vezes o único documento final, representando assim a única descrição precisa do sistema. É na construção que se encontram as atividades obrigatórias de fato, enquanto outras atividades complementares são muitas vezes negligenciadas, e é também nesta fase onde ocorre a maior variação de produtividade, podendo alguns desenvolvedores ser dez vezes mais rápidos que a média.

2.1.1 Problemas Enfrentados e Hábitos dos Desenvolvedores de Software

De acordo com LaToza *et al*. (2006), grande parte do tempo gasto pelos desenvolvedores é em codificação, sendo metade deste tempo utilizado para correção de *bugs* ao invés da criação de novos módulos. Apesar da disponibilidade de visões de alto nível do código e de editores visuais, tais como as ferramentas de modelagem UML, os desenvolvedores ainda preferem examinar o código em si. Esta seção descreve os principais pontos da pesquisa feita por LaToza *et al.* (2006).

Em relação à comunicação, ferramentas como e-mail e *messengers* são pouco utilizadas, sendo trocadas pelo "boca-a-boca" e ficando com um papel menor para contatos informais, como por exemplo, a marcação do almoço. Os desenvolvedores alegam demora nas respostas dos e-mails, má interpretação do conteúdo e o tempo gasto para redigir o texto, que muitas vezes não possui a quantidade de informação correta (seja para mais ou menos) para que o receptor entenda, além do tédio de redigir. Embora dados mostrem que a eficiência dos encontros não marcados é maior que qualquer outro método, a utilização de métodos informais, sem uso da tecnologia, acarreta na não retenção das informações trocadas.

Descobriu-se que os desenvolvedores produzem poucas documentações sobre o seu código, normalmente não atualizadas com o decorrer do tempo, armazenando ideias importantes e algumas especificações apenas em suas mentes. Desta forma, cada

desenvolvedor torna-se indispensável e cria-se uma clara divisão entre o código que determinado indivíduo controla e o código do resto da sua equipe. Mais forte ainda do que esta divisão é a separação do código entre as equipes, que tentam se isolar das demais.

Os próprios desenvolvedores informaram, através de entrevistas, os principais problemas enfrentados por eles, sendo os mais comentados:

- Dificuldade para entender a lógica por trás de código existente: foi relatada pela grande maioria dos desenvolvedores como sendo o problema mais grave. O caminho natural é a verificação do código-fonte, porém o que se mostrou o método mais eficiente foi o *debug*. A falta de documentação atualizada e a dificuldade para encontrar até mesmo a desatualizada acabam obrigando que o desenvolvedor busque as informações desejadas com outros desenvolvedores;
- Interrupções durante tarefas: são geradas por encontros não marcados, por exemplo, podendo provocar o esquecimento por parte do interrompido da sua tarefa atual, prejudicando assim seu rendimento. Por este motivo, alguns desenvolvedores simplesmente se isolam para que não sejam incomodados;
- Dificuldade para saber de mudanças que tenham ocorrido em outros locais: é vista como algo prejudicial e esta pode ser a causa de muitos problemas. O desconhecimento do todo faz com que duplicatas sejam geradas, conflitos em arquivos compartilhados sejam gerados, entre outros;
- Duplicação de código: foi apontada, por desenvolvedores, como um grande problema. Porém, ao contrário do senso comum, não ocorre apenas pela prática do "copiar e colar", tendo outros motivos como causa. Alguns exemplos são: desenvolvedores implementando, separadamente, funcionalidades iguais; manutenção de diversos ramos de um projeto (versão em produção e versão utilizada

pelos clientes, por exemplo); re-implementação de código, pelo mesmo desenvolvedor, em linguagens diferentes. A manutenção de várias cópias é difícil, pois após um tempo torna-se complicado a mudança consistente em todas as cópias existentes, devido à perda de histórico das duplicatas.

Apesar da aversão dos desenvolvedores a certos processos, como a confecção e manutenção de documentação, eles se mostraram abertos a mudanças, testando constantemente novas ferramentas e práticas de trabalho para otimizar seu trabalho. Muitas equipes de desenvolvimento estão experimentando "práticas ágeis" [STOBER e HANSMANN 2009], que consistem em uma coleção de comportamentos destinados a fazer o desenvolvimento de software mais eficiente.

2.2 Awareness

Conforme as pessoas se engajam em atividades comuns ao dia a dia, elas mantêm a percepção (*awareness*) dos outros ao seu redor, ou seja, um entendimento sobre o que eles fazem, onde estão ou o que dizem. Essa compreensão pode ajudar as pessoas a fazerem inferências sobre as intenções, ações ou mesmo emoções de terceiros, além de fornecer um contexto para o compartilhamento de atividades e interações sociais. *Awareness* não se limita somente às pessoas próximas, mas sim a todas aquelas de quem é possível se obter algum conhecimento [PANOS *et al.* 2009].

A definição acima apresenta um foco na percepção de pessoas, porém essa percepção pode ser facilmente estendida ao que é chamado de consciência da situação (*situation awareness*). *Situation awareness* pode ser entendido como "saber o que está acontecendo", ou mais precisamente "a percepção dos elementos no ambiente dentro de um volume de tempo e espaço, a compreensão do seu significado e a projeção de seu estado no futuro próximo" [ENDSLEY e GARLAND 2000].

Os sistemas de percepção podem ser definidos como sistemas destinados a ajudar as pessoas a construir e manter a consciência das atividades uns dos outros, mesmo quando os participantes não estão no mesmo ambiente. Para o ambiente de desenvolvimento de software, esses sistemas deveriam combinar tanto a representação da situação, como por exemplo, mudanças de código, quanto a informação a respeito das pessoas (desenvolvedores).

2.3 Gestão do Conhecimento

A partir de uma perspectiva prática de negócios, Bergeron (2003) define a Gestão do Conhecimento como sendo uma estratégia de otimização de negócios que identifica, seleciona, organiza, destila e empacota informações essenciais para o negócio da empresa, de um modo que aumenta o desempenho do empregado e a competitividade da corporação.

Polanyi (1967) categorizou o conhecimento em tácito e explícito. O último é basicamente o que pode ser facilmente documentado e distribuído, enquanto o conhecimento tácito reside na mente humana, comportamento e percepção e, assim, é difícil de ser formalizado e distribuído. Como esperado, as abordagens tradicionais de gerenciamento de conhecimento usualmente concentram-se no conhecimento explícito, mas alguns autores indicam que é necessário capturar o processo e transferir conhecimento tácito a fim de entender completamente um processo da organização.

Existem diversas definições de estágios, ou fases, compondo o processo de gerenciamento de conhecimento. Segundo Bose (2004) *apud* Goldoni e Oliveira (2006), o processo cíclico de gestão do conhecimento é composto por:

- Criação do conhecimento: o conhecimento é criado no momento em que as pessoas descobrem novas maneiras de fazer as coisas. O conhecimento pode ser criado

pelos funcionários da organização ou pode ser transferido de laboratórios de pesquisa para a organização;

- Captura do conhecimento: após ser construído, o conhecimento criado deve ser armazenado na sua forma primitiva;
- Refinamento do conhecimento: neste momento, o conhecimento tácito é contextualizado e refinado, juntamente com o conhecimento explícito;
- Armazenamento do conhecimento: a codificação do conhecimento tácito e explícito ajuda no entendimento do conhecimento para uso posterior;
- Gerenciamento do conhecimento: o conhecimento deve se manter atualizado. Desta forma, a organização deve garantir que o conhecimento seja revisado;
- Disseminação do conhecimento: o conhecimento deve estar disponível para todos os funcionários da organização. Ferramentas como groupware e internet / intranet auxiliam nesta etapa.

Independente do seu processo, cada programa de gestão de conhecimento necessita de equilíbrio no tipo de conhecimento que está se concentrando. Baseado neste foco, o programa de gestão de conhecimento pode ser categorizado em um dos quatro estilos definidos por Choi e Lee (2003):

- Passivo: o conhecimento não é gerenciado de uma maneira sistemática – pouco interesse em KM;
- Orientado ao Sistema: coloca ênfase em codificação e reuso do conhecimento, e como consequência, aumenta a habilidade de codificação através da Tecnologia da Informação (TI);
- Orientado à Pessoa: enfatiza na aquisição e compartilhamento do conhecimento tácito e da experiência interpessoal;

- Dinâmico: explora ambos os conhecimentos, tácito e explícito, e faz isso de forma dinâmica.

2.3.1 Gestão do Conhecimento e Redes Sociais

Organizações buscam aumentar a sua competitividade no mercado. Para isto, elas procuram novas maneiras de aumentar sua produtividade, a qualidade de seus produtos e de diminuir os custos. Uma das maneiras de atingir este objetivo é a disseminação de maneira eficiente do conhecimento individual entre os membros da organização.

A preservação e empacotamento do conhecimento corporativo, como por exemplo, a informação no contexto no qual ela é usada, é especialmente relevante hoje em dia, dado que a maior parte da força de trabalho orientada a serviços é composta por trabalhadores do conhecimento. Para competir com sucesso na economia atual, organizações devem tratar o conhecimento que contribui para suas principais competências da mesma forma que fariam com qualquer outro ativo estratégico e insubstituível [BERGERON 2003].

De acordo com Choi e Lee (2003), a Gestão do Conhecimento em uma empresa de software é a oportunidade de se criar uma linguagem padrão entre os desenvolvedores, de tal maneira que eles possam interagir, lidar e distribuir conhecimento e experiências. A redução da perda de Capital Intelectual de desenvolvedores que deixam a empresa, a redução de custo no desenvolvimento de novos produtos, o aumento da produtividade por fazer o conhecimento acessível mais facilmente para todos os empregados são alguns dos benefícios em utilizar uma estratégia de Gestão do Conhecimento.

Neste contexto, as redes sociais têm mostrado sinais de ser uma poderosa ferramenta de disseminação do conhecimento individual. Elas ajudam a compartilhar os

conhecimentos explícitos e até mesmo tácitos de seus participantes, formando assim uma base sólida de conhecimento da organização.

Staab *et al.* (2005) afirma que os ambientes das redes sociais são um bom mecanismo para promover maior interatividade entre indivíduos. A captura de conhecimento tácito através de ferramentas de interação dentro das Redes Sociais Baseadas na Web também permite estender este conhecimento. De acordo com Davenport e Prusak (1998) *apud* Costa *et al.* (2009), para criar novo conhecimento é necessário expor o ser humano a novas informações, para que ele processe estas e gere novo conhecimento dentro da sua mente.

2.4 Redes Sociais

Tanto na vida profissional quanto pessoal os seres humanos tendem a formar grupos baseados em afinidades ou conhecimentos em comum. Atraímo-nos àqueles com os quais dividimos interesses. A maioria de nós pertence a, pelo menos, uma rede social no mundo real. De maneira não surpreendente, essas redes estão migrando rapidamente para o universo online.

Desde a última década, as redes sociais estão presentes em diferentes formas no mundo online. Porém, só vieram a adquirir grande importância nos últimos quatro anos. As redes sociais são criadas pelas mais diversas razões, entre elas: reencontrar amigos, divulgar bandas, compartilhar interesses em comum, divulgar ou classificar conteúdo, etc. Entretanto, apesar de terem suas diferenças, elas geralmente apresentam os seguintes conceitos:

- Perfis – cada membro de uma rede tem um perfil online, que serve como a sua identidade na rede. As informações contidas nesse perfil podem variar de acordo com o propósito da rede. No âmbito profissional, por exemplo, perfis muitas vezes

contêm informações sobre as habilidades, experiência, formação e interesses, além de qualquer outra informação relevante do indivíduo.

- Conexões – redes sociais online normalmente permitem aos indivíduos criarem conexões entre si. Em alguns casos, essas conexões são implícitas, pois derivam de ações do passado (como envio de e-mail para outro membro da rede). Em outros casos, as ligações são explícitas, ou seja, são criadas pelos próprios usuários.

2.4.1 Definições

De acordo com Naimzada *et al.* (2008), uma rede não direcionada (N, g) descreve um sistema de relações recíprocas entre os indivíduos de um conjunto N = (1, 2,..., n), como amizades, fluxos de informação, etc. Os indivíduos são os nós do grafo g e as arestas representam as relações bilaterais entre eles. É comum referir-se ao grafo g como uma rede (omitindo o conjunto de indivíduos).

A notação $i, j \in g$ indica que i e j estão conectados na rede g. Portanto, uma rede g é uma lista de pares de indivíduos que estão ligados uns aos outros. O conjunto de todas as possibilidades de conexão entre indivíduos em N é denotado por $g^N = \{i, j \mid i, j \in N, i \neq j \}$. Desta maneira, $G = \{g \subset g^N \}$ é o conjunto de todas as possíveis redes em N e g^N é denominado como a rede completa.

Um caminho em g entre os indivíduos i e j é uma sequência de indivíduos $i = i_1, i_2,..., i_k = j$, sendo $K \geq 2$, tal que $i_k i_{k+1} \in g$ para cada $k \in \{ 1,..., K - 1\}$. Indivíduos que não estão conectados por um caminho estão em diferentes componentes C de g. Aqueles que estão ligados por um caminho estão no mesmo componente. Portanto, os componentes de uma rede são os subgrafos distintos e conectados de uma rede. O conjunto de todos os componentes da rede pode ser indicado por $C(g)$. Portanto, $g = \bigcup_{g'}$

$\in_{C(g)}$ *g'*. Os indivíduos que tem, pelo menos, uma conexão na rede g podem ser representados como $N(g)$.

2.4.2 Análise Estrutural das Redes Sociais

Análise das redes sociais (do inglês, Social Network Analysis – SNA) é o estudo das relações sociais entre um conjunto de atores. O seu principal objetivo é detectar e interpretar possíveis padrões envolvendo essas relações. A aplicação de SNA foi primeiramente feita por sociólogos [WASSERMAN e FAUST, 1994]. Porém, o interesse no assunto tem crescido entre físicos e administradores [ROSSO, 2009].

As SNA também podem ser aplicadas em várias áreas, tais como gestão do conhecimento [CROSS *et al.*, 2004], estudo da estrutura das publicações científicas, rede elétrica, fenômenos naturais [DOROGOVTSEV e MENDES, 2003], análises econômicas [NAIMZADA *et al.*, 2008] e desenvolvimento de software [ROSSO, 2009]. A seguir, são apresentadas as definições e métricas usadas na SNA, assim como outros conceitos necessários para a sua compreensão e análise. As referências utilizadas são Wasserman (1994) e Rosso (2009).

2.4.2.1 Densidade, Coesão e Distância

Densidade da rede expressa o grau de conectividade da mesma. É definida pela razão entre o número de conexões existentes e o número máximo possível de conexões dentro da rede. Uma rede completa é aquela que possui a máxima densidade possível, ou seja, todos os vértices estão ligados entre si. Vale observar que essa medida é afetada pelo número de vértices, visto que é mais fácil uma rede com poucos vértices ser mais densa do que uma rede com milhares de vértices. Portanto, é importante comparar essa medida somente entre redes de tamanho similar.

O conceito de distância é relativo aos vértices. A distância entre dois vértices é a distância geodésica dos mesmos, ou seja, o menor caminho entre os vértices *A* e *B*.

A coesão é calculada como a média das distâncias entre cada par de vértices da rede. Uma coesão baixa indica que a rede não impõe limites ao fluxo da informação. Pelo lado positivo, a informação é democrática e bem difundida, o que facilita o acesso à mesma. Pelo lado negativo, pode representar uma lentidão no processo, devido ao excesso da divulgação da informação.

2.4.2.2 Centro e Periferia

A centralidade é o termo utilizado para se referir à posição de um vértice dentro da rede, enquanto o termo centralização é usado para caracterizar uma rede inteira. A rede é altamente centralizada quando houver uma clara fronteira entre o centro e a periferia. Em uma rede altamente centralizada, a informação propaga-se facilmente, porém o centro da rede é indispensável para transmissão da informação. A seguir, são apresentadas diferentes maneiras para se medir a centralidade.

Dois vértices são adjacentes se possuem uma conexão entre eles. Em uma rede com arestas podemos distinguir o grau de entrada e o grau de saída. O grau de entrada de um vértice é o número de arestas que ele recebe. O grau de saída é o número de arestas que ele envia. No caso de uma rede bidirecionada, esses graus são iguais. Portanto, o grau de centralidade do vértice é o próprio grau dele, no caso das redes direcionadas definem-se duas centralidades. Ele mede o quanto um vértice está centralizado em relação aos demais vértices da rede.

O grau de centralidade é uma medida local que só leva em consideração as conexões que um vértice possui. De uma maneira mais geral, o grau de centralização da rede é calculado pela variação dos graus dos vértices dividida pelo grau máximo

possível de uma rede de mesmo tamanho. A Figura 2.1 exibe uma rede altamente centralizada.

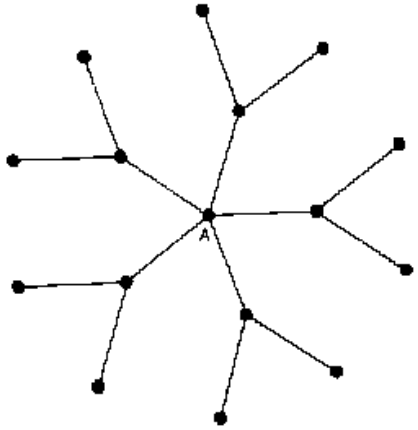

Figura 2.1 Rede altamente centralizada

Grau de proximidade é uma medida de cunho global, diferentemente do grau de centralidade, já que ela utiliza a distância entre outros vértices da rede e não somente as conexões diretas de um único vértice. Um vértice é considerado globalmente central se ele está a curta distância em relação a muitos outros vértices da rede. O grau de proximidade de um vértice *v* é igual ao número total de vértices da rede menos um dividido pela soma de todas as distâncias entre o vértice *v* e os demais. Ele mede o quanto um vértice está próximo ou pode alcançar os demais vértices da rede.

Outra abordagem a respeito da centralidade se apoia na ideia de que um vértice é mais central na medida de sua importância como um intermediário da comunicação na rede. O grau de intermedialidade de um vértice *v* é a proporção de todas as distâncias entre pares de vértice que passam por *v*. Ele mede o quanto um vértice exerce papel de mediador sob outros vértices ou está entre dois ou mais vértices.

2.4.2.3 Grupos Coesos

Essencialmente, as pessoas tendem a se organizar em grupos de interesse e gostos similares. Essa relação é geralmente simétrica (por exemplo, a amizade) e tem

uma tríade de fechamento [FOSTER *et al.* 1963] (por exemplo, meus amigos são, também, amigos entre si). Em termos de estrutura de rede, temos um aglomerado de arestas delimitando conjuntos distintos de vértices que estão interligados entre si. Estas partes de alta densidade dão origem às comunidades e aos subgrupos.

Resumindo, um grupo é feito por nós de uma malha densa (a maioria dos nós estão conectados diretamente uns aos outros) e fortemente vinculada (a maioria das conexões permanece dentro da rede, ao invés de se conectarem diretamente com o ambiente exterior).

2.4.2.4 Afiliação e Redes Dois Modos

Afiliação em redes ocorre quando temos dois ou mais grupos distintos de vértices onde cada elemento de um grupo só pode estabelecer conexões com os elementos dos demais grupos. Por exemplo, um grupo de atores (desenvolvedores) que só podem fazer conexões com elementos do grupo de eventos (arquivos de código). Por esta razão, quando ocorre essa afiliação, a rede é chamada de dois modos.

Através desse tipo de rede é possível fazer o estudo das relações entre atores e eventos. Além disso, pode-se dividir a rede em duas redes de um modo e estudar as relações da perspectiva dos atores (quais desenvolvedores mudaram o mesmo arquivo de código) ou eventos (que arquivos são mudados em conjunto). A Figura 2.2 mostra como é feita esta transformação.

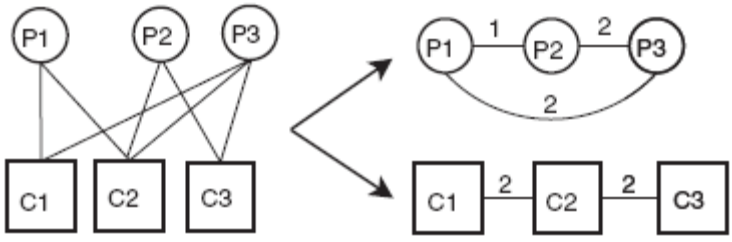

Figura 2.2 Transformação de uma rede de 2-modos para duas redes de 1-modo

2.4.2.5 Pontes e *Brokers*

Em termos de estrutura de rede, uma ponte é uma aresta, cuja remoção implica no aumento do número de componentes conectados da rede. *Brokers* são os vértices das extremidades da ponte. Eles têm um papel fundamental na disseminação da informação de um grupo para outro. A Figura 2.3 destaca as pontes de um grafo.

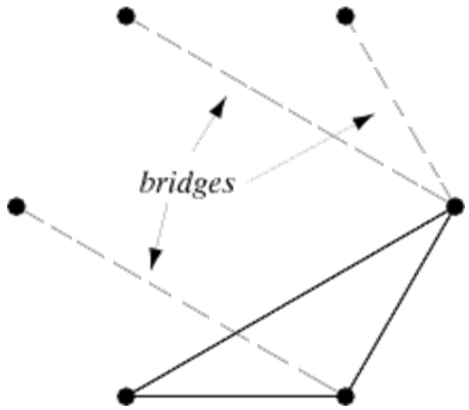

Figura 2.3 Pontes destacadas em pontilhado

2.4.3 Redes Sociais e Desenvolvimento de Software

A natureza distribuída do desenvolvimento de software e o uso de canais computadorizados para se comunicar produzem um ambiente particularmente atrativo para a utilização de SNA a fim de investigar a evolução e compreensão do processo de desenvolvimento do software.

Rosso (2009) construiu e analisou uma rede social sobre o projeto de desenvolvimento do servidor *web Apache* [APACHE 2010]. O servidor *web Apache* é um respeitado projeto de código aberto (*open source*). De acordo com a *Netcraft* [NETCRAFT 2010], o *Apache* lidera o mercado de servidores web, com 54% da fatia, contra 25% do Microsoft IIS.

Os vértices na rede social do *Apache* são os desenvolvedores e a relação utilizada para conectá-los é o fato deles terem trabalhado no mesmo arquivo de código. Utilizando o repositório de dados desse projeto é possível extrair as informações sobre quais desenvolvedores contribuíram para cada arquivo do projeto. Outras fontes também podem ser usadas para extrair essas relações sociais entre os desenvolvedores, tais como: entrevistas e questionários, listas de email, *bug databases*, etc.

O trabalho de Rosso (2009) concluiu que o uso de SNA auxilia o arquiteto do projeto a entender e gerenciar como os desenvolvedores interagem, identificar quem são os desenvolvedores trabalhando nos vários componentes de software, encontrar pontos chave na comunicação e coordenação do projeto e identificar os *brokers* do projeto. Após a análise, ações como reorganização da equipe ou da arquitetura do projeto podem ser tomadas. Amrit *et al.* (2004) realizou um trabalho semelhante ao de Rosso.

2.5 Considerações finais

Neste capítulo, apresentamos o que é construção de software e quais os problemas que podem ocorrer nesse processo, introduzimos o conceito de *awareness* e como obter proveito sobre essa prática. Também foi dissertado sobre gestão de conhecimento, e como esse processo se manifesta em um ambiente de desenvolvimento. A respeito das redes sociais, vimos a sua definição geral, a notação matemática, os conceitos e métricas utilizados para analisar a rede, como utilizar essa análise no ambiente de desenvolvimento de software e quais as conclusões obtidas dessa utilização.

No próximo capítulo é feita uma apresentação da tecnologia que foi utilizada no desenvolvimento do *website* deste projeto. Após essa apresentação, é realizado um detalhamento, não muito específico, de cada um dos componentes dessa tecnologia.

3 *ASP.NET MVC*

Neste capítulo é apresentada uma visão geral sobre o novo *framework* da Microsoft utilizado no desenvolvimento de aplicações *web*, nomeado ASP.NET MVC. Detalhes conceituais e técnicos, assim como o funcionamento dos principais componentes desse *framework,* são apresentados.

3.1 Introdução

ASP.NET MVC faz parte do *framework* para o desenvolvimento de aplicações *web* ASP.NET da Microsoft. Dentro da plataforma .NET, ele é um dos dois diferentes modelos utilizados para criar esse tipo de aplicação. O outro modelo utilizado é denominado *Web Forms*. Uma aplicação MVC (*Model-View-Controller)* é projetada e construída com base em três conceitos, como apresentado na Figura 3.1.

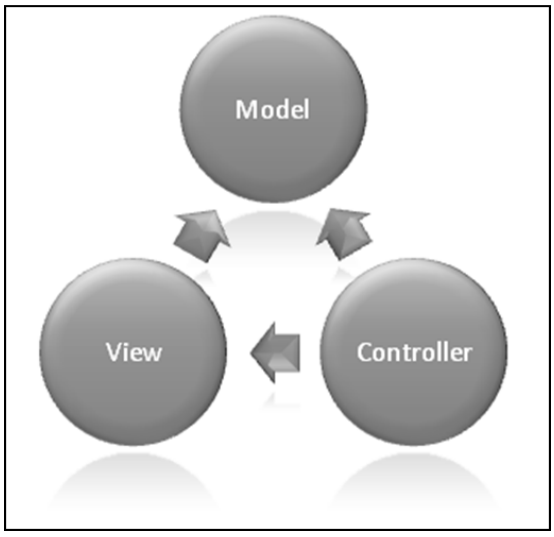

Figura 3.1 Arquitetura MVC

- Os modelos são as classes que representam objetos do domínio em que o sistema está inserido. Esses objetos de domínio encapsulam os dados armazenados em um banco de dados, bem como o código utilizado para manipular estes dados e aplicar a lógica de negócios específica do domínio;
- As *views* são páginas geradas dinamicamente que correspondem à apresentação da aplicação. Em ASP.NET, elas são tipicamente uma marcação HTML;
- O controlador é uma classe especial que gerencia a relação entre as *views* e os modelos. Ele se comunica com o modelo e decide qual *view* deve ser exibida para atender cada requisição do usuário.

Essa separação de conceitos proporciona uma maior flexibilidade para construir e manter a aplicação. Por exemplo, ao separar as *views* dos controladores e modelos, é possível modificar a aparência da aplicação sem aplicar qualquer modificação na lógica de negócio. Também é possível dividir o trabalho por especialidade. Por exemplo, *designers* podem trabalhar nas *views* enquanto desenvolvedores de sistemas trabalham nos modelos e controladores [ASP.NET 2010].

3.2 Routing

Ao invés de ter relações diretas entre URL e arquivos residentes no disco rígido, como acontece no *Web Forms*, um *website* desenvolvido utilizando o ASP.NET MVC mapeia as URLs em ações de um controlador. Este mapeamento possui dois objetivos principais: (a) identificar as requisições de entrada e mapeá-las em uma ação de um controlador, executando esta ação; e (b) construir URLs de saída que correspondem a ações de um controlador, permitindo que estas sejam executadas através de métodos como *RedirectToAction()*.

Toda aplicação ASP.NET MVC precisa de pelo menos uma *rota* para definir como a aplicação deverá lidar com as requisições do usuário. Aplicações de grande porte podem ter dezenas de rotas definidas. Uma rota consiste em diversos segmentos da URL (um segmento é aquilo que está entre as barras '/' do diretório virtual de uma URL), onde cada segmento corresponde a um parâmetro. A Tabela 3.1 mostra a rota padrão do ASP.NET MVC.

Padrão da URL	**Valores Iniciais**	**Exemplos de URLs que combinam com o padrão definido**
{controller}/{action}/{id}	new {controller ="home", action="index", id=""}	/users/edit/10 /users/create /users /

Tabela 3.1 Rota padrão do MVC

O valor do parâmetro {*controller*} é utilizado para instanciar o controlador que irá gerenciar a requisição. Por convenção, o ASP.NET MVC anexa o sufixo "Controller" ao valor desse parâmetro e busca uma classe com esse nome (*case insensitive*) que implemente a interface *System.Web.Mvc.IController*.

O valor do parâmetro {*action*} é usado para indicar qual método do controlador deverá ser executado para tratar a requisição. Por exemplo, a URL "/users/create" irá invocar o método *Create* do controlador *UsersController*.

Já o parâmetro {*id*} representa o argumento da ação. Caso a ação não possua argumentos, o valor deste parâmetro é vazio. Por exemplo, a URL "/users/edit/10" irá invocar o método *Edit(int id)* do controlador *UsersController* atribuindo o valor 10 ao parâmetro id.

Também é possível criar rotas com número indefinido de parâmetros, além de ser possível adicionar restrições em uma determinada rota. A Tabela 3.2 exibe outros exemplos de padrões que podem ser criados.

Padrão na URL	Exemplos de URLs que combinam com o padrão
service/{action}-{format}	/service/display-xml
{reporttype}/{year}/{month}/{date}	/sales/2008/1/23

Tabela 3.2 Exemplos de rota

3.3 Controladores

Controladores, dentro do padrão MVC, são responsáveis por responder a solicitações do usuário, realizando mudanças no modelo de acordo com essa solicitação. Desta maneira, os controladores estão focados no fluxo da aplicação, tratando os dados recebidos e fornecendo-os para as *views* relevantes [CONERY *et al.* 2009].

Para uma classe ser considerada um controlador em um projeto MVC, ela precisa implementar a interface *IController* e ter o seu nome terminado com o sufixo *Controller*, necessário para que o sistema de *routing* identifique o controlador. Uma classe criada como controlador geralmente herda da classe abstrata *System.Web.Mvc. Controller*, que por sua vez implementa a interface *IController*. Embora essa classe abstrata tenha sido criada para servir de base para todos os controladores, por fornecer diversos recursos às suas subclasses, é possível trocá-la por qualquer outra classe que implemente a interface *IController*.

Todo método público de um controlador é considerado uma ação e pode ser acessado por um endereço URL, desde que haja uma rota que faça essa relação. A ação é a unidade mais granular de resposta a uma requisição do usuário. Ela é responsável por tratar a requisição e retornar uma resposta que será exibida para o usuário, que geralmente é um código em HTML. Por convenção, toda ação retorna um resultado que deriva da classe abstrata *ActionResult*. Os possíveis tipos de resultados da ação estão descritos na Tabela 3.3.

Tipos de ActionResult	Descrição
EmptyResult	Representa uma resposta nula ou vazia. Não faz nada.
ContentResult	Escreve o conteúdo específico como texto (chama o método *ToString()* do objeto) diretamente na resposta.
JsonResult	Serializa o objeto fornecido no formato JSON e o escreve na resposta.
RedirectResult	Redireciona o usuário para a URL fornecida.
RedirectToRouteResult	Redireciona o usuário para uma URL através do sistema de *Routing*.
ViewResult	Utiliza o mecanismo de *view* para desenhar a *view* na resposta.
PartialViewResult	Similar ao *ViewResult*, mas desenhando uma *view* parcial, tipicamente em resposta à uma chamada AJAX.
FileResult	Serve como a classe base para um conjunto de resultados que manipulam arquivos.
FilePathResult	Deriva de *FileResult* e escreve um arquivo na resposta baseado no caminho do arquivo.
FileContentResult	Deriva de FileResult e escreve um vetor de bytes na resposta.
FileStreamResult	Deriva de FileResult e escreve um bloco de memória na resposta.
JavaScriptResult	Utilizado para executar no cliente um código *JavaScript* fornecido pelo servidor.

Tabela 3.3 Tipos de ActionResult

A classe abstrata *Controller* fornece alguns métodos de suporte para a criação destes *ActionResults* (Tabela 3.4). Além dos métodos que criam explicitamente um *ActionResult*, é possível criar ações que retornem um *ContentResult* de maneira implícita. Para isso, basta colocar um tipo que não seja um *ActionResult* como retorno da ação. A vantagem dessa abordagem é facilitar o entendimento das intenções do desenvolvedor para com aquela ação.

Método	Descrição
Redirect(…)	Retorna um *RedirectResult.*
RedirectToAction(…)	Retorna um *RedirectToActionResult.*
RedirectToRoute(…)	Retorna um *RedirectToRouteResult.*
View(…)	Retorna um *ViewResult.*
PartialView(…)	Retorna um *PartialViewResult.*
Content(…)	Retorna um *ContentResult.*
File(…)	Retorna um objeto que deriva de *FileResult.*
Json(…)	Retorna um *ContentResult* contendo a serialização do objeto no formato JSON.
JavaScript(...)	Retorna um *JavaScriptResult.*

Tabela 3.4 Métodos de ajuda para criação de ActionResults

Para uma mesma ação podem existir diferentes métodos no controlador, de acordo com os diferentes comandos (*verbs*) pelos quais a ação pode ser disparada, sendo os mais comuns GET e POST. Para definir a qual comando um determinado método

deve responder, basta decorar o método com o atributo "[AcceptVerbs(HttpVerbs.X)]", onde X é o comando desejado. Desta maneira, pode-se ter um método chamado quando o usuário solicita uma URL utilizando GET e outro método chamado quando o usuário solicita a mesma URL utilizando POST.

A passagem dos dados da *view* para as ações do tipo POST é feita através de *Model Binders*. Os m*odel binders* são responsáveis por transformar os dados da *view* em um único objeto, permitindo à ação acessar a informação através desse objeto (Figura 3.2). O ASP.NET *framework* é distribuído com três *model binders*, que são suficientes para realizar a maioria das transformações. Porém, no caso de tipos complexos de dados, pode ser necessária a criação de um *model binder* customizado.

```
[AcceptVerbs(HttpVerbs.Post)]
public ActionResult Edit(Product product)
{
    //...
}
```

Figura 3.2 O objeto "product" é construído através de *model binders*

Além dos atributos indicando o comando, outro tipo de atributo, chamado de filtro de contexto, pode ser associado a ações ou controladores. Os filtros de contexto determinam um comportamento específico para uma ação. Existem três tipos de filtros disponíveis no *framework* (filtros customizados também podem ser criados), sendo eles:

- *Authorize*: utilizado para restringir o acesso a um controlador ou ação;
- *HandleError*: utilizado para indicar qual ação será executada em caso de exceção;
- *OutputCache*: utilizado para fornecer *cache* de saída para as ações.

3.4 Views

A *view* é responsável por fornecer ao usuário uma interface visual. Uma referência aos objetos componentes do modelo é concedida para a *view* e ela se encarrega de transformar esse modelo em um formato apresentável para o usuário. Em ASP.NET MVC, esse procedimento consiste na leitura de um *ViewDataDictionary* construído pelo controlador (através da propriedade *ViewData*) e na transformação desses dados em um conteúdo HTML.

As *views* podem ser não tipadas ou fortemente tipadas. As Figuras 3.3 e 3.4 exibem, respectivamente, as diferenças entre as suas construções e como cada uma delas acessa os dados.

Controller

```
public ActionResult List()
{
    var products = new List<Product>();
    for (int i = 0; i <10; i++)
    {
        products.Add(new Product(ProductName = “Product ” + i));
    }
    ViewData[“Products”] = products;
    return View();
}
```

View

```
<%@ Page Language=“C#” MasterPageFile=“~/Views/Shared/Site.Master”
Inherits=“System.Web.Mvc.ViewPage<>” %>
<ul>
<% foreach(Pruduct p in (ViewData[“Products”] as IEnumerable<Product>>)) {%>
    <li><%= Html.Encode(p.ProductName) %></li>
<% } %>
</ul>
```

Figura 3.3 View não tipada

```
Controller

public ActionResult List()
{
    var products = new List<Product>();
    for int I = 0; I <10; i++)
    {
        products.Add(new Product(ProductName = “Product ” + i));
    }
    return View(products);
}

View

<%@ Page Language=“C#” MasterPageFile=“~/Views/Shared/Site.Master”
Inherits=“System.Web.Mvc.ViewPage<IEnumerable<Product>>” %>
<ul>
<% foreach(Product p in Model) {%>
    <li><%= Html.Encode(p.ProductName) %></li>
<% } %>
</ul>
```

Figura 3.4 View fortemente tipada. O sublinhado destaca as diferenças

A classe *ViewPage* – classe base de todas as *views* – possui uma propriedade do tipo *HtmlHelper*, chamada de *Html*. Essa propriedade possui uma série de métodos que ajudam na criação de marcadores HTML e na transformação dos dados do modelo nesses marcadores. Por exemplo, o método *Encode* retorna o resultado do método *ToString* do objeto que é passado como parâmetro, convertendo caracteres especiais (como caracteres acentuados, por exemplo) para o seu padrão HTML.

O ASP.NET MVC possui um mecanismo de *views* que pode ser substituído ou alterado. Esse mecanismo é responsável por interpretar o *ViewResult* (retorno de uma ação) e enviar a resposta dessa interpretação para o usuário. Por padrão, o mecanismo utilizado é o *WebFormsViewEngine,* responsável por interpretar páginas em ASP.NET e transformá-las em código HTML. Situações em que a troca de mecanismo é comum são:

- Desejo de programar a *view* em uma linguagem diferente, como *Ruby* ou *Python*;
- Necessidade de um código HTML mais conciso que respeite padrões mais rígidos;
- Exibir resultados que não são código HTML, tais como, gráficos, PDF, RSS, etc.

3.5 Validação

O ASP.NET MVC 2.0 introduziu uma nova maneira de se realizar a validação dos dados, tanto no cliente como no servidor. Essa validação é feita através de *DataAnnotations*, que são atributos utilizados para decorar os campos de um modelo de dados. Esses atributos definem os padrões de validação que serão utilizados pela aplicação, como campos obrigatórios, tamanho mínimo de strings, entre outros. Assim como na maior parte do *framework*, é possível customizar esses atributos ou criar um novo mecanismo de validação.

Para realizar a validação no cliente, após decorar o modelo com os atributos, é necessário que o navegador do usuário esteja habilitado a executar *JavaScript* e que as linhas de código da Figura 3.5 estejam inseridas na *view*. A validação no servidor não requer qualquer configuração adicional e as informações relativas à validação podem ser visualizadas na propriedade *ModelState* do controlador (Figura 3.6).

```
<head runat="server">
    <script src="<%= Url.Content("~/Scripts/jquery-1.4.1.min.js") %> type="texte/javascript"></script>
    <script src="<%= Url.Content("~/Scripts/MicrosoftAjax.js") %> type="texte/javascript"></script>
    <script src="<%= Url.Content("~/Scripts/MicrosoftMvcAjax.js") %> type="texte/javascript"></script>
   <script src="<%= Url.Content("~/Scripts/MicrosoftMvcValidation.js") %> type="texte/javascript"></script>
    <% Html.EnableClientValidation();%>
</head>
```

Figura 3.5 Código da *view* necessário para validação no cliente

```
[AcceptVerbs(HttpVerbs.Post)]
public ActionResult Edit(Product product)
{
    if(ModelState.IsValid)
    {
        db.SaveChanges(product);
       ViewData["Message"] = product.ProductName + " Updated";
    }
    else
    {
        ViewData["product"] = product;
        return View();
    }
}
```

Figura 3.6 Propriedade *ModelState*

3.6 Considerações Finais

Nesse capítulo foram descritos, de forma resumida, os principais componentes do novo *framework* da Microsoft para desenvolvimento de aplicações *web*, ASP.NET MVC. Para maiores informações consulte Conery *et al.* (2009) e Walter (2010). No próximo capítulo será apresentada uma proposta para a criação de uma rede social entre desenvolvedores de software, visando inserir conceitos de gestão do conhecimento e *design awareness* em um ambiente de desenvolvimento de software, com o objetivo de aumentar a produtividade dos desenvolvedores durante o desenrolar de um processo de construção de software.

4 *Solução Proposta*

4.1 Uma Rede Social Voltada para a Construção de Sistemas

Este trabalho propõe o uso de uma rede social para oferecer aos desenvolvedores de um projeto de software o máximo de informações sobre o que ocorreu e o que está ocorrendo com outros desenvolvedores que participam de sua equipe. As informações manipuladas por esta rede social são coletadas a partir de duas fontes de dados: o sistema de controle de versões do projeto e dados incluídos pelos desenvolvedores.

Para alcançar esse objetivo, construiu-se um *website* responsável por administrar os usuários da rede social (os desenvolvedores participantes do projeto) e um conjunto de *add-ins* desenvolvidos na IDE *Microsoft Visual Studio 2008*. A Figura 4.1 apresenta os principais componentes da solução proposta.

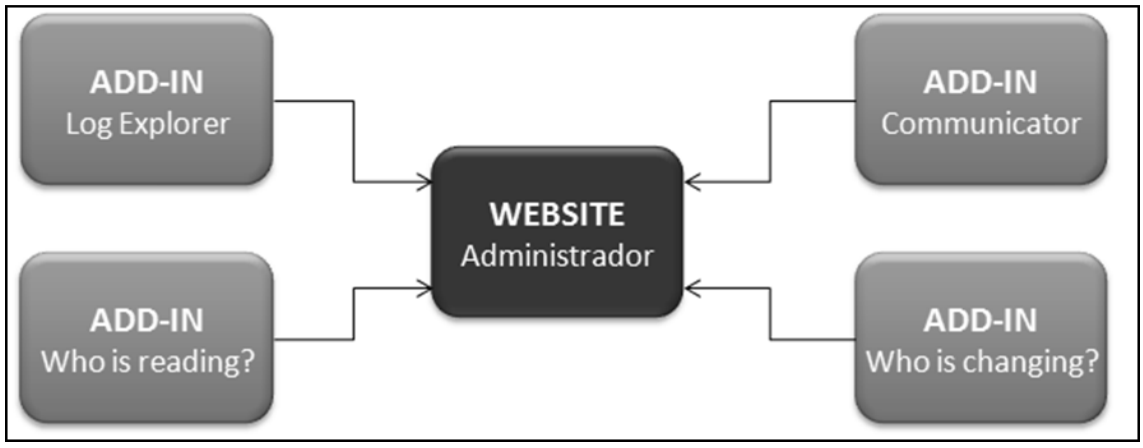

Figura 4.1 Diagrama da relação entre os componentes da rede social

4.1.1 Sistema de Controle de Versões

Em meio ao processo de desenvolvimento de software, diversas pessoas participam na produção de documentos sobre o projeto e de artefatos, como módulos de código-fonte. Esta característica exige que se tenha certo controle sobre as alterações feitas nestes artefatos, a fim de evitar que uma pessoa ou equipe sobrescreva erroneamente o que foi feito por outra [WIRES e FEELEY 2007].

Diante da necessidade de centralizar e organizar o desenvolvimento de software em equipe, surgiram os sistemas de controle de versões. O objetivo destes sistemas é gerenciar arquivos, que são normalmente de documentação ou de código, mantendo um registro sobre as alterações que foram feitas em cada um destes arquivos, bem como por quem, a que horas e em que dia [TROMBETTA *et al.* 2008].

Os softwares de controle de versões armazenam todos os arquivos utilizados pela equipe de desenvolvimento em um local denominado repositório. A alteração destes arquivos é realizada em um modelo distribuído. Nesse modelo, cada desenvolvedor trabalha com uma cópia local dos arquivos que está manipulando. A atualização da cópia no repositório compartilhado (*commit*) é feita ao final das modificações desejadas.

Diferentes informações são armazenadas sobre cada atualização de cada arquivo. Embora essas informações possam variar dependendo do sistema de controle de versões utilizado, as mais comuns são a versão do arquivo (cada alteração gera uma versão nova), o dia e hora da alteração, o nome do usuário que fez a alteração e um comentário a respeito do que foi modificado. A atribuição de uma nova versão a cada arquivo no momento em que é feita a atualização da cópia do repositório permite estabelecer um histórico sobre as modificações feitas, bem como resgatar versões anteriores de um documento ou código.

Existem diversos sistemas que podem ser utilizados no controle de versões de arquivos. Alguns exemplos são o CVS (*Concurrent Version System*) [CVS 2010], *Subversion* [SUB 2010] e o VSS (*Visual SourceSafe*) [VSS 2010].

4.1.2 Os Componentes da Rede Social Proposta

Tendo em vista as dificuldades enfrentadas pelos desenvolvedores na construção de software, criou-se uma rede social, formada por um conjunto de ferramentas, que visa aumentar a comunicação e propagação da informação, assim como difundir o conhecimento adquirido entre os desenvolvedores.

Os dados utilizados pela rede social são extraídos do sistema de controle de versões. A partir desses dados, é possível exibir no ambiente de desenvolvimento de software algumas informações relevantes sobre os projetos, como, por exemplo:

- Quais desenvolvedores fazem parte do projeto?
- Com que frequência eles atualizam os documentos e o código do projeto?
- Quais os objetivos destas atualizações?

Além dessas informações vindas do repositório de dados, a rede social também produz e exibe novas informações para os desenvolvedores, como:

- Mensagens que um determinado desenvolvedor acha pertinente divulgar sobre um documento ou arquivo de código;
- Frequência com que os desenvolvedores lêem um arquivo de código-fonte;
- Mensagens mais relevantes, tanto do repositório do sistema de controle de versões quanto as mensagens criadas pelos próprios desenvolvedores, identificadas por pontos de prestígio.

Os pontos de prestígio são o mecanismo utilizado para destacar a relevância das mensagens contidas no sistema. O administrador do sistema inicialmente distribui um

conjunto de pontos de prestígio para cada usuário. Em seguida, cada usuário pode distribuir os pontos atribuídos a ele entre as mensagens registradas no sistema. Quanto mais pontos uma mensagem contiver, mais relevante ela será.

4.2 Website do Projeto

O *website* do projeto é a interface administrativa do sistema. É através dele que controlamos os perfis dos usuários (na maioria, desenvolvedores participantes da equipe do projeto) da rede social. As tarefas possíveis de se realizar pelo *website* são: criar um novo usuário, editar as informações dos usuários, atribuir pontos de prestígio para cada usuário distribuir pelas mensagens que são do seu interesse e alterar os privilégios de um usuário.

Um usuário da rede social apresenta as seguintes informações:

- Nome: nome do usuário por extenso;
- Usuário: *login* do desenvolvedor utilizado no sistema de controle de versões. Esse campo também será o *login* do usuário no site e deve ser único;
- Senha: senha utilizada em conjunto com o *login* para ter acesso ao conteúdo do site. A senha deve ter no mínimo seis caracteres;
- Perfil: Controla os privilégios do usuário no site. Existem dois perfis: Administrador e Desenvolvedor. Os desenvolvedores têm acesso somente à lista de usuários da rede, enquanto os administradores podem realizar qualquer uma das tarefas anteriormente mencionadas;
- Endereço de e-mail: endereço de e-mail do usuário;
- Pontos de prestígio: indica quantos pontos de prestígio ainda restam para o usuário distribuir entre as mensagens que considerar relevantes;

- Foto de perfil: foto do usuário. Esse é o único campo não obrigatório durante o cadastro ou edição de um usuário.

Uma das características do *website* é o seu visual limpo e simples. Para todo o site, uma única folha de estilo (CSS) foi utilizada, embora exista um CSS exclusivo para impressão. Isso torna um *redesign* completo de todo o *website* uma tarefa extremamente fácil, já que é preciso realizar alterações em um único lugar. Alguns padrões de usabilidade, como menus superior, inferior e lateral, também estão presentes.

Durante a criação ou alteração de um usuário, é feita validação de seus dados tanto no cliente quanto no servidor. A validação no cliente ocorre no momento em que um campo perde o foco. Assim que o usuário termina de preencher um determinado campo é possível determinar se o valor inserido é válido ou não. Porém, esse tipo de validação só é possível se o usuário deixar habilitado o uso de *javascript* no navegador em que estiver visualizando o site. A validação no servidor ocorre no momento em que o usuário submete as informações do formulário para o servidor. Esse tipo de validação é padrão e não requer nenhuma configuração específica.

Além das características mencionadas, o *website* apresenta autenticação, autorização e internacionalização. Autenticação diz respeito à necessidade de um usuário estar autenticado no *website*, ou seja, ter efetuado o *login*, por exemplo, para visualizar a lista de usuários é preciso estar autenticado. Algumas informações necessitam que o usuário autenticado tenha certo nível de privilégio para poder acessá-la, isto é, ter autorização, por exemplo, somente usuários com perfil de administrador podem editar informações. A internacionalização indica que o site está disponível em mais de um idioma; no trabalho realizado, em português e em inglês.

4.2.1 O *Add-in* Log Explorer

Este *add-in* tem como função principal exibir as mensagens deixadas por desenvolvedores ao submeterem alterações nos arquivos componentes do sistema. Ao selecionar determinado arquivo são exibidas no Log Explorer, em ordem decrescente de submissão, as mensagens submetidas para este mesmo arquivo. É possível filtrar o resultado de forma que apenas apareçam as mensagens submetidas por determinados desenvolvedores.

As informações exibidas para cada mensagem são: nome do desenvolvedor que realizou a alteração; imagem do desenvolvedor, caso possua; comentário deixado pelo desenvolvedor quando a alteração foi submetida; identificador da revisão gerada pelo sistema de controle de versões, referente à alteração em questão; data e hora em que a alteração foi submetida. O *add-in* também permite que os desenvolvedores concedam pontos de prestígio para as mensagens que consideram relevantes e que tenham sido deixadas por outros desenvolvedores.

4.2.2 O *Add-in* Communicator

Este *add-In* permite que os desenvolvedores deixem mensagens (espera-se que sejam relevantes), associadas a um arquivo, para serem apresentadas posteriormente a todos os desenvolvedores de forma similar ao que ocorre no Log Explorer.

As informações exibidas para cada mensagem são: nome do desenvolvedor que enviou a mensagem; imagem do desenvolvedor, caso possua; mensagem propriamente dita; data e hora em que a mensagem foi enviada.

Assim como as mensagens exibidas pelo Log Explorer, as mensagens do *Communicator* também podem receber pontos de prestígio. Porém, ao contrário do *add-in* anterior, o *Communicator* permite que as mensagens sejam removidas pelo desenvolvedor que as criou.

4.2.3 O *Add-in* Who is Reading?

Este *add-in* registra o momento em que os desenvolvedores focalizam e retiram o foco de um arquivo de código-fonte dentro do *Microsoft Visual Studio*. Além disso, o *add-in* exibe as informações registradas por ele no *banco de dados* do sistema, permitindo que os desenvolvedores saibam quem está lendo determinado arquivo, bem como os últimos leitores do mesmo arquivo.

Para evitar a exibição de informações em excesso, o *add-in* permite que seja configurado um intervalo de tempo para o qual as informações serão exibidas, ou seja, apenas as leituras ocorridas dentro deste período aparecerão na tela do *add-in*.

Além da exibição de quem está interessado em determinado arquivo, é exibida uma proporção que informa o interesse de cada leitor no arquivo. Esta proporção permite que um desenvolvedor saiba quem está demonstrando maior interesse nos módulos que ele desenvolveu; permite que um arquiteto identifique os desenvolvedores dos módulos que despertaram maior interesse no sistema, no sentido de premiar estes desenvolvedores ou organizar treinamentos específicos relacionados a estes módulos. Caso o desenvolvedor selecione um diretório ao invés de um arquivo, este visualiza a média dos desenvolvedores que leram os arquivos presentes naquele diretório. A média é a proporção das ocorrências de determinado evento gerado por um usuário (como por exemplo, visualização ou atualização de um arquivo) em relação ao todo.

4.2.4 O *Add-in* Who Changed?

Este *add-in* é similar ao "Who is Reading?" e inicialmente foi nomeado como "Who is Changing?". Porém, devido a uma mudança na modelagem, onde apenas a data de *check-in* passou a ser armazenada – antes também armazenava a data do *check-out* – seu nome foi alterado. A diferença principal é que neste caso são registrados e exibidos os arquivos alterados pelos desenvolvedores.

A filtragem por um intervalo de tempo, o comportamento do *add-in* caso um diretório seja selecionado e a exibição da proporção (neste caso informando a quantidade de alterações feitas por cada desenvolvedor) ocorre como no *add-in* anterior. Para o "Who Changed?" uma das vantagens da exibição da proporção é permitir que um desenvolvedor interessado em alterar determinado módulo, mas que tenha dúvidas em como fazê-lo, descubra quem deve procurar para aprender mais sobre o módulo.

4.3 A Arquitetura Proposta para a Solução

O sistema foi construído seguindo padrões de desenvolvimento para que a manutenção do mesmo ocorra de forma simples, quando necessária. Ele é dividido em diversos projetos, cada qual com uma finalidade específica, facilitando uma possível reutilização de bibliotecas. Por exemplo, o projeto do modelo de dados é referenciado por todos os outros projetos do sistema.

O projeto principal, onde se encontram o modelo das entidades descritas na especificação e as funcionalidades básicas, como acesso ao banco de dados e implementação das interfaces dos serviços, foi nomeado "*SocialAddIn.Foundation*" e é uma biblioteca referenciada por todos os outros projetos.

O projeto chamado "*SocialAddIn.LogReader*" é responsável por verificar o arquivo gerado pelo sistema de controle de versões utilizado pelos desenvolvedores, buscando as alterações aplicadas sobre os documentos registrados neste sistema e resgatando suas informações associadas, como data de submissão, mensagem atrelada e desenvolvedor responsável. Esse projeto é de fundamental importância, visto que é o responsável por "popular" o banco de dados acessado tanto pelo *website* quanto pelos *add-ins*. Esse projeto foi compilado e instalado como um serviço do Windows, que fica monitorando constantemente o diretório do sistema de controle de versões.

A comunicação dos a*dd-ins* com o banco de dados ocorre através de um conjunto de *web services,* cada qual oferecendo suporte a um determinado *add-in*. Existe um projeto, denominado "SocialAddIn.WebServices", responsável por armazenar todos os w*eb services*, que são arquivos ".svc" – extensão necessária para identificar um serviço – apontando para classes presentes no *SocialAddIn.Foundation*, que, de fato, implementam as funções disponibilizadas pelos w*eb services*. Cada *web service* disponibiliza um conjunto de funções que podem ser utilizadas pelas aplicações que o consomem.

Os *add-ins* utilizam essa estrutura para realizar suas operações. Eles consultam o modelo de dados declarado na biblioteca principal e consomem os seus respectivos w*eb services,* que por sua vez acessam o banco de dados alimentado pelo *website* e pelo serviço do Windows.

4.4 Distribuição do Trabalho de Desenvolvimento

O desenvolvimento desse projeto foi dividido entre os alunos que cursaram a disciplina Tópicos Avançados em Construção de Software (TACS) ofertada no primeiro semestre de 2010, no Programa de Pós-Graduação em Informática (PPGI) do Centro de Ciências Exatas e Tecnologia (CCET) da Universidade Federal do Estado do Rio de Janeiro (UNIRIO). A divisão do trabalho foi feita da seguinte forma:

- Who Changed? → Dois alunos do mestrado;
- Who is Reading? → Dois alunos do mestrado;
- Log Explorer e Communicator → Dois alunos do mestrado;
- Website administrador e implementação compartilhada entre os diversos *add-ins* → Autores deste trabalho.

Grande parte do projeto foi desenvolvida nos computadores pessoais de cada um dos participantes. Não foi utilizado um processo de software específico durante o desenvolvimento do trabalho, cabendo a cada grupo decidir como seria conduzido o seu trabalho. A comunicação entre os participantes ocorreu, principalmente, de três formas: e-mail, Bugzilla [UNIRIOTEC 2010] e encontros no laboratório da Universidade, sendo essa última a que se mostrou mais eficiente e produtiva.

Para testar o funcionamento do que foi desenvolvido pelos mestrandos, em conjunto com o dos graduandos, foi montado um servidor com os requisitos necessários em uma das máquinas do laboratório de Pesquisa em Distribuição e Redes. A única restrição exigida na implementação atual era que todas as máquinas que rodassem os *add-ins* estivessem na mesma rede que o servidor.

Devido à desistência dos alunos de Mestrado responsáveis pelo *add-in* "Who is Reading?", esse não foi desenvolvido.

4.5 O Modelo de Dados da Rede Social

Após a análise das funcionalidades pretendidas para a rede social, criou-se um banco de dados, no *Microsoft SQL Server 2008 Express* [MSSE 2010], com todas as entidades consideradas importantes para a representação do problema. A estrutura do banco de dados passou por algumas mudanças, devido a interações entre os envolvidos no projeto da rede social, até que chegasse ao modelo apresentado na Figura 4.2.

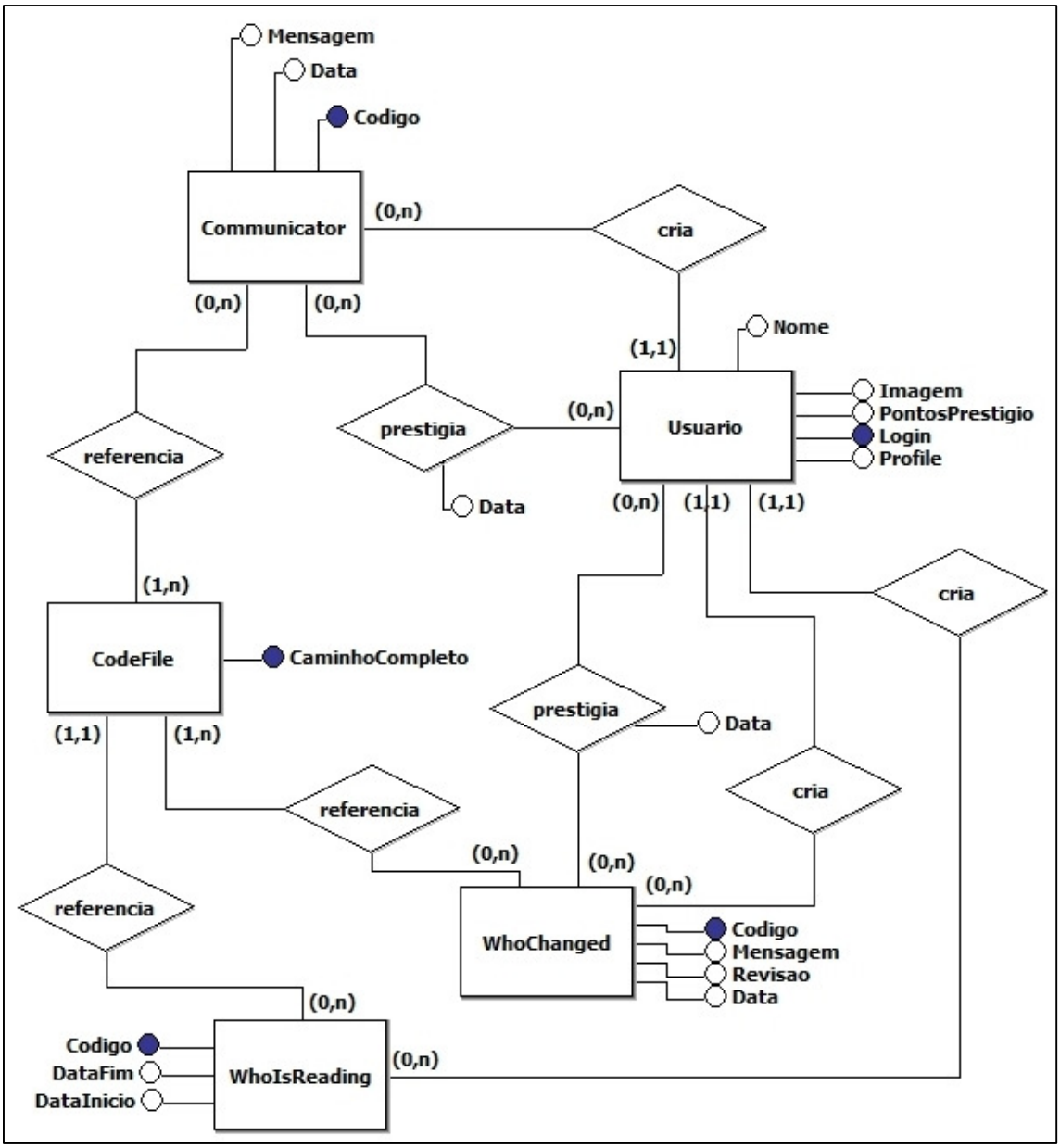

Figura 4.2 Modelo Conceitual do Banco de Dados

Utilizando a tecnologia *LINQ* [MICROSOFT 2010], foi feito um mapeamento direto entre as tabelas do banco de dados e as classes de modelo do projeto. O *Microsoft Visual Studio* disponibiliza uma classe, chamada *LINQ to SQL*, que gera todas as classes de modelo correspondentes a partir de um banco de dados criado no *Microsoft SQL Server*. Essa classe agiliza o processo de desenvolvimento, além de evitar possíveis problemas com relação à estrutura das classes, visto que o código é gerado de forma automática. Detalhando cada classe do projeto, temos:

- CodeFile: corresponde a um arquivo do projeto;
- Communicator: corresponde a uma mensagem enviada por um usuário;
- WhoChanged: corresponde a uma mensagem de um usuário no momento do *check-in* de um arquivo no sistema de controle de versões;
- PrestigePointsCommunicator: responsável por manter o histórico de pontos dados por um usuário para uma mensagem;
- PrestigePointsLog: responsável por manter o histórico de pontos dados por um usuário para uma mensagem gerada no momento do *check-in*.
- User: corresponde a um usuário (administrador do *website* ou desenvolvedor que utiliza algum dos a*dd-ins*).
- WhoIsReading: armazena o período (início e fim) em que um usuário visualizou um arquivo;
- CommunicatorCodeFile: responsável pela ligação entre a mensagem submetida pelo usuário e os arquivos aos quais a mensagem corresponde;
- LogCodeFile: responsável pela ligação entre a mensagem submetida pelo usuário no momento do *check-in* e os arquivos aos quais a mensagem corresponde;
- UserProfileImage: corresponde a uma imagem armazenada no Flickr e utilizada por um usuário do *website*.

Para que o projeto não ficasse restrito a uma determinada tecnologia, foi utilizada uma interface, denominada *IDataService*, na qual todas as funções de acesso aos dados foram declaradas. Para cada tecnologia diferente que se queira utilizar, é necessário que seja criada uma classe herdando desta interface e implementando seus métodos. Além dessa interface, existe outra classe, denominada *ModelFactory*, responsável pela transformação das classes geradas pelo mapeamento objeto-relacional (ORM – *Object-Relational Mapping*), por exemplo, *LINQ to SQL*, nas classes de

negócio utilizadas em todo o projeto. Essa abordagem diminui a área afetada por uma possível mudança de tecnologia, evitando que a base do projeto precise ser alterada.

4.6 Construindo o *Website* Administrador do Sistema

O *website* do sistema foi construído utilizando o *framework* ASP.NET MVC 2.0 [MVC 2010]. O *website* é um projeto de pequeno porte, desenvolvido por um único desenvolvedor durante um período de quatro meses. A folha de estilo CSS foi obtida em Gallery (2010), enquanto o logotipo e ícone do *website* foram criados pelo próprio desenvolvedor.

A fim de facilitar o desenvolvimento do projeto, o sistema de controle de contas de usuários distribuído com a plataforma ASP.NET foi utilizado na construção do *website*. Dessa maneira, o banco de dados com as contas dos usuários, necessárias para o mecanismo de autenticação nativo da plataforma ASP.NET, ficou separado do banco de dados utilizado pelos demais componentes do sistema. Em um ambiente de produção, essa prática seria desaconselhável, pois além de exigir a manutenção de dois bancos de dados, teria que ser criado um relacionamento entre as entidades que representam os usuários nestes dois bancos.

O projeto possui quatro controladores, sendo um de uso exclusivo para validação (*Validation*). O controlador *home* é responsável por exibir a página inicial do *website* e a página de informação sobre os autores. O controlador *Account* é responsável pelo controle de contas de usuário do *website*. Finalmente, o controlador *User* é responsável pela criação e edição dos usuários do sistema.

Apenas dois grupos de classes de modelo foram utilizados: um para a edição, criação e listagem de usuários e outro para fazer o acesso ao site. Além dos atributos padrão de validação, esses modelos foram decorados com atributos criados

especialmente para essa aplicação, como *ImageType*, *PropertyMustMatch* e *ValidatePasswordLength*. Filtros de contexto customizados foram utilizados para os seguintes fins:

- Garantir que um usuário esteja autenticado no website (*RequiresAuthentication*);
- Garantir que um usuário autenticado tenha o perfil necessário para acessar determinada ação ou controlador (*RequiresProfile*);
- Determinar que uma ação ou controlador não deve armazenar qualquer tipo de cache (*NoCache*).

Um procedimento AJAX foi utilizado para carregar de maneira assíncrona a lista de usuários do *website*. Essa lista foi exibida em uma tabela *jQuery* [JQUERY 2010] dentro de uma *view* parcial. A internacionalização do *website*, em português e em inglês, foi feita através de arquivos de *resources*.

4.7 O Projeto e a Construção dos *Web Services*

Web service é uma solução utilizada na integração de sistemas e na comunicação entre aplicações diferentes. Essencialmente, o *web service* faz com que os recursos da aplicação estejam disponíveis na rede de uma forma normalizada, permitindo que as aplicações-cliente recuperem, extraiam e utilizem os recursos fornecidos.

Neste projeto, a API (*Application Programming Interface*) utilizada para construir os serviços foi o WCF (*Windows Communication Foundation*) [WCF 2010]. Esta API foi projetada de acordo com os princípios da arquitetura orientada a serviço para suportar sistemas distribuídos, onde os serviços são consumidos por clientes, podendo cada cliente consumir múltiplos serviços e cada serviço ser consumido por múltiplos clientes.

Um serviço WCF é composto por três partes: (a) os arquivos que implementam o serviço que será fornecido; (b) *o Host environment*, que é o ambiente que abrigará o serviço; e (c) *os endpoints*, que são os pontos de conexão entre o cliente e o serviço. Cada *endpoint* expõe um conjunto de métodos previamente definidos em um contrato, além de definir um *binding*, responsável por especificar como o cliente se comunicará com o serviço e o endereço onde o *endpoint* se encontra.

4.7.1 Definindo os Web Services

Cada *add-in* possui o seu respectivo *web service*, que está declarado no projeto *SocialAddIn.WebServices* através de um arquivo com extensão ".svc". Este arquivo informa o nome completo (incluindo *namespace*) da classe que implementa os métodos que serão disponibilizados pelo serviço. A implementação dos métodos ocorre em duas etapas:

- **Definição da interface**: A interface informa que métodos serão disponibilizados pelo serviço. Cada método necessita do atributo [*OperationContract*], enquanto a interface depende do atributo [*ServiceContract*];
- **Implementação da interface**: Após a definição da interface, é necessário que uma classe implemente todos os métodos definidos por esta. Essa classe não necessita de um atributo específico e é implementada como qualquer outra classe.

Neste projeto, a escolha dos métodos presentes na interface foi feita a partir da especificação dos *add-ins* e de acordo com as demandas dos seus desenvolvedores, através do cadastro de pedidos pelo *Bugzilla* [UNIRIOTEC 2010]. A tabela com os métodos disponibilizados pelos serviços encontra-se no Anexo *I*.

4.7.2 Implementando os Web Services

Cada *web service* possui um conjunto de três classes organizadas em pastas no projeto *SocialAddIn.Foundation*, da seguinte forma:

- **Interfaces**: armazena as interfaces dos serviços. Estas classes seguem o padrão IxService, sendo x o nome do *add-in* correspondente;
- **Remote**: armazena as classes que implementam as interfaces. Estas classes seguem o padrão xRemote, sendo x o nome do *add-in* correspondente;
- **WCF**: armazena as classes que também implementam as interfaces. Neste caso, a implementação é apenas um encapsulamento das etapas necessárias para a execução de um método do Remote por um cliente do serviço. Dado um *endpoint*, o procedimento consiste em criar e abrir um canal de comunicação com o serviço para em seguida realizar uma chamada para o método correspondente. Após o retorno do método, o canal é fechado e o retorno é propagado. Estas classes seguem o padrão xService, sendo x o nome do a*dd-in* correspondente.

A Figura 4.3 ilustra essa estrutura. A pasta *Controllers* possui diversos controladores que gerenciam qual o tipo de serviço está sendo utilizado. Cada controlador armazena uma interface que deve ser instanciada de acordo com a forma que se deseja acessar os dados. No momento, estão disponíveis duas formas: *web services* (na pasta *WCF*) e local (na pasta *Remote*). Utilizando local, o acesso aos dados será realizado na máquina onde a aplicação estiver sendo executada. Ao optar por *web services*, cada chamada a um método irá instanciar um cliente correspondente à interface e irá executar o método de mesmo nome neste cliente, como mencionado acima na explicação sobre a pasta WCF. O cliente, neste caso, pode acessar um *endpoint* publicado em outra máquina. Se por acaso o usuário utilizar um *Mock* para

testar o sistema, ou qualquer outra forma de acesso aos dados, ele pode implementá-lo utilizando a interface correspondente.

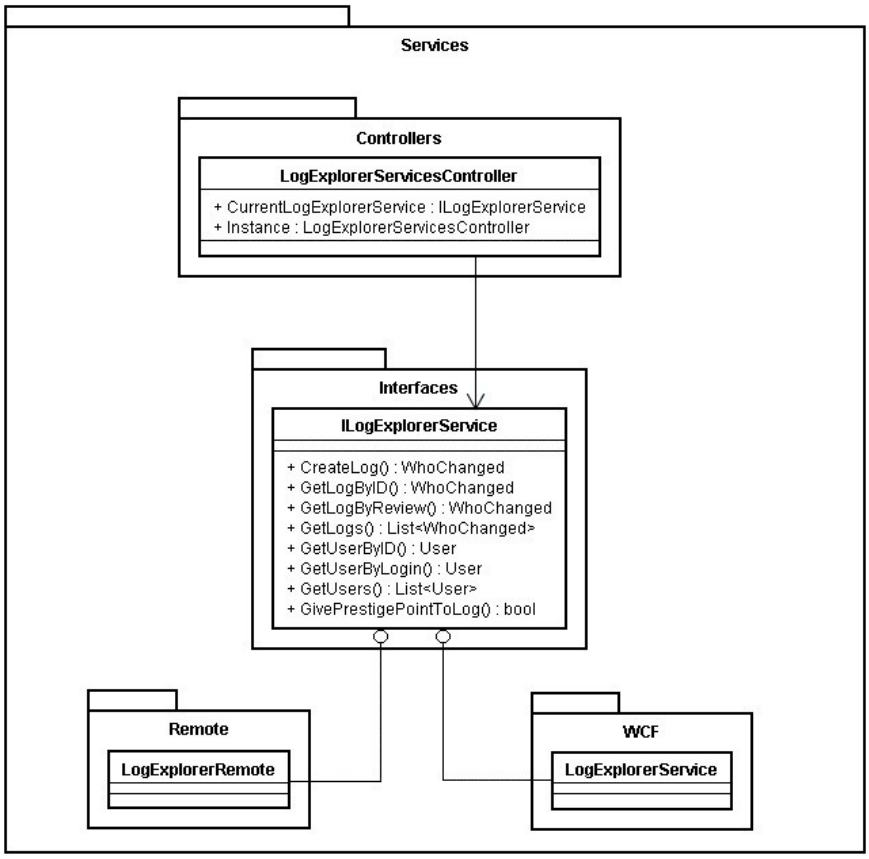

Figura 4.3 Exemplo do relacionamento das classes dos web services

Além dos arquivos com extensão ".svc", o projeto dos serviços também possui um arquivo XML de configuração com a extensão ".config" onde são armazenadas variáveis globais que podem ser usadas pela aplicação, como, por exempo, *connection strings* para os bancos de dados. Neste arquivo de configuração também são definidos os *endpoints* para os serviços. As propriedades mais importantes de um *endpoint* são:

- *Address*: é o endereço propriamente dito do serviço, ou seja, o local onde o serviço está hospedado. Pode ser definido dinamicamente, no momento em que o cliente do serviço for criado;

- *Binding*: define qual o tipo de *binding* será utilizado pelo serviço. O *binding* define diversas propriedades do serviço, como o tempo que ele pode ficar aberto aguardando uma resposta e a quantidade máxima de *bytes* transmitidos de uma vez;
- *Contract*: nome completo da interface que o serviço expõe.

4.8 Considerações Finais

Este capítulo apresentou a rede social proposta para aumentar a comunicação entre os desenvolvedores de um projeto de software. Essa rede foi implementada como um conjunto de *add-ins* do *Microsoft Visual Studio 2008* e um *website*. Os *add-ins* trazem para o ambiente de desenvolvimento um conjunto de informações relevantes para os desenvolvedores e que estão armazenadas de forma centralizada no *website*. Além de armazenar e fornecer acesso às informações, o *website* atua como uma interface administrativa do sistema. Detalhes técnicos do processo de construção desse sistema foram apresentados. O próximo capítulo apresenta um exemplo de uso da rede social proposta.

5 *Exemplo de Uso*

Neste capítulo são exibidas algumas telas das ferramentas apresentadas no capítulo anterior, com o objetivo de demonstrar um exemplo prático de como elas podem ser utilizadas.

5.1 Website

O *website* do projeto apresenta um *layout* simples. A sua página principal (Figura 5.1) não apresenta nenhuma informação, apenas um menu com *links* para as outras áreas do site. No canto superior direito é possível observar a área de *login* do site e as opções para troca de idioma.

Figura 5.1 Tela inicial do *website*

Depois de efetuada a autenticação no site, é possível navegar pela lista de usuários registrados (Figura 5.2). Ao clicar na imagem do usuário será exibido um formulário para edição do mesmo. Durante essa edição é possível atribuir mais pontos de prestígio para esse usuário.

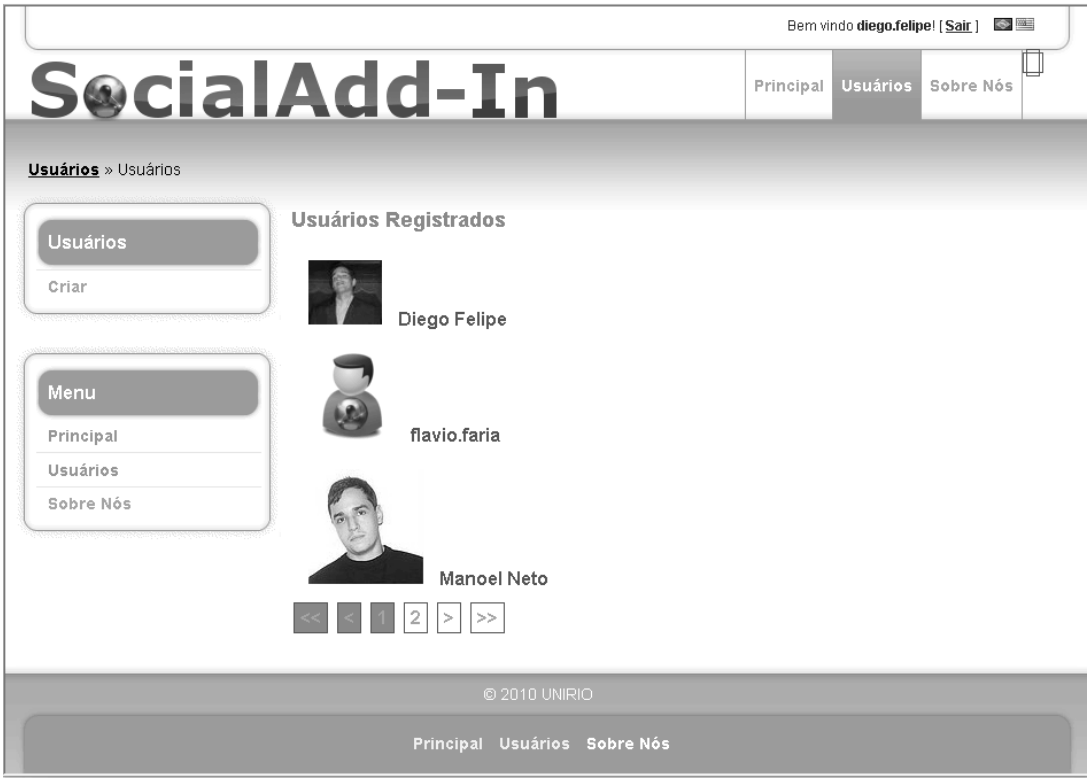

Figura 5.2 Lista de usuários registrados na rede social

A principal funcionalidade do *website* é o cadastro de usuário. A partir desse cadastro, os usuários podem utilizar os *add-ins* para gerar novos conteúdos. A Figura 5.3 exibe o formulário de cadastro preenchido e com erros detectados pelo mecanismo de validação no cliente.

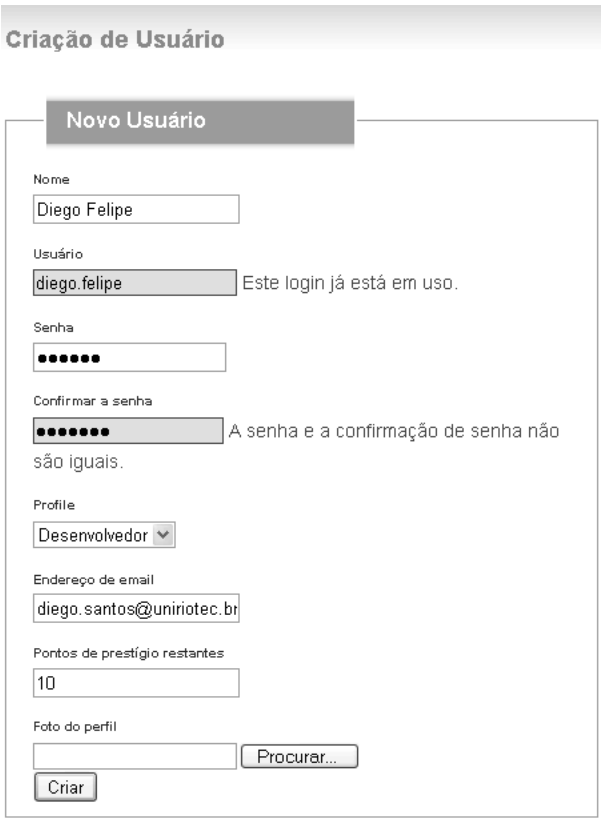

Figura 5.3 Formulário de cadastro de Usuário

5.2 Who Changed?

Este *add-in* exibe a lista de usuários que modificou o arquivo atualmente selecionado no ambiente de desenvolvimento de software. Ao dar foco em um arquivo de código no ambiente, o *add-in* automaticamente mostra para o usuário corrente quais desenvolvedores fizeram alterações no arquivo, além de calcular a proporção das alterações realizadas por cada desenvolvedor (Figura 5.4).

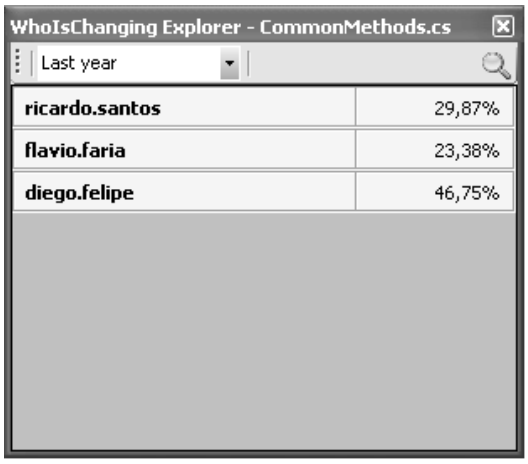

Figura 5.4 Lista de usuários que alteraram o arquivo "CommonMethods.cs"

Ao clicar na lupa no *add-in*, uma janela com a estrutura de diretórios do projeto é criada. Através dessa janela é possível visualizar as mesmas informações do *add-in* de maneira agregada, ou seja, visualizar a lista de usuários que fizeram alterações em um pacote do projeto ou no projeto como o todo (Figura 5.5).

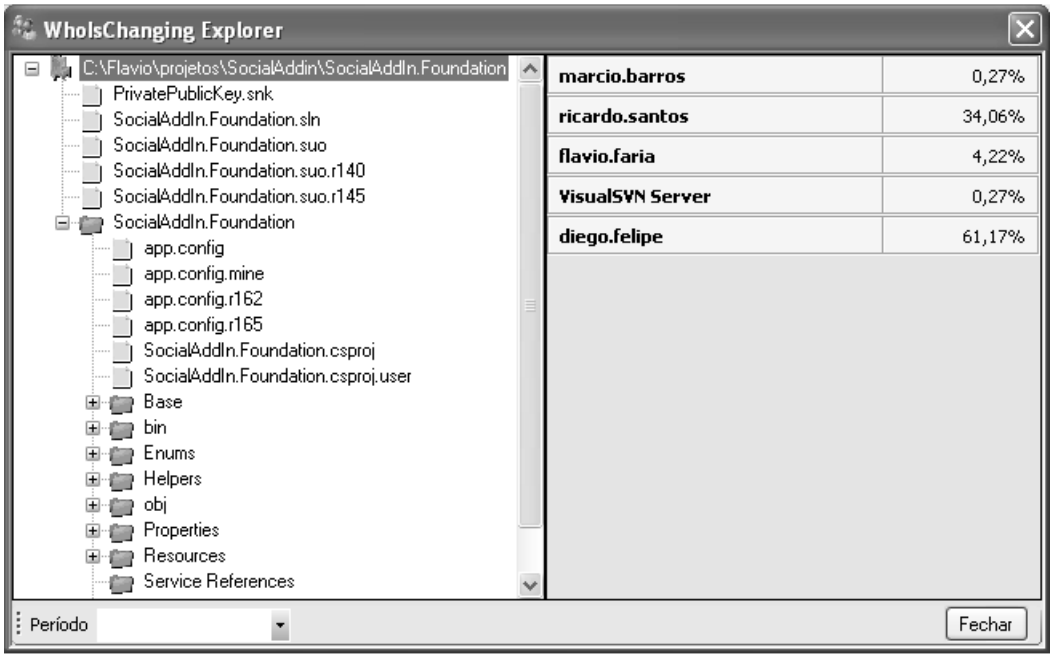

Figura 5.5 Lista de usuários que alteraram todo o projeto "SocialAddIn.Foundation"

5.3 LogExplorer e Communicator

Estes *add-ins* são responsáveis por exibir as mensagens que os desenvolvedores deixaram relacionadas ao arquivo selecionado no ambiente de desenvolvimento de software. O LogExplorer exibe as mensagens dos *commits* realizados no sistema de controle de versão, enquanto o Communicator exibe as mensagens criadas diretamente pelos desenvolvedores.

Ao dar foco em um arquivo no ambiente de desenvolvimento, uma lista com as mensagens referentes a esse arquivo é exibida nos *add-ins*. Para cada mensagem, é informado o usuário que a criou, a data de sua criação e o número de pontos de prestígio

que a mensagem recebeu. Além dessas informações, é possível atribuir pontos de prestígio para as mensagens com o objetivo de destacar a relevância do conteúdo.

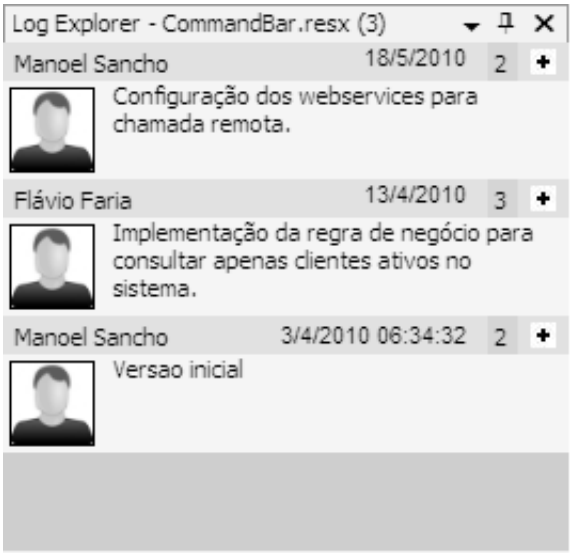

Figura 5.6 Janela do LogExplorer

A Figura 5.6 mostra o LogExplorer, exibindo as mensagens vinculadas a um *commit* do arquivo "*CommandBar.resx*". A Figura 5.7 exibe uma conversa, no Communicator, entre desenvolvedores a respeito de como configurar uma conexão que o arquivo "*TestLogReader.cs*" precisava para cumprir seus objetivos.

Figura 5.7 Janela do Communicator

5.4 Outros add-ins

A intenção inicial do projeto ora proposto era que, além dos *add-ins* já mencionados, existissem o *WhoIsReading* e um *add-in* de monitoramento e coleta de informações sobre o uso dos demais. Este último não começou a ser desenvolvido devido à desistência de alunos do curso de Mestrado onde os demais *add-ins* foram desenvolvidos, logo no início do curso. O *WhoIsReading* foi descontinuado na metade do processo de desenvolvimento. Por conta disto, não é possível exibir as telas referentes a estas aplicações.

5.5 Considerações Finais

Neste capítulo foi mostrado como os *add-ins* e o *website* propostos no capítulo anterior podem ser utilizados para facilitar a vida dos desenvolvedores de software e, desta maneira, tentar aumentar a qualidade do produto desenvolvido.

6 *Conclusão*

6.1 Contribuições

Neste trabalho foi criada uma rede social voltada para os desenvolvedores de um projeto de software. Essa rede é composta de um *website*, que possui uma função administrativa, um conjunto de *add-ins* para um ambiente de desenvolvimento de software, e um conjunto de *web services* através dos quais os dois componentes anteriores se comunicam. Através do uso dos *add-ins*, os desenvolvedores podem obter, no ambiente de desenvolvimento, informações necessárias para auxiliar a realização de suas tarefas, no contexto da construção de software, de maneira rápida e objetiva.

6.2 Trabalhos futuros

Visando o uso em um ambiente operacional, a primeira etapa é a criação de um instalador para o sistema. Esse instalador englobaria toda a lógica necessária das configurações e requisitos do sistema, diminuindo, assim, a complexidade existente para montar essa infraestrutura, além de reduzir o tempo e o esforço gasto nesse processo.

Novos *add-ins* podem ser criados. Além do *WhoChanged*, já descrito no capítulo IV, outros *add-ins* podem ser relevantes aos usuários. Um *add-in* que monitorasse o uso dos demais, a fim de averiguar o quão importantes e usados eles são, tornaria o sistema mais robusto.

Como cada um dos *add-ins* foi desenvolvido por uma equipe diferente, eles apresentam diferenças na interface com o usuário. A padronização da interface de todos os *add-ins* seria uma importante tarefa a ser realizada, pois além de criar uma identidade visual com o usuário, aceleraria a curva de aprendizagem em relação ao uso do sistema.

Outro ponto interessante a ser considerado é a extensão do *website*, com o objetivo de trazer para o navegador (*browser*) as informações disponibilizadas através dos *add-ins*. Essa extensão permitiria o uso do *website* não só pelos administradores, mas por todos os desenvolvedores. Cada usuário teria o seu perfil e cada projeto teria a sua comunidade. A comunidade de um projeto conteria todos os seus documentos e cada uma das informações dos *add-ins* referentes a ele. Por outro lado, o perfil do usuário possuiria somente as informações ligadas a ele.

6.3 Limitações do estudo

A implementação do projeto não foi testada em um ambiente de desenvolvimento real. Logo, não é possível afirmar os ganhos com a utilização da solução proposta em um ambiente real. Os projetos envolvidos também não passaram por testes mais rígidos e, portanto, existe a possibilidade deles conterem *bugs*.

Não foi possível configurar um servidor da Universidade para disponibilizar os serviços (*web services*) na Internet. Apenas uma rede local foi utilizada na comunicação entre os clientes e o servidor.

Diagramas UML como, por exemplo, diagramas de atividades, diagramas de sequência, casos de uso e de estado não foram criados para os projetos dos *add-ins*. A documentação destes componentes do sistema consiste de relatórios técnicos dos alunos do Mestrado, do modelo físico do banco de dados e do modelo de classes de negócio da aplicação.

Referências Bibliográficas

AMRIT C.; HILLEGERSBERG J.; KUMAR K. A social network perspective of Conway's law. In Proceedings of the CSCW Workshop on Social Networks, Chicago, IL, USA, 2004.

APACHE Foundation. Apache Web Server. Disponível em: http://www.apache.org Acessado em: Junho 2010.

ASP.NET. Disponível em: http://www.asp.net/mvc/whatisaspmvc Acessado em: Junho 2010.

BARROS M. "Notas de aula", 2010.

BERGERON B. Essentials of Knowledge Management, 2003.

BOSE R. "Knowledge management metrics". Industrial Management & Data Systems, vol. 104, no. 6. 2004.

CHOI B.; LEE H. "An Empirical Investigation of KM Styles and their Effect on Corporate Performance". Information & Management, vol. 40. 2003.

CONERY R.; GUTHRIE S.; HAACK P.; HANSELMAN S. Professional ASP.NET 1.0. Wiley, Indianapolis, IN, 2009.

COSTA R.A.; SILVA E.M.; NETO M.G.; DELGADO D.B.; RIBEIRO R.A.; MEIRA S.R.L. Social Knowledge Management in Practice: A Case Study. 2009.

CROSS R.L.; PARKER A. The Hidden Power of Social Networks: Understanding How Work Really Gets Done in Organizations. Harvard Business School Press: Boston, MA, 2004.

CVS Concurrent Version System. Disponível em: http://www.nongnu.org/cvs/ Acessado em: junho 2010.

DAVENPORT T.; PRUSAK L. "Conhecimento Empresarial: Como as Organizações Gerenciam seu Capital Intelectual". Rio de Janeiro: Campus, 256. 1998.

DOROGOVTSEV S.N.; MENDES J.F.F. Evolution of Networks: From Biological Nets to the Internet and WWW. Oxford University Press: Oxford, 2003.

ENDSLEY M.R.; GARLAND D.J. Situation Awareness Analysis and Measurement. Lawrence Erlbaum Associates, London. 2000.

FOSTER C.C.; RAPOPORT A.; ORWANT C.J. A study of a large sociogram: Elimination of free parameters. Behavioral Science. 1963.

GALLERY Design ASP.NET, Disponível em: http://www.asp.net/mvc/gallery/ Acessado em: Março 2010.

GOLDONI V.; OLIVEIRA M. Indicadores para o Processo de Gestão do Conhecimento: a Visão de Especialistas. 2006.

JQUERY. Disponível em: http://jquery.com/ Acessado em: Junho 2010.

KO A.J.; VENOLIA G.; DELINE R. Information Needs in Collocated Software Development Teams. 2007.

LATOZA T.D.; VENOLIA G.; DELINE R. Maintaining Mental Models: A Study of Developer Work Habits. 2006.

MCCONNELL S., Code Complete 2nd edition, Microsoft Press, 2004.

MICROSOFT LINQ. Disponível em: http://msdn.microsoft.com/en-us/netframework/aa904594.aspx Acessado em: Junho 2010.

MSSE Microsoft SQL Server 2008 Express Disponível em: http://www.microsoft.com/sqlserver/2008/pt/br/default.aspx Acessado em: Julho 2010.

MVC ASP.NET. Disponível em: http://www.asp.net/mvc Acessado em: Junho 2010.

NAIMZADA A.K.; STEFANI S.; TORRIERO A. “Networks, topology and dynamics: Theory and applications to economics and social systems” In: “Lecture Notes in Economics and Mathematical Systems”, 1ed. Springer. 2008.

NETCRAFT web server survey. Disponível em: http://survey.netcraft.com/archive.html Acessado em: Junho 2010.

PANOS M.; RUYTER de B.; MACKAY W. Awareness System Advances in Theory Methodology, and Design. pp. v. Springer, London. 2009.

POLANYI M. “The tacit dimension,” in: Knowledge in Organizations, Ed. London: Butterworths. 1967.

ROSSO C. D. Comprehend and analyze knowledge networks to improve software evolution. In: Journal of software maintenance and evolution: research and practice. 2009.

STAAB S.; DOMINGOS P.; MIKE P.; GOLBECK J.; DING L.; FININ T.; JOSHI A.; NOWAK A.; VALLACHER R. “Social Networks Applied”, IEEE Intelligent Systems, v. 20. 2005.

STOBER T., HANSMANN U.; Agile Software Development: Best Practices for Large Software Development Projects. Springer 2009

SUB Subversion Disponível em: http://subversion.tigris.org/ Acessado em: Junho 2010.

TROMBETTA A.; KOROGI G.; PERALTA K. Sistema de Controle de Versões. Faculdade de Informática – PUCRS, 2008.

UNIRIOTEC Bugzilla. Disponível em: http://outpost.uniriotec.br:8080/bugzilla/ Acessado em: Junho 2010.

VSS Visual SourceSafe. Disponível em: http://msdn.microsoft.com/pt-br/library/ms181038%28VS.80%29.aspx Acessado em: Junho 2010.

WALTER S. ASP.NET MVC Framework Unleashed. USA, 2010.

WASSERMAN S.; FAUST K. Social Network Analysis. Cambridge University Press: Cambridge, 1994.

WCF Windows Communication Foundation Disponível em: http://msdn.microsoft.com/en-us/netframework/aa663324.aspx Acessado em: Julho 2010.

WIRES, J.; FEELEY, M. Secure File System Versioning at the Block Level. ACM SIGOPS Operating Systems Review. Proceedings of the 2007 Conference on EuroSys EuroSys '07, Volume 41 Issue 3. 2007.

Anexo I – Serviços Disponibilizados

Neste anexo, são listados os principais métodos (serviços) que são disponibilizados para uso dos *add-ins*. A tabela a seguir exibe o nome do método e as seguintes características: tipo de retorno, parâmetros de entrada, Interface a qual o método faz parte, além de uma descrição do que ele se propõe a fazer.

UploadLog	**Interface**	ISourceControlService
	Tipo de retorno	Bool
	Parâmetros	SourceControlType sourceType string serializedLog
	Explicação	Insere no banco de dados as informações contidas no relatório do sistema de controle de versão. O "sourceType" informa de qual SCV estão vindo as informações e o "serializedLog" é o relatório serializado.
GetUserByLogin	**Interface**	*Todas
	Tipo de retorno	User
	Parâmetros	string usr_Login
	Explicação	Retorna o usuário correspondente ao login passado como parâmetro.
GetUserByID	**Interface**	*Todas
	Tipo de retorno	User
	Parâmetros	Guid usr_ID
	Explicação	Retorna o usuário correspondente ao ID passado como parâmetro.
GetUsers	**Interface**	*Todas
	Tipo de retorno	List< User>
	Parâmetros	---
	Explicação	Retorna uma lista contendo todos os usuários.
CreateNewUser	**Interface**	IWebSiteService
	Tipo de retorno	Bool
	Parâmetros	User newUser
	Explicação	Insere o usuário passado como parâmetro no banco de dados.

UpdateUser	**Interface**	IWebSiteService
	Tipo de retorno	Bool
	Parâmetros	User user
	Explicação	Atualiza o usuário passado como parâmetro no banco de dados.
GetUsersInterested	**Interface**	IWhoIsInterestedService
	Tipo de retorno	List< User>
	Parâmetros	ActionType interestType DateTime? startDate DateTime? endDate List<string> cfl_FullPaths bool showAllHistory
	Explicação	Retorna uma lista de usuários que submeteram alterações ou visualizaram (definido pelo parâmetro "interestType") os arquivos que tenham caminho igual a algum caminho contido no parâmetro "cfl_FullPaths" e que estejam no intervalo das datas "startDate", "endDate". O parâmetro "showAllHistory" indica se os usuários retornados pela operação conterão os objetos correspondentes às suas submissões / visualizações.
CreateWhoChanged	**Interface**	IWhoIsInterestedService
	Tipo de retorno	WhoChanged
	Parâmetros	DateTime wch_Date string wch_Message string wch_Review List<string> cfl_FullPaths string usr_Login
	Explicação	Insere no banco de dados um objeto WhoChanged com os atributos passados como parâmetro. Para cada string no parâmetro "cfl_FullPaths" é inserido um objeto LogCodeFile.
GetLogs	**Interface**	ILogExplorerService
	Tipo de retorno	List< WhoChanged>
	Parâmetros	List<Guid> cfl_IDs List<Guid> usr_IDs bool showAllHistory
	Explicação	Retorna uma lista de WhoChanged. A lista possui objetos WhoChanged que estejam relacionados a algum arquivo que possua um ID igual a algum ID contido no parâmetro "cfl_IDs" e que tenha sido criado por algum usuário que possua um ID igual a algum ID contido no parâmetro "usr_IDs". O parâmetro "showAllHistory" indica se os "WhoChanged" retornados pela operação conterão os objetos correspondentes à recepção de prestige points.
GivePrestigePointToLog	**Interface**	ILogExplorerService
	Tipo de retorno	bool
	Parâmetros	Guid wch_ID string usr_Login
	Explicação	Insere no banco de dados um objeto PrestigePointsLog com os parâmetros passados.

CreateLog	**Interface**	ILogExplorerService
	Tipo de retorno	WhoChanged
	Parâmetros	DateTime wch_Date string wch_Message string wch_Review List<string> cfl_FullPaths string usr_Login
	Explicação	Insere no banco de dados um objeto WhoChanged com os atributos passados como parâmetro. Para cada string no parâmetro "cfl_FullPaths" é inserido um objeto LogCodeFile.
GetMessages	**Interface**	ICommunicatorService
	Tipo de retorno	List<Communicator>
	Parâmetros	List<Guid> cfl_IDs List<Guid> usr_IDs bool showAllHistory
	Explicação	Retorna uma lista de Communicator. A lista possui objetos Communicator que estejam relacionados a algum arquivo que possua um ID igual a algum ID contido no parâmetro "cfl_IDs" e que tenha sido criado por algum usuário que possua um ID igual a algum ID contido no parâmetro "usr_IDs". O parâmetro "showAllHistory" indica se os "Communicator" retornados pela operação conterão os objetos correspondentes à recepção de prestige points.
GivePrestigePointToMessage	**Interface**	ICommunicatorService
	Tipo de retorno	Bool
	Parâmetros	Guid cmc_ID string usr_Login
	Explicação	Insere no banco de dados um objeto PrestigePointsCommunicator com os parâmetros passados.
CreateMessage	**Interface**	ICommunicatorService
	Tipo de retorno	Communicator
	Parâmetros	DateTime cmc_Date string cmc_Message List<string> cfl_FullPaths string usr_Login
	Explicação	Insere no banco de dados um objeto Communicator com os atributos passados como parâmetro. Para cada string no parâmetro "cfl_FullPaths" é inserido um objeto CommunicatorCodeFile.
RemoveMessage	**Interface**	ICommunicatorService
	Tipo de retorno	Bool
	Parâmetros	Guid cmc_ID Guid usr_ID
	Explicação	Atualiza o Communicator correspondente ao cmc_ID passado como parâmetro, alterando o valor do atributo "Enabled" para false, no banco de dados. A alteração só será executada caso o usr_ID passado como parâmetro seja o mesmo do Communicator contido no banco de dados.