FACULDADES IBMEC

PROGRAMA DE PÓS-GRADUAÇÃO E PESQUISA EM
ADMINISTRAÇÃO E ECONOMIA

DISSERTAÇÃO DE MESTRADO PROFISSIONALIZANTE EM ADMINISTRAÇÃO

Utilização da Arquitetura de Web Services no Desenvolvimento de Sistemas de Informação em Micro e Pequenas Empresas

JOSIR CARDOSO GOMES

Orientador: Prof. Dr. Valter Moreno Jr.

Rio de Janeiro, 22 de novembro de 2005

Utilização da Arquitetura de Web Services no Desenvolvimento de Sistemas de Informação em Micro e Pequenas Empresas

JOSIR CARDOSO GOMES

Dissertação submetida ao corpo docente da Coordenação das Faculdades IBMEC/RJ, como parte dos requisitos necessários à obtenção do grau de Mestre em Administração de Empresas.

Aprovada por:

Prof. _____ (Orientador)

Valter de Assis Moreno Jr., D.Sc. University of Michigan

Prof. _____

Simone Bacelar Leal Ferreira, D.Sc. PUC/RJ

Prof. _____

Sandra Mariano, D.Sc. COPPE/UFRJ

Rio de Janeiro, 5 de Dezembro de 2005

GOMES, JOSIR CARDOSO.

Utilização da Arquitetura de Web Services no Desenvolvimento de Sistemas de Informação em Micro e Pequenas Empresas. Rio de Janeiro: IBMEC / RJ, 2005.

158 pg.

Dissertação (Mestrado em Administração de Empresas) – Faculdades IBMEC/RJ, 2005.

Orientador: Prof. Valter de Assis Moreno Junior.

1. Web Services. 2. Administração de Empresas. 3. Engenharia de Software. 4. Software Livre

I. IBMEC/RJ. II. Título (série).

RESUMO

Acompanhar a profusão de novas tecnologias que surgem a todo momento é tarefa árdua para qualquer empresa, principalmente para as de pequeno porte. Em particular, para consultorias e organizações especializadas em desenvolvimento de software, é fundamental que estas estejam atualizadas nas novas tecnologias para que possam se manter competitivas. Este trabalho descreve o desenvolvimento de um sistema de informação (DSI) para pequenas e médias empresas baseado na arquitetura de Web Services (WS) e tem como objetivo avaliar os fatores críticos de sucesso no DSI baseados em WS executado por uma pequena *software house* e identificar potenciais vantagens e percalços que foram encontrados durante o projeto.

ABSTRACT

To be up to date on new technologies that emerge every day is a hard task to any organization, mainly to small ones. In particular to small software houses, it's crucial to research and study new software architectures and technics to maintain their competitiveness. This work describes the development of an Information System (IS) for small and medium enterprises (SME) using web services (WS) and a service oriented architecture (SOA). Its main goal is evaluate the critical success factors (CSF) on the software development based on WS executed by a small software house and identify potential advantages and pitfalls.

AGRADECIMENTOS

A Deus, a Jesus, ao Espirito Santo e em especial à Virgem Maria que me mostraram os caminhos para eu pudesse acreditar na vida e na humanidade.

Ao meu pai, Josir Simeone Gomes, que, com o seu exemplo de grande homem, pesquisador e humanista, me motivou desde criança a estudar. Às noites que ele passou “trancado no escritório”, privado da companhia dos filhos para se dedicar à sua pesquisa, que me serviram de estímulo para seguir em frente neste trabalho. Ao meu pai visionário, que foi o primeiro a me mostrar um computador, que teve a idéia de trocar um video-game por um micro-computador TK-80 e me abriu o caminho para a escolha da minha profissão.

À minha mãe querida, Maria Fernanda Cardoso Gomes, que, com a sua dedicação e abnegação, me fez entender que o sacrifício pela família e pelos entes queridos não só vale a pena como serve de exemplo para a nossa sociedade tão perdida e tão envolvida na ilusão do consumo. Agradeço também por cuidar dos meus “pimpolhos” com tanto carinho.

À minha amorosa esposa, Patrícia Gonçalves Saldanha, exemplo de perserverança, fé, honestidade e amor ao próximo, que me apoiou nestes 2 anos e meio de mestrado e tem sido minha melhor amiga nestes últimos 20 anos. Que fique aqui registrado para os nossos netos e bisnetos que a sua inteligência, gentileza e beleza me cativam a cada novo dia e foram força motriz para que eu concluisse esse trabalho.

Aos meus filhos muito amados, Bruna, Carolina e João Pedro, que tiveram MUITA paciência comigo, se privaram de muitos fins de semana e se comportaram além da conta.

Ao meu sogro e minha sogra, Raymundo e Maria, inseparáveis, que sempre foram mais do que pais para mim. Por cuidarem das crianças para que eu pudesse estudar e também para que eu pudesse me divertir – que ninguém é de ferro...

Aos meus avôs e avós que me deram muito orgulho de ser Cardoso e de ser Gomes.

À minha irmã Camila, apesar dela ser muito brigona, por compartilhar comigo o mesmo ideal de Paulo Freire de querermos ser bons educadores.

À minha segunda irmã Raquel Saldanha, que mesmo de longe, sempre esteve tão perto e por compartilhar seus sonhos conosco.

Ao frei Clemente Kesselmeier que me fez relembrar quais são os caminho realmente importantes e me deu prova viva de que ainda existem homens de fé dignos e bons.

Ao meu orientador, Prof. Valter Moreno Junior por aceitar e apoiar o tema da dissertação, pela paciência nas correções, pelas sugestões de leitura, pela disponibilidade e, principalmente, pela sua amizade.

A todos os professores do IBMEC que me deram a oportunidade de crescer em vários aspectos. Em especial ao Prof. Paulo Prochno que, em tão pouco tempo, me deu inúmeras boas idéias e me mostrou novas visões das organizações e ao Prof. Procópio Lima Neto que me deu um ânimo empreendedor e me fez ver inúmeras oportunidades no mundo empresarial.

A todo o pessoal da Media Systems que, com carinho, aguentou o meu mau humor e me ajudaram no dia a dia a suplantar os problemas do “Tio Bill”. Em especial ao Elias, à D.Eugênia e à Marion que me incentivaram e entenderam meus atrasos constantes.

A todos os desenvolvedores que participaram do projeto e foram fundamentais em

suas sugestões e contribuições para o sucesso desse trabalho. Em especial aos colegas de profissão, Demian Lessa e Marcelo Almeida que me auxiliaram no desenvolvimento do site e na configuração dos softwares em geral.

A toda a comunidade de software livre do Brasil que tem lutado para mostrar este novo caminho de solidariedade, respeito e partilha que a sociedade começa a trilhar. Em especial a equipe do site VivaoLinux e das listas de discussão xp-rio, java-br e delphi-br.

A todo o pessoal da secretaria do IBMEC, em especial ao Jeová - sua alegria sempre renovava os alunos quando chegavamos cansados à noite para assistir às aulas.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Siglas

API	*Application Programming Interface*
CMM	*Capability Maturity Model*
DSI	Desenvolvimento de Sistemas de Informação
EDI	Eletronic Data Interchange
EJB	*Enterprise Java Beans*
HTML	*Hyper Text Markup Language*
HTTPS	*Hyper Text Transfer Protocol over Secure socket layer*
LDAP	*Lightweight Directory Access Protocol*
MPE	Micro ou Pequena Empresa
SGBD	Sistema Gerenciador de Banco de Dados
SI	Sistema de Informação
SL	Software Livre
SOA	*Service Oriented Architecture*
SOAP	*Simple Object Access Protocol*
TI	Tecnologia da Informação
UDDI	*Universal Description Discovery and Integration*
UML	*Unified Modelling Language*
WS	*Web Services*
WS-Inspection	*Web Services Inspection Language*
WSDL	*Web Services Descripton Language*
WWW	*World Wide Web*
XML	*eXtensible Markup Language*

1. INTRODUÇÃO

A eficaz gestão da informação se tornou um dos pontos centrais, quase que decisivo, para o sucesso de uma organização. Empresas que conseguem disponibilizar informações relevantes, precisas, e em tempo hábil a seus funcionários, clientes e fornecedores, alavancam o seu poder de decisão, minimizam erros e, assim, têm maior potencial para satisfazer seus clientes. (O´Brien, 2003)

Para atender a demanda cada vez mais crescente em armazenar, recuperar e intercambiar informação, os pesquisadores e profissionais de Tecnologia da Informação (TI) têm procurado novos métodos e novas técnicas para desenvolver sistemas de informação (SI), tais como a arquitetura Cliente/Servidor e o paradigma da Orientação a Objetos.

Segundo Hagel (2004), as atuais arquiteturas de TI e os sistemas legados das empresas são um obstáculo a estratégias mais ágeis, em função das dificuldades que impõem à introdução de um novo produto ou serviço. Essas arquiteturas dificultam uma transição suave para novas soluções, pois são compostas por sistemas monolíticos e de difícil manutenção. Constantemente os administradores se deparam com decisões difíceis, tendo que optar entre dar continuidade a seus sistemas legados, mantendo a organização “engessada” e sem possibilidade de interagir com novas tecnologias, ou adquirir pacotes integrados, que geralmente envolvem custos elevados e têm um grande impacto nas diversas variáveis organizacionais (ex., estrutura, processos, competências, relacionamentos, clima, etc.).

(Hagel, 2004)

Para uma empresa, o ideal é que houvesse a possibilidade de se incorporar novas funcionalidades e implantar novos processos aos seus SI vigentes sem que fosse necessário alterar toda a sua estrutura de TI. Tal ideal deu origem a propostas de novas arquiteturas onde as empresas pudessem encontrar com facilidade os “pedaços” ou “partes” de software que fossem interessantes à sua operação.

Assim, seguindo o sucesso dos modelos orientados a objetos, as arquiteturas baseadas em componentes e, mais recentemente, baseadas em serviços (SOA – *Service Oriented Architecture*), o conceito de *Web Services* (WS) surgiu como aposta da indústria de software para a viabilização de novas estratégias de desenvolvimento e implantação de sistemas de informação que visam minimizar a necessidade de mudanças bruscas em seus clientes.(Clabby, 2002; Barry, 2003; Netcraft, 2004; VUNET, 2004).

Web Services (WS) é o resultado de um trabalho realizado pelas maiores empresas de software no mundo, com o objetivo de criar padrões tecnológicos abertos para implementar a arquitetura orientada a serviços, utilizando a infra-estrutura da Internet. Antes de ser apenas uma nova tecnologia, WS busca criar uma nova forma de comprar, vender e disponibilizar soluções de informática, permitindo que as empresas possam substituir incrementalmente seus sistemas de informação, diminuindo a dependência de seus fornecedores e tendo oportunidade de experimentar novas soluções sem a necessidade de abrir mão de seus sistemas existentes. Segundo Clabby (2002, p.2), “WS pretende afetar a forma como os sistemas de tecnologia de

informação são modelados, distribuídos e comprados".

Com a adoção dos padrões abertos e a promessa de investimento contínuo nessa solução feita pelas grandes empresas de software, os departamentos de TI começaram a partir de 2002 a experimentar e testar a arquitetura de WS (Datamonitor,2003). Numa pesquisa recentemente realizada entre as maiores empresas do mundo, 37% dos 273 entrevistados disseram já ter sistemas em produção utilizando WS, e 44% estão em processo de pesquisa e desenvolvimento (Westbridge, 2004).

Até a presente data, não há dados disponíveis publicamente sobre a utilização de WS no Brasil. Contudo, à medida que as empresas multinacionais começarem a requisitar soluções que interajam com os WS já implantados em seus países de origem, é de se esperar que as empresas brasileiras adotem a tecnologia em maior escala. Dessa forma, as empresas de software e consultorias fornecedoras de serviços de terceirização que buscarem investir nessa tecnologia poderão obter vantagens competitivas nesse novo cenário.

Supondo que a arquitetura orientada a serviços venha a se tornar preponderante a longo prazo, parece ser vital que as empresas busquem conhecer e analisar formas de adequar essa tecnologia aos seus sistemas atuais. Principalmente para micro e pequenas empresas (MPE), que são o foco deste trabalho, a adoção de WS pode trazer inúmeras oportunidades e ao mesmo tempo ameaças para a sua competitividade e sucesso no ambiente de negócios. Como foi dito anteriormente, WS não está restrito apenas ao aspecto tecnológico, pois envolve também a integração de processos de negócios numa empresa e entre esta e seus

parceiros com a finalidade de agregar valor ou reduzir custos. Dessa forma, é importante que se investigue os dois tipos de atores envolvidos nesse modelo de negócio e contexto específico: as MPEs que desenvolvem sistemas de informação, (*software-houses*), e as MPEs que utilizam produtos e serviços de TI.

Segundo os dados do SEBRAE, 84,79% das empresas instituídas formalmente no Brasil são consideradas como micro ou pequenas empresas. Ainda segundo este mesmo órgão, 64% das empresas exportadoras se enquadram no padrão de negócios de micro e pequeno porte, representando 7,6 bilhões de dólares em vendas para o mercado externo (Sebrae, 2004).

Desenvolver e estudar soluções de TI para este segmento pode trazer inúmeras vantagens para empresas especializadas em desenvolvimento de software e para aquelas que adotam seus produtos. Segundo pesquisa realizada pelo IDC Brasil em 2003, a previsão é que o crescimento em investimento de TI por parte das pequenas e médias empresas seja de quase 6% (ComputerWorld, 2003).

Sob o ponto de vista das empresas que desenvolvem sistemas de informação (SI), os WS podem abrir novos mercados e criar novas oportunidade de negócios, que antes não eram possíveis, em função das estruturas utilizadas, baseadas em vendas de pacotes fechados que dependiam de um esforço extra de uma equipe de vendedores ou de pontos de venda em lojas especializadas.

Especialmente para MPEs, que têm recursos limitados e pouca escala, há um

mercado enorme que pode ser explorado. Com a arquitetura de WS, espera-se que seja possível desenvolver soluções e serviços que antes só se encaixavam em modelos de negócios maiores que necessitavam de uma grande infra-estrutura tecnológica e humana que sempre esteve fora do alcance das pequenas empresas. Um exemplo rápido deste argumento é o padrão EDI (Eletronic Data Interchange) no qual apenas grandes empresas tinham condições de prover serviços de troca de informações através de EDI, devido à complexidade dos protocolos e métodos que tinham que ser utilizados. (Morris, 2000; NIST, 2005)

Já do ponto de vista de quem compra soluções de TI, tem-se dado grande atenção ultimamente ao conceito de alinhamento entre soluções tecnológicas e a estratégia da empresa (Brodbeck, 2002; Laurindo, 2001). Como em todo nova tecnologia, os empreendedores devem estar atentos aos problemas que podem surgir em uma adoção equivocada de TI. Embora a simples compra de pacotes e soluções integradas de valor elevado não seja sinônimo de um bom retorno financeiro, ao não se preparar tecnologicamente para crescer ou para gerir melhor suas operações, através do uso de sistemas de informação, uma empresa pode estar minando sua competitividade e comprometendo sua sobrevivência no mercado.

Um projeto de TI alinhado com a estratégia da empresa pode trazer inúmeras vantagens competitivas e potencializar sua força produtiva (Brodbeck, 2002). Especificamente em relação às MPEs, é importante que estas estejam preparadas para crescer e, ao mesmo tempo, sobreviver em um ambiente cada vez mais competitivo.

Alguns autores que examinaram os fatores críticos de sucesso para uma MPE,

criaram modelos que tentam demonstrar quais os passos que estas empresas geralmente seguem e quais as barreiras que foram superadas durante o seu crescimento. Quinn e Cameron (1983 apud Daft, 2002), por exemplo, colocam que podem ser observados quatro estágios no ciclo de vida de uma empresa, até atingir a maturidade. Segundo o autor, antes de cada novo estágio, uma crise surge e faz com que a empresa tenha que se adaptar para superá-la. No primeiro estágio, chamado de empreendedor, a empresa está criando seus produtos e mercado. Já o estágio de Coletividade é alcançado quando a empresa cresce e o maior número de funcionários faz com que a empresa tenha problemas de gerenciamento. Nesta fase, a empresa procuram se ajustar montando equipes e descobrir pessoas que possam liderar estas equipes. No estágio de formalização, a empresa busca a criação de normas, procedimentos e sistemas de controle como forma de minimizar conflitos gerenciais e coordenar melhor suas áreas.

Figura 1: Ciclo de Vida das Empresas (Adaptado de Daft,2002 pg.264)

Especificamente no estágio de Formalização, o autor descreve que existe uma crise da delegação e controle e que as empresas precisam estar com seus sistemas internos aptos para a mudança, ou seja, sua estrutura deve estar preparada para se adequar a pressões exercidas pelos ambientes externo e interno. Assim, ao investir em sistemas de informação que sejam adaptáveis e de fácil manutenção, as empresas podem estar dando um passo acertado para se prevenir e resolver com antecedência os conflitos e crises que surgirão durante o seu crescimento. A adaptabilidade é justamente um dos pontos centrais que a tecnologia de WS pretende melhorar. Sua implementação permite que as empresas adaptem seus sistemas de informação de forma gradual, sem perda de qualidade e com baixo custo de forma segura e sustentável (Barry, 2003).

1.1. Objetivo

Avaliar novos métodos para desenvolver sistemas de informação que se sejam mais adaptáveis ao ambiente e que minimizem o custo de desenvolvimento é questão central para empresas que desenvolvem software. Este estudo tem como objetivo avaliar os fatores críticos de sucesso no desenvolvimento de sistemas de informação baseados em WS executado por uma pequena *software house* e identificar potenciais vantagens e percalços que possam ser encontrados durante o projeto.

Como objetivo secundário, tem-se a avaliação dos benefícios e problemas associados ao uso de ferramentas de desenvolvimento de SI baseadas em software livre como forma de

diminuir os custos do processo de construção de software.

1.2. Relevância

Este estudo pretende atingir três públicos distintos: o de tomadores de decisão relativas à tecnologia da informação nas empresas, tais como CIO (Chief Information Officer), CTO (Chief Technology Officer) e gerentes de TI de empresas cujo negócio principal não seja TI; os micro e pequenos empresários do setor de desenvolvimento de sistemas de informação (DSI) e por fim pesquisadores de sistemas de informação que estejam buscando casos reais para avaliar a tecnologia de WS.

Para o primeiro público, o trabalho pode trazer subsídios para que se avalie como a arquitetura baseada em Web Services pode trazer benefícios para a operação de suas empresas. Já para o segundo grupo, dos desenvolvedores, esta pesquisa se propõe a mostrar novos modelos de negócios que podem advir desta arquitetura para PMEs e servir como uma base de reflexão para seus processos atuais de desenvolvimento.

Da mesma forma, avaliar a tecnologia de WS e as arquiteturas orientadas a serviços pode trazer inúmeros benefícios para a pesquisa acadêmica, já que, atualmente, a indústria de TI está investindo em grande escala nessa arquitetura e pouco se conhece sobre seus impactos nas variáveis organizacionais.

1.3. Delimitação do Estudo

O presente trabalho visa estudar exclusivamente o universo das MPEs brasileiras, especificamente do ramo de serviços. A definição do que é uma Micro ou Pequena empresa tomará como base o critério do SEBRAE[1], que, por sua vez, utiliza os dados do IBGE (Sebrae,2004).

Como será visto no terceiro capítulo, devido à natureza do método de pesquisa empregado, não se pretende avaliar a tecnologia de WS em si ou tampouco sugerir que as soluções adotadas neste estudo sejam as melhores a serem adotadas em qualquer situação.

1.4. Plano da Dissertação

O trabalho será estruturado da seguinte forma: após esta introdução, será feita uma revisão da literatura, que aborda inicialmente a história e o avanço das arquiteturas de desenvolvimento de SI, para em seguida apresentar o funcionamento da tecnologia de WS dentro deste contexto. Em seguida, descreve-se o universo das MPE especializadas em DSI, o processo de DSI em si e, por fim, as estratégias e oportunidades que podem surgir da adoção das arquiteturas baseadas em WS.

No terceiro capítulo, descreve-se a metodologia de pesquisa adotada e o porquê da escolha dessa metodologia.

O quarto capítulo apresenta os resultados obtidos através de uma pesquisa-ação sobre

1 A definição está descrita na seção 2.1

a utilização da arquitetura de WS por uma pequena empresa especializada em desenvolvimento de software com o intuito de resolver um problema de disponibilização de informações financeiras e fiscais entre os sócios e os escritórios de contabilidade contratados para realizar o registro contábil e fiscal da empresa.

No último capítulo, serão apresentadas as conclusões sobre a eficácia da arquitetura de WS para a solução do referido problema e os benefícios e desvantagens encontrados. Além disso, serão dadas sugestões de novos estudos que poderão ser iniciados a partir desta pesquisa.

2. REFERENCIAL TEÓRICO

"Não é a mais forte das espécies que sobrevive, nem a mais inteligente e sim a que estiver mais preparada para a mudança".
Charles Darwin, Biólogo e Filósofo.

Este capítulo apresenta os principais conceitos abordados na realização nessa pesquisa. Inicialmente, será apresentado um histórico das atuais arquiteturas de desenvolvimento de sistemas de informação (DSI), até o surgimento da arquitetura baseada em Web Services (WS), e as características e vantagens dessa tecnologia para as empresas, sob o ponto de vista técnico e estratégico.

Em seguida, será feita uma análise da realidade das micro e pequenas empresas (MPEs)de serviços no Brasil, e, em particular, as especializadas em desenvolvimento de sistemas de infromação (DSI). Na seção seguinte, o processo de DSI será detalhado, destacando-se os principais fatores que o influenciam e quais as estratégias empresariais que podem ser adotadas para este tipo de empreendimento. Finalmente, serão listadas as formas como a tecnologia de WS podem beneficiar estrategicamente uma MPE desenvolvedora de software e como a utilização de software livre pode influenciar o contexto organizacional dessas empresas.

2.1. Sistemas de Informação e a evolução das arquiteturas de software

Stair (1998) define que um sistema de informação (SI) é um conjunto integrado de recursos (humanos e tecnológicos), cujo objetivo é satisfazer adequadamente a totalidade das necessidades de informação de uma organização e os respectivos processos de negócio. Definição similar é feita por O'Brien que diz que um SI é um conjunto organizado de pessoas, hardware e software[2] que coleta, transforma e dissemina informações em uma organização ou em uma comunidade (O´Brien, 2003). Já Steenis (1990) considera que um SI é composto por três elementos: hardware, software e "*otherware*". Ele considera "*otherware*" todos os elementos que fazem parte do sistema, tais como a organização possuidora do SI, as pessoas que o utilizam e o gerenciam, os procedimentos operacionais, os objetos físicos, e até mesmo os objetivos do sistema.

Especificamente, o software é composto "pelas instruções de processamento que são programadas em um computador e as mídias de dados, ou seja, os objetos tangíveis nos quais são registrados os dados" (O´Brien, 2003, pg. 22). Por ser uma área central na construção de SI, os pesquisadores de ciência da computação tem buscado definir as melhores práticas de desenvolvimento de software. Devido ao aumento constante da complexidade dos SI, os analistas e engenheiros de sistemas procuraram desenvolver formas de estruturar e documentar a interligação entre os diversos componentes de software. Segundo Bass (2003), a esta estruturação dá-se o nome de arquitetura:

2 As palavra hardware e software não aparecem em itálico denotando palavra estrangeira pois estas já foram incorporadas à grande parte dos dicionários de língua portuguesa.

> "The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationship among them." (Bass, 2003, pg.21)

Uma arquitetura de software se preocupa com os aspectos mais gerais dos diferentes tipos de componentes de software utilizados numa organização, e com suas formas de interação (Garlan et al., 1997). Segundo Silveira (2000), a arquitetura para desenvolvimento de software deve cumprir dois papéis:

1. fornecer um nível de abstração no qual os projetistas podem argumentar sobre o comportamento do sistema: funcionalidade, desempenho, confiabilidade, e etc; e
2. fornecer uma "consciência" para a evolução do sistema, indicando quais aspectos do sistema podem ser facilmente alterados sem comprometer a sua integridade.

Sob o ponto de vista da Administração de Empresas, a mudança na arquitetura de sistemas de informação pode ser vista como fruto da necessidade de adaptar os SI aos desafios e mudanças que surgiram e que não foram resolvidos de forma satisfatória por uma arquitetura anterior. Esta questão pode ser visualizada na figura 2, que será explicada em uma breve revisão da evolução das arquiteturas de SI.

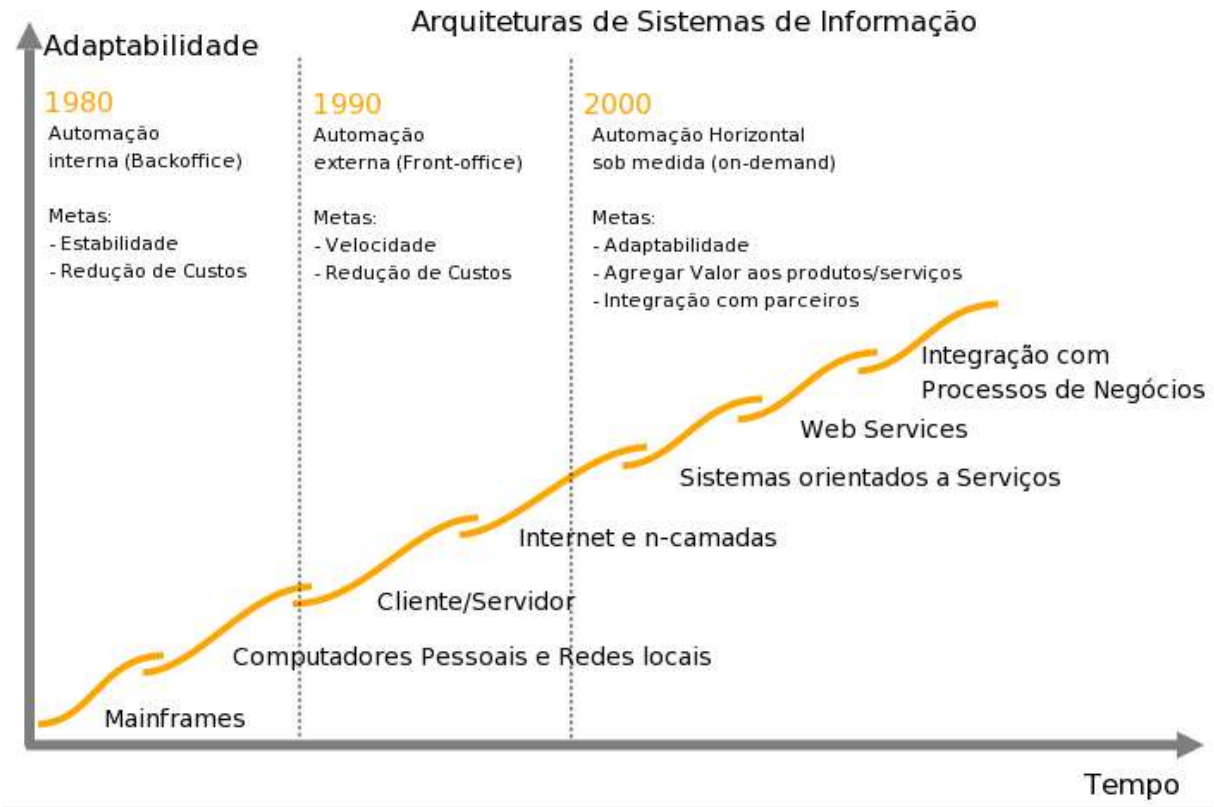


Figura 2: Evolução das arquiteturas de sistemas de informação (Adaptado de HP, 2004)

É importante frisar que o surgimento de uma arquitetura não fez com que outra arquitetura deixasse de ser usada imediatamente. O declínio do uso de uma arquitetura é lento, pois as empresas não estão dispostas ou não têm recursos para mudar seus sistemas a cada 5 anos (tempo médio entre o surgimento de uma nova arquitetura), e geralmente a transição entre duas arquiteturas não é simples. A prova disso é que todas as arquiteturas aqui apresentadas são utilizadas até hoje de uma forma ou de outra por empresas de todos os portes (O'Brien, 2003).

A afirmação de O'Brien pode ser constatada em recente pesquisa realizada pelo IDC (2005) que mostra que as todas as arquiteturas acima ainda são utilizadas atualmente pelas empresas pesquisadas.

2.1.1. Arquiteturas baseada em *Mainframes* e Cliente/Servidor

Os sistemas baseados em *mainframes*, adotados a partir do final dos anos 60, constituiram a primeira arquitetura utilizada largamente pelas empresas. Segundo Sayles (2004), o objetivo primário do uso de computadores naquele momento era reduzir o custo de mão de obra e aumentar a escala de serviços ou produção, através da automatização de rotinas repetitivas e armazenamento de grande volume de dados. Toda a arquitetura era baseada em um computador central, que provia informações para terminais que não tinham qualquer poder de processamento, limitando significativamente a manipulação de dados pelos usuários. O maior problema desta tecnologia era justamente a concentração das decisões e implementações relativas à manipulação dos dados nos chamados CPDs (centros de processamento de dados). Os CPDs raramente conseguiam fornecer as informações requeridas pelos usuários com a rapidez necessária, pois estavam quase sempre sobrecarregados (já que todo processamento estava centralizado em suas mãos) ou simplesmente isolados do restante da empresa e sem uma compreensão real dos aspectos essenciais do negócio. Para completar o quadro, o custo e o tempo necessários para a preparação dos programas, processamento, e extração de dados era muito alto (O'Brien, 2003; Sayles, 2004).

Com o advento dos microcomputadores ou PCs (*Personal Computers*) e das redes locais, os diversos setores de uma empresa perceberam que podiam armazenar informações

relevantes nestes computadores de menor porte e, portanto, não depender mais dos *mainframes*, pelo menos para pequenos volumes de dados. A febre das planilhas eletrônicas e dos processadores de texto, que se iniciou com o Visicalc e Lotus 1-2-3, e se consolidou com o Microsoft Excel trouxe um poder de processamento e uma sensação de ganho de produtividade enormes, principalmente para as pequenas e médias empresas que não dispunham de recursos para comprar ou desenvolver sistemas de informação em computadores de grande porte. Seguindo esta tendência, surgiram os primeiros bancos de dados para PCs, tais como o Dbase, Borland Paradox e Microsoft Access (O'Brien, 2003).

Entretanto, os PCs, como forma de armazenar os dados corporativos das empresas, apresentavam diversas desvantagens. Os gerentes perceberam que muitas vezes a mesma informação era armazenada várias vezes, em departamentos diferentes, gerando inconsistências e um retrabalho desnecessário. Da mesma forma, como os dados estavam descentralizados e em formatos diferentes, era difícil gerar consultas consolidadas. Por fim, devido à perda de dados e limitações dos programas de escritório e do próprio PC, ficou claro para as empresas que armazenar grandes volumes de dados em planilhas e bancos de dados locais era impraticável (O'Brien, 2003).

Neste ínterim, ainda nos anos 80, a arquitetura cliente-servidor, baseada nos bancos de dados relacionais, estava sendo implantada em grandes empresas. Com isso, a capacidade de manipulação da informação foi aumentada sobremaneira, principalmente através da criação de linguagens de recuperação de dados, tal como SQL (*Structured Query Language*) (Date,

1991). Esta segunda arquitetura era baseada em computadores de maior porte que os PCs (os chamados **servidores**), onde sistemas gerenciadores de banco de dados (SGBD) armazenavam e gerenciavam os dados armazenados, atendendo a solicitações encaminhadas pelos programas que rodavam em computadores de menor porte (os chamados **clientes**), que geralmente eram os mesmos PCs que executavam os pacotes de escritórios. Com a arquitetura cliente/servidor, foi possível criar, por exemplo, softwares complexos, como os ERP (*Enterprise Resource Planing*), destinados à integração de diferentes sistemas de informação corporativos (O´Brien, 2003).

Entretanto, o custo para implementar mudanças de rotina ou criar novas funcionalidades nos sistemas que utilizam essa arquitetura continua sendo muito grande. Segundo Brown (2004, p.2), a forma rígida pela qual os programas cliente/servidor são implementados, bem como a dificuldade enfrentada quando se tenta modificar o software, geram limitações significativas para a gestão corporativa. As despesas e dificuldades podem ser tão grandes, que algumas empresas preferem abandonar seus projetos de SI a tentar alterar os programas.

Um outro fator que limita essas duas arquiteturas é que elas foram desenhadas para serem executadas nas redes internas das empresas, tendo apresentado problemas para estender as soluções criadas com base nessa arquitetura com o ambiente externo a fim de interconectar a empresa e seus parceiros, clientes, fornecedores, etc. Dessa forma, na metade da década de

noventa, os cientistas de computação perceberam que era necessário desenvolver novas arquiteturas que pudessem melhorar a comunicação entre empresas.

2.1.2. A Internet e a Arquitetura em n-Camadas

Com o advento da *World Wide Web* (WWW ou Web), no início dos anos 90, as organizações encontraram nos protocolos abertos uma nova forma de disponibilizar seus sistemas e serviços. Várias empresas perceberam que a Internet podia se tornar um canal de intercâmbio de informações com seus clientes e fornecedores, incrementando consideravelmente o comércio eletrônico, que, até então, era realizado em plataformas caras e proprietárias, tal como o EDI. Entende-se, aqui, o comércio eletrônico como sendo "a realização de toda a cadeia de valor dos processos de negócio num ambiente eletrônico, por meio da aplicação intensa das tecnologias de comunicação e de informação, atendendo aos objetivos de negócio" (Albertin, 1999, p. 15).

Ao mesmo tempo, os desenvolvedores de software perceberam o imenso potencial dos navegadores ou *browsers*, tais como Netscape, Internet Explorer, Opera e Mozilla. Estes navegadores permitiam que desenvolvessem um novo tipo de sistema, onde qualquer computador, em qualquer parte do mundo, com qualquer sistema operacional que suportasse os padrões da Web, podia acessar os aplicativos corporativos que, antes, tinham que ser instalados em cada estação de trabalho, e adapatados para cada tipo de sistema operacional. Essa nova arquitetura era capaz de reduzir o custo total de manutenção dos sistemas cliente-

servidor, além de aumentar significativamente a flexibilidade do processamento e acesso a informações. Com isso, possibilitava um ajuste mais rápido das empresas às novas necessidades de negócio que porventura surgissem (O´Brien, 2003 pg.81).

As primeiras aplicações para Web utilizavam um modelo similar ao cliente-servidor original. No entanto, o cliente era, agora, o próprio navegador da Web. Uma aplicação, sendo executada num servidor da Internet, se comunicava com o banco de dados, que podia estar ou não no mesmo computador. A aplicação enviava as solicitações do cliente para o banco de dados, e passava os dados que eram recuperados do SGBD (Sistemas Gerenciadores de Banco de Dados), no formato padrão HTML, para o navegador. Nesse tipo de arquitetura, havia agora 3 camadas: o navegador, que implementava a interface com o usuário; o programa que acessava ou solicitava a gravação dos dados no SGBD; e o próprio SGBD. Assim, buscava-se separar o acesso ao banco de dados e o processamento das regras de negócio embutidas no sistema da interface visual (também chamada *front-end*). Os ganhos de escalabilidade, flexibilidade e a otimização de recursos computacionais obtidos com essa arquitetura motivaram os desenvolvedores a ir além das três camadas. A idéia era que houvesse várias camadas de software, cada qual responsável e especializada por uma função no SI: uma camada iria montar a página a ser enviada ao navegador, outra iria se preocupar exclusivamente com algumas das regras de negócio, e assim por diante (O´Brien, 2003).

Ainda dentro desse modelo de várias camadas, a arquitetura baseada em componentes tem se destacado, sendo bem similar à arquitetura orientada a serviços que será vista a seguir.

Basicamente, a arquitetura baseada em componentes procura subdividir e organizar as aplicações em componentes, que são pequenos programas executáveis que têm uma interface pública que pode ser acessada por outros componentes. Um dos objetivos desta arquitetura é tentar aproximar os componentes do SI aos modelos lógicos orientados a objetos (Vitharana,2003).

Na maioria das vezes, estes componentes são agrupados e distribuidos sob forma de **bibliotecas de componentes** que visam solucionar problemas específicos, como por exemplo, acessar um determinado SGBD, realizar cálculos complexos de engenharia, acessar os dados de um *mainframe*, conexão com dispositivos eletrônicos (celulares, catracas de portaria, etc) ou acessar computadores na Internet utilizando protocolos abertos ou proprietários (Schupp, 2004).

Entretanto, por utilizarem tecnologia proprietária, os aplicativos baseados em componentes necessitarem de um investimento inicial muito elevado além de não terem portabilidade, ou seja não é simples fazer com que componentes escritos em plataformas diferentes se comuniquem entre si de forma imediata (Vitharana,2003; Schupp, 2004). Exemplos de arquiteturas que utilizam este tipo de arquitetura são o COM+, desenvolvido pela Microsoft, e o EJB (Enterprise JavaBeans), desenvolvido pela Sun Microsystems e utilizado em implementações escritas na linguagem Java (Stal, 2002).

2.1.3. Arquiteturas Orientadas a Serviços (SOA)

Leyman (2002) define SOA (*Service Oriented Architecture*), ou arquitetura orientada a serviços, como um novo tipo de arquitetura onde aplicativos e rotinas são disponibilizados como serviços numa rede de computadores (ex., Intranets, Extranets, Internet), podendo assim ser utilizados por diferentes aplicações e para vários propósitos. Com esse tipo de arquitetura, o desenvolvimento de novas aplicações se resume em selecionar os serviços disponíveis na rede e combiná-los numa determinada sequência de execução, de acordo com as regras de negócio a serem atendidas.

Um dos principais fatores motivadores da SOA e das arquiteturas baseada em componentes está na busca do conceito de modularidade. Esse conceito é, na realidade, mais geral, indo além da questão específica do desenvolvimento de sistemas. A modularidade é uma estratégia para organizar sistemas complexos. Um sistema modular é composto de unidades (ou módulos) que são projetados independentemente, mas que interajem e se comportam como partes de um sistema maior (Baldwin e Clark, 1997). Schilling (2000, apud Iyer et al., 2003) define modularidade como um construto que descreve graus pelos quais os componentes de um sistema podem estar separados e serem recombinados. A modularidade se refere tanto a sistemas cujos componentes estejam fortemente acoplados, quanto ao grau em que as “regras” da arquitetura de um sistema permite (ou proíbe) a combinação destes componentes (Iyer et al, 2003).

As razões por trás do desenvolvimento de tal arquitetura estão relacionadas às dificuldades que geralmente se enfrenta para manter a estratégia da empresa, seus processos de negócio e a infraestrutura de TI que os suporta sempre alinhados, mesmo quando o ambiente competitivo da organização exige mudanças e correções de rumo freqüentes. A arquitetura SOA visa permitir que novas regras de negócio sejam implementadas rapidamente nos sistemas corporativos, sem que seja necessário realizar intervenções profundas e custosas nos programas e aplicativos. A implementação de novas regras seria realizada simplemente através da eliminação dos serviços que deixariam de ser utilizados, da inclusão dos serviços exigidos pelas novas regras de negócio, ou da reordenação da sequência de execução dos serviços no sistema.

Segundo Stal, SOA não é uma idéia nova. Esta arquitetura tem sido estudada desde a metade da década de 80. Desde então, houve várias tentativas de implementar sistemas baseados em SOA: a IBM, por exemplo, desenvolveu o padrão DCE (*Distributed Computing Environment*); um consórcio de grandes empresas de TI desenvolveu o padrão CORBA (*Common Object Request Broker Architecture*); e a Microsoft criou o padrão DCOM. Entretanto nenhuma dessas implementações realmente foi à frente devido a três fatores principais: (1) não havia software para integração de aplicativos (*middleware*) que fosse padronizado e aberto (não-proprietários); (2) não existiam definições de interfaces padronizadas para permitir a conexão entre os módulos; e (3) não havia compatibilidade e integração entre os produtos dos diferentes fornecedores (Stal, 2002).

2.1.4. Web Services

Em setembro de 2000, foi criado um grupo de trabalho no World Wide Web Consortium (W3C) com o objetivo de desenvolver uma arquitetura onde diversos protocolos permitissem a interoperabilidade entre aplicações e sistemas, de plataformas, ambientes e arquiteturas diferentes. Esse grupo de trabalho, formado por representantes das maiores empresas de software do mundo, tais como Microsoft, IBM, Oracle e Sun Microsystems, definiu, assim, uma nova arquitetura computacional chamada de Web Services, com condições de melhorar o suporte e aprimorar e agilizar a interação entre processos de negócio, e, por conseguinte, entre empresas (W3C, 2003).

A definição formal adotada pelo W3C diz que WS é um sistema de programas desenhados para suportar a interação máquina a máquina, através de uma rede, utilizando protocolos padronizados (W3C, 2003). Outros sistemas interagem com um WS utilizando mensagens SOAP (uma forma de XML), normalmente transmitidas utilizando o protocolo HTTP em conjunto com outros padrões utilizados na Web (W3C, 2004). Esta definição não é muito clara pois pressupõe que o leitor já tenha conhecimento prévio das definições de SOAP, HTTP e XML. Assim, as nomenclaturas acima serão detalhadas na próxima seção.

Buscando uma definição mais gerencial, pode-se dizer que a arquitetura de WS é baseada no conceito de distribuição e modularidade, adotando protocolos abertos e padronizados com o intuito de promover a integração de aplicações com baixo acoplamento

(*loosely coupled applications*) (Sleeper, 2001; Iyer et al., 2003). O objetivo é habilitar essas aplicações ou componentes a disponibilizar suas funções e trocar dados, utilizando a infraestrutura padrão da Internet (Huang, 2004).

Aqui é importante definir o que é acoplamento já que este conceito está no cerne das vantagens dos WS. Segundo Coad (1993), acoplamento é o nível de inter-dependência entre os módulos ou componentes de um programa ou sistema. Sob o ponto de vista prático, quanto maior for o acoplamento, maior será a dificuldade em tornar os componentes de um sistema independentes. De maneira inversa, caso seja necessário alterar um módulo ou componente que tenha baixo acoplamento, menor será o impacto da mudança no sistema em que ele está inserido.

Sob o ponto de vista de quem vai utilizar os WS (o consumidor dos WS), o modelo funciona da seguinte forma:

(1) A empresa que necessita de um serviço específico busca alguém que preste esse serviço através de um diretório de WS (uma espécie de páginas amarelas de serviços) implementada com o protocolo UDDI;

(2) Achado o serviço necessário, o cliente consulta o serviço para obter uma descrição detalhada para que possa obter os requisitos e procedimentos relativos ao funcionamento do WS. Uma vez de posse desta descrição, o cliente contrata o serviço com o provedor do WS que então lhe dá acesso ao serviço desejado.

Inclusive, essa operação pode ser feita de forma automatizada, ou seja, sem que hava a intervenção humana durante a contratação.

(3)Uma vez que o serviço tenha sido contratado, o cliente pode então acessar e interagir com os serviços do WS. Este processo está representado na figura abaixo.

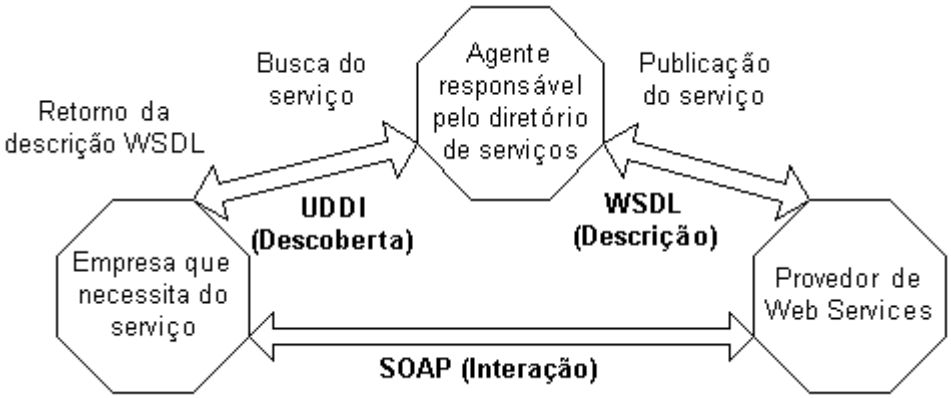


Figura 3: Visão macro da arquitetura de WS (Adaptado de Iyer et al, 2003).

Já sob o ponto de vista do provedor, ou seja de quem vai prover os WS, a mecânica funciona da seguinte forma: (1) o desenvolvedor possui um serviço implementado em software que ele quer disponibilizar ou vender; (2) ele formata esse serviço sob forma de um Web Service, e o publica em um diretório público, para que seus prováveis clientes saibam que o serviço está disponível; (3) para cada cliente que se interesse pelo serviço, a empresa que desenvolveu o WS disponibiliza o acesso e acompanha se o serviço está sendo utilizado corretamente e com a qualidade esperada (confiabilidade, velocidade, etc).

A base de todos o modelo de padrões utilizados na implementação de WS é o XML ou *eXtensible Markup Language*. Segundo Clabby, XML é uma recomendação que descreve

como os dados trocados entre aplicativos devem ser estruturados. Na prática, é uma forma de descrever e compartilhar dados, usando um formato comum de apresentação (Clabby, 2002). Por ser definido sob forma de texto, com marcações bem definidas, e em função de sua natureza hierárquica, XML é facilmente entendido tanto por um ser humano, quanto por um programa de computador que precise interpretá-lo. Um exemplo da notação do XML que descreve o empréstimo de dois livros de uma biblioteca pode ser visto abaixo:

```
<emprestimo>
  <livro id="1231313">
    <nome>Brás Cubas</nome>
    <autor>Machado de Assis</autor>
    <editora>Editora Tres</editora>
    <ano>1997</ano>
  </livro>
  <livro id="2342322">
    <nome>A origem das espécies</nome>
    <autor>Charles Darwin</autor>
    <editora>Editora Moderna</editora>
    <ano>1995</ano>
  </livro>
</emprestimo>
```

Segundo Kreger (2003), a padronização definida para WS pode ser dividida em três camadas cujos nomes coincidem com seus papéis conceituais: Interação, Descrição e Descoberta. Para cada uma destas camadas, as regras e funcionalidades de outras três sub-camadas devem ser respeitadas, para que a funcionalidade do modelo completo possa ser alcançada. O modelo e suas camadas pode ser visto na figura a seguir.

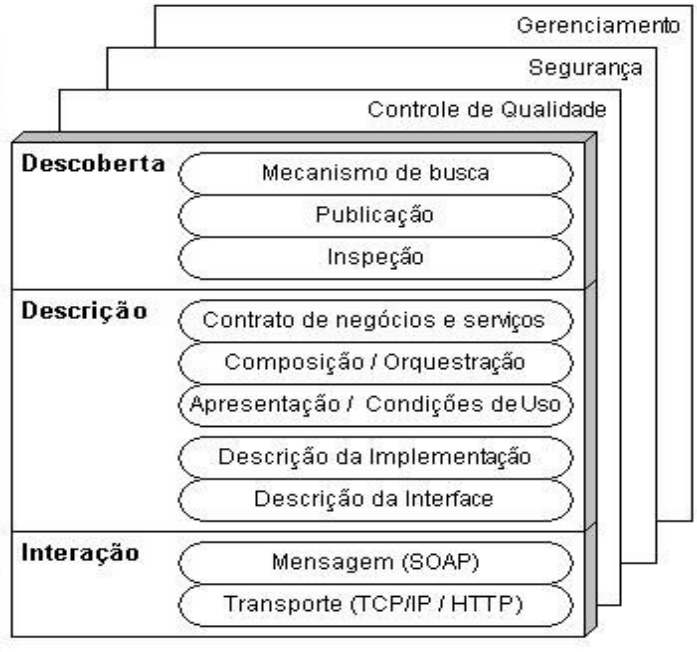


Figura 4: Camadas conceituais que compõe a arquitetura de WS (adaptado de Kreger, 2003).

Kreger (2003) relaciona a primeira camada de interação com o processamento físico onde se dá efetivamente a execução das ações desejadas, e onde se definem os protocolos de comunicação padronizados, tais como HTTP e SOAP. Esses protocolos são utilizados para que os dados recebidos e enviados possam ser transportados na forma mais transparente possível pela infraestrutura da Internet e redes das corporações (intranets e extranets), sem a necessidade de softwares intermediários (*middleware*) ou de qualquer outro artifício técnico. Vale ressaltar que, embora SOAP seja o protocolo mais utilizado na implementação de Web Services, existem outros opções, tais como Xml-RPC.

A segunda camada é chamada Descrição, e tem como objetivo descrever: (1) as funções que cada serviço pode prestar (descrição da implementação); (2) que informações de entradas são necessárias para que o serviço possa ser executado (descrição da interface); e (3)

quais os tipos de resultados devem ser esperados (também na descrição da interface). A camada de descrição segue uma padronização chamada WSDL (*Web Services Description Language*), que também é definida com base no padrão XML.

Ainda na segunda camada, temos a descrição das funções transacionais e de orquestração ou coordenação. Com elas, é possível especificar como o serviço que está sendo descrito pode se encaixar num processo de negócio como um todo. A especificação BPEL4WS, por exemplo, que está em fase adiantada de padronização, permitirá que essa camada seja integrada aos pacotes de gerenciamento de processos e às ferramentas CASE (Computer Aided Software Engineering).

A terceira camada, chamada Descoberta, é descrita pelos padrões UDDI (*Universal Description Discovery and Integration*) e WS-Inspection (*Web Services Inspection Language*). As definições dessa camada permitem que as empresas e agentes envolvidos numa interação possam procurar e descobrir serviços que sejam interessantes para as suas operações. Para isso, já existem organizações se mobilizando para fornecer portais com serviços de busca e diretórios por categoria de serviços, no modelo de páginas amarelas. Um exemplo é o site www.xmethods.com.

Além destas três camadas conceituais, existem outras três camadas que permeiam e controlam cada uma das camadas acima citadas. A camada de controle de qualidade contabiliza as falhas que por ventura venham a ocorrer durante a execução do WS, e verifica se o tempo de resposta do serviço está dentro do pré-estabelecido. Já a camada de segurança

verifica quem está acessando os serviços, e monta os históricos de acesso. Finalmente, a camada de gerenciamento permite que o provedor dos WS possa monitorar e controlar as estatísticas, históricos e parâmetros do serviço que estão sendo gerados pelas duas camadas anteriores (Kreger, 2003).

Os padrões definidos para a utilização de WS baseiam-se em técnicas que são bem conhecidas e já foram utilizadas anteriormente sob outras formas. Por exemplo, o padrão UDDI pode ser visto como uma evolução do modelo de busca e consulta de APIs (Application Programming Interface), e o WSDL pode ser visto com uma evolução da publicação e especificação de interfaces. Entretanto, diferentemente de tecnologias anteriores, este padrão está sendo adotado de forma conjunta pelas maiores empresas do setor de desenvolvimento de software e este fator pode trazer inúmeros benefícios para quem desenvolve e para quem se utiliza de SI (Iyer et al., 2003).

2.2. Micro e Pequenas Empresas (MPE) desenvolvedoras de software

Grande parte da economia brasileira está calcada nas operações de micro e pequenas empresas. Segundo os dados do SEBRAE (2004), 84,79% das empresas instituídas formalmente no Brasil são consideradas como MPEs, empregando 60% da força de trabalho do país. Ainda segundo este mesmo órgão, 64% das empresas exportadores se enquadram no padrão de negócios de micro e pequeno porte, representando 7,6 bilhões de dólares em vendas para o mercado externo (Sebrae, 2004).

A classificação de empresas utilizada pelo SEBRAE tem por base a classificação do IBGE (Sebrae, 2004) que define:

- **Microempresa:**

  i) na indústria: até 19 pessoas ocupadas;

  ii) no comércio e serviços: até 09 pessoas ocupadas;

- **Pequena empresa:**

  i) na indústria: de 20 a 99 pessoas ocupadas;

  ii) no comércio e serviços: de 10 a 49 pessoas ocupadas.

Em particular, dentro das atividades de comércio e serviço no país, 80% das empresa se enquadram como MPE. Segundo a pesquisa anual de Comércio e Serviços do IBGE, estima-se que, em 2001, foram ocupados 7,3 milhões de postos de trabalho neste setor e que foram gerados R$ 168,2 bilhões em receita operacional líqüida e R$ 61,8 bilhões em valor adicionado (IBGE, 2003).

Dee acordo com Nogueira Neto (2002), uma das características comuns nas empresas de pequeno porte é o baixo nível de utilização de Tecnologia de Informação (TI). Geralmente, o que se encontra nesse tipo de empresa é uma cultura informal, com pouca organização do uso da informação. Vale ressaltar que essa característica não é exclusiva do Brasil ou dos países de economia emergente. Diversos estudos realizados no Reino Unido demonstram que as MPE evitam aplicativos ou softwares sofisticados, e que seus membros não têm conhecimento suficiente para utilizar sistemas de informação de forma efetiva (Bunker,

2002).

Um outro ponto interessante são os fatores que motivam MPEs a utilizarem soluções de TI. Em estudos realizados na década de setenta no Reino Unido, constatou-se que o entusiasmo pela tecnologia por parte dos sócios era o grande fator de adoção de TI nas pequenas e médias empresas (Levy, 2001). Este argumento faz sentido se for levado em conta que o processo decisório das MPEs é feito por uma ou duas pessoas, que geralmente são os próprios sócios da empresa (Cloete, 2002).

Thong (1999) apresenta conclusões semelhantes quando relata, através de uma pesquisa quantitativa, que as pequenas empresas que possuem diretores (CEOs) inovadores com conhecimento de SI, uma atitude positiva em relação às vantagens, compatibilidades e complexidades de SI, e empregados com maior conhecimento de SI tendem a adotá-los com maior facilidade.

Não foi possível encontrar dados ou estudos específicos sobre pequenas e médias empresas de desenvolvimento de software nas buscas realizadas na literatura relevante para este trabalho. Contudo, é importante citar o trabalho de Machado et al. (2001), que investigou empresas de base tecnológica. Tendo por base a definição do Office of Technology Assesment (OTA) do congresso norte-americano para empresas de alta tecnologia e a definição do SEBRAE para micro e pequenas empresas, os autores conceituaram MPEs de base tecnológica da seguinte forma:

> Micro e pequenas empresas de base tecnológica são empresas industriais com menos de 100 empregados, ou empresas de serviço com menos de 50 empregados,

> que estão comprometidas com o projeto, desenvolvimento e produção de novos produtos e/ou processos, caracterizando-se, ainda, pela aplicação sistemática de conhecimento técnico-científico. Estas empresas usam tecnologias inovadoras, têm uma alta proporção de gastos com P&D, empregam uma alta proporção de pessoal técnico-científico e de engenharia e servem a mercados pequenos e específicos (Machado, 2001, pg.7).

Deve-se notar que essa caracterização não se enquadra no conceito de MPEs definido pelo SEBRAE e IBGE, além de não distinguir as empresas industriais cujo foco é a construção de equipamentos, das de desenvolvimento de sistemas de informação. No entanto, a definição sugere as três principais características distintivas das MPEs de base tecnológica: (1) o porte da empresa; (2) o uso intensivo de tecnologias inovadoras para geração de novos produtos e serviços e (3) o foco em nichos de mercado.

Uma outra pesquisa citada por Paduan (2000) informa que, em 2000, no Brasil, existiam 5.400 empresas ligadas à produção e comercialização de software, contando com um total de 158.000 funcionários. Assim, segundo a pesquisa, pode-se estimar que existem em média 29 funcionários por empresa, ou seja, pode-se dizer que a média das empresas de desenvolvimento de software se encaixam na categoria de micro e pequena empresa.

Sob o ponto de vista comercial, e segundo pesquisa da SOFTEX (2002), as empresas cuja a atividade fim seja o desenvolvimento de software podem ser categorizadas em dois grandes grupos:

1) Desenvolvimento de pacotes: neste grupo, o foco é desenvolver sistemas que atendem a um problema genérico, como por exemplo, uma folha de pagamento, um jogo para computadores ou um Sistema Integrado de Gestão. Geralmente, o

software é de propriedade da empresa desenvolvedora, que vende a licença de uso ou cobra uma taxa ou aluguel.

2) Desenvolvimento para terceiros: neste caso, a empresa é contratada para resolver um problema específico de uma empresa. Neste caso, o software pode ser propriedade de quem desenvolve ou da empresa que contrata o desenvolvimento.

No primeiro grupo, o desenvolvimento se assemelha a uma linha de produção física, onde se busca identificar as necessidades do mercado e em seguida se cria um produto para atender a esta necessidade. Estes sistemas são construídos para resolver problemas genéricos e são vendios em quantidade, sem que haja customização específica para um dado cliente. O produto desenvolvido é chamado de software de “prateleira” ou “pacote” para indicar que este pode ser vendido tal como um pacote de arroz em uma prateleira de supermercado (Küster, 1999; SOFTEX, 2002).

Já no segundo grupo, o desenvolvimento pode ser conceituado muito mais como uma **prestação de serviço.** Gianesi (1996) chama a atenção para algumas características fundamentais deste tipo de atividade:

> Os serviços são experiências que o cliente vivencia enquanto que os produtos são coisas que podem ser possuídas. A intangibilidade dos serviços torna difícil para os gerentes, funcionários e mesmo para os clientes, avaliar o resultado e a qualidade do serviço. (Gianesi, 1996 pg.32)

Ainda segundo Gianesi, como em qualquer outra atividade de serviços, pode-se enumerar as características que são inerentes a este setor:

- a interação do cliente que geralmente é alta e, no caso específico de

desenvolvimento de software, na maioria das vezes, essencial;

- A questão da intangibilidade dos serviços é mais forte no desenvolvimento de software do que em outros setores. Muitas vezes o próprio resultado do serviço, o software, será tão intangível quanto o serviço em si;
- A produção e o consumo de serviços são simultâneos;
- Serviços não podem ser estocados.

Estas características podem ser vistas até mesmo para os chamados softwares de "prateleira" ou "pacotes" e muitas vezes este serviço é tão ou mais importante que o software em si.. Ou seja, até mesmo os softwares "de prateleira" pressupõe algum tipo de serviço embutido, seja este um serviço de treinamento, ou de suporte para correção de falhas ao software.

2.3. O processo de Desenvolvimento de Sistemas de Informação (DSI)

A forma como o software é modelado e construído pode ser o grande diferencial para uma empresa que fornece o serviço de DSI, tendo impacto direto em sua competitividade. Assim, é importante modelar e entender como funciona este processo para que se possa encontrar indicadores que possam medir seu desempenho e servir de base para a sua melhoria.

Segundo Olson (1994), estas é um dos assuntos mais estudados atualmente na área de Sistemas de Informação. Existem diversas iniciativas que buscam generalizar este processo e uma das mais conhecidas é da Universidade Carnegie Mellon que, em conjunto com diversas

empresas, propôs o modelo CMM (Capability Maturity Model). O modelo descreve princípios e práticas para medir a maturidade do processo de desenvolvimento de software. Sendo uma aplicação dos princípios de Gerenciamento de Qualidade Total – TQM e a sua ênfase é na satisfação do cliente do processo de DSI (Olson, 1994).

O objetivo do CMM é auxiliar as empresas desenvolvedoras a melhorar a maturidade de seus processos através de um caminho evolucionário que abrange desde os mais caóticos processos, até os mais maduros e disciplinados. O foco é identificar áreas chaves do processo de DSI e determinar práticas exemplares que podem levar a um processo mais consistente, eficiente e eficaz (Olson, 1994).

Assim como o CMM, os modelos ISO-9000, SPICE, TRILLIUM e GQM/QIP buscam a identificação de práticas que melhorem o processo de DSI, incluindo auditorias, treinamento, definição do projeto em si e ações corretivas (Sorumgard, 1997).

Como o processo de DSI abrange diversas técnicas e envolve inúmeros passos, é importante definir um modelo para que se tenha uma visão geral das principais etapas do processo. O modelo tomado como base para o estudo é o resultado de uma generalização montada a partir do padrão ISO/IEC 12207 (1995) e da compilação das idéias de vários autores (Boehm, 1981; Stair, 1998; Beck, 2000; Russ, 2000; Grew, 2005). A tônica na confecção deste modelo foi buscar os elementos comuns a todos os modelos estudados com o intuito de simplificar a visualização do processo de DSI.

Segundo Stair (1998), o processo de DSI se inicia a partir da análise de um problema

encontrado por um indíviduo ou empresa que acredita, ou espera, que o dado problema possa ser solucionado com o uso de um SI. O futuro usuário busca profissionais especializados que irão avaliar como o uso de SI pode solucionar o problema para, em seguida, propor uma solução adequada. No caso de uma empresa que esteja necessitando deste serviço, ela pode utilizar recursos internos, ou seja, funcionários que já desenvolvem ou mantêm SI internamente ou pode contratar fornecedores externos, que é o foco deste estudo.

Em ambos os casos, uma vez contratados, os desenvolvedores iniciam o processo utilizando uma metodologia de desenvolvimento, ou seja, um agregado de técnicas e ferramentas que tem por objetivo padronizar o processo de desenvolvimento de sistemas em uma empresa (Ghezzi, Jazayeri e Mandrioli, 1991). Pressman (2001) define ainda que uma metodologia para desenvolvimento de sistemas especifica a seqüência de passos e a serem seguidos durante o desenvolvimento de um sistema de informação. A cada um destes passos, associa-se um conjunto de atividades, seus produtos e as regras de verificação que garantem a passagem para a próxima fase (Pressman, 2001). As metodologias tradicionais w*aterfall* (Royce, 1987 *apud* Boehm, 1998), em espiral (Boehm, 1998), RUP (Kruchten, 2003) e as mais modernas tais como Extreme Programming (Beck, 2000 *apud* Teles, 2004) são exemplos de metodologias de DSI.

O processo se inicia quando a empresa contratante define o escopo do problema e quais são os requisitos gerais necessários para a sua solução. Estes requisitos podem ser apresentados sob forma de RFP (*Request for Proposal*), ou seja, um documento formal que

descreve as necessidades da empresa, ou podem ser através de entrevistas, feitas com o analista de sistemas (O'Brien, 2003 pg.344).

Assim, partindo do pressuposto que o levantamento inicial foi feito a contento e o contratante fez uma análise da viabilidade do projeto e já aprovou a sua execução, o próximo passo seria definir uma arquitetura e infra-estrutura de TI adequada para que o sistema possa ser desenvolvido. Neste ponto, o analista de sistemas irá planejar a arquitetura baseado não só nas necessidades do cliente, mas também nos recursos (software, hardware e equipe de desenvolvimento) que dispõe para a construção do software (O'Brien, 2003, pg.328).

De acordo com Teles (2004), uma vez tendo feito o levantamento inicial, o projetista monta uma proposta para apresentar ao cliente definindo o escopo do que será construído, preço, prazo e forma de pagamento. O autor afirma que esta negociação pode se dar de duas formas, em função do escopo e da precificação do sistema. Esta precificação pode ser categorizada em dois tipos: fixa ou variável, ou seja, ou o contratante paga um preço fixo pelo sistema, ou ele paga pelas horas trabalhadas a medida que o projeto for sendo entregue (Teles, 2004).

No primeiro caso, a empresa contratada geralmente adiciona um valor monetário extra por conta do risco de não conhecer os detalhes do projeto. Conforme explica Patton (2003), por não saber exatamente qual é o tamanho do trabalho a ser realizado já que o cliente dificilmente transmite todos os detalhes durante o levantamento inicial, a empresa contratada tem que embutir no preço final um valor extra por conta do risco que está correndo de ter que

gastar mais recursos do que tinha planejado inicialmente.

Uma metáfora interessante para descrever este fenômeno seria a do arquiteto que é contratado para projetar um prédio e a ele só dado a área do terreno e quantos andares o prédio terá. A construtora, entretanto, não especifica quantos apartamentos terão por andar, ou quantos quartos terá cada apartamento, mas quer que o arquiteto estime um valor para a obra. Assim, para garantir que a construtora não terá prejuízo, o arquiteto acaba por orçar o prédio por um valor muito maior do que poderia ser definido caso todas as informações estivessem disponíveis.

Segundo Teles (2004), projetos deste tipo geralmente encontram resistências por parte do desenvolvedor, pois, para miminizar seus riscos, ele tentará minimizar os tempos de testes e as funcionalidades que devem ser implementadas comprometendo a qualidade do sistema. É comum surgirem situações tais como: "Diga que o software está completo, mesmo quando você sabe que ele não atende a todos os requisitos" ou "Pare de perder tempo testando", ou pior "Envie a primeira versão que funcionar, sem se preocupar em fazê-la mais fácil de manter ou mesmo compreensível".

Já um projeto de escopo variável é pago pelo número de homem-horas trabalhadas independente do escopo do projeto. Neste caso, a empresa contratada tem a obrigação de fornecer os profissionais para trabalharem em determinado período e se compromete a entregar as funcionalidades básicas definidas no levantamento. Este tipo de projeto ainda é pouco utilizado, pois muitas empresas contratantes receiam que o produto não seja entregue

já que o escopo não foi definido a priori. Segundo Beck (2000), novas metodologias, chamadas metodologias ágeis, têm surgido para tentar minimizar esta percepção de risco por parte do cliente. Estas metodologias procuram entregar partes do sistemas em intervalos muito curtos (entre 1 e 3 semanas) para que o cliente possa ver que o projeto está atingindo os resultados esperados e de acordo com suas necessidades (Beck, 2000; Teles, 2004). Segundo Teles, no contrato de escopo variável, o cliente

> "tem uma idéia previsível sobre aquilo que ele pode vir a obter de antemão, mas a realidade rapidamente se intromete. O que ele irá obter no final é invariavelmente diferente daquilo que imaginou no início. Ao desistir da ilusão de que é possível controlar o escopo no início do contrato, ele obtém algo muito mais valioso. Ao invés de obter o que desejava no início do projeto, ele agora pode obter o que quer no final do projeto, depois de ter completado o processo de aprendizado". (Teles, 2004 pg. 227)

Uma vez que o projeto foi acordado entre as empresas, os desenvolvedores iniciam a fase de levantamento de requisitos funcionais, isto é, obtêm do contratante os detalhes do problema e quais as formas ideais para solucioná-lo. Estes requisitos são obtidos através de reuniões presenciais e de análise de documentos da empresa. O resultado desta análise é materializado sob forma de modelos e no projeto de software, que, dependendo da metodologia utilizada, são levados para aprovação do cliente (O´Brien, 2003).

Uma vez tendo levantado o escopo do problema, os desenvolvedores projetam como o sistema será construido e em seguida passam para a fase de codificação ou programação do sistema. Esta codificação pode ser feita por programadores ou pelos próprios analistas que fizeram o levantamento.

Em metodologias mais tradicionais, o processo de levantamento de requisitos é feito

por completo até que todo o problema esteja mapeado e modelado (Steenis, 1989). Diversos autores ressaltam a importância de miminizar os erros nessa fase pois a posterior correção destes se mostra muito custosa (Bohem, 1987; Papaccio, 1988 apud Steenis, 1898). Já em metodologias ágeis, na medida em que pequenas partes do problema são levantadas, a modelagem e codificação já são iniciadas, para que o usuário possa testá-las e experimentá-las. Com isso, o risco de erro da modelagem do problema é quase nulo pois tanto o usuário quanto o desenvolvedor têm tempo de corrigir o problema rapidamente, sem que outros módulos tenham que ser reescritos. Nesta abordagem, pode-se dizer que se procura o conceito de j*ust-in-time*[3] para o desenvolvimento do sistema (Teles, 2004).

Da mesma forma, nas metodologias tradicionais, o processo de teste em conjunto com o usuário é feito ao final de todo o desenvolvimento, ou quando grande módulos do SI ficam prontos. Já em metodologias ágeis, o processo de teste faz parte do desenvolvimento como um todo e não como uma tarefa à parte. O importante, entretanto, é retratar que existe uma fase de testes e que esta é utilizada para aumentar a qualidade do produto que está sendo gerado. Esta fase também é chamada de controle de qualidade. Alguns autores defendem que deve existir um sistema de controle de qualidade (Software Quality Assurance) com um enfoque gerencial para controlar estimativas e tentar prever o tempo de desenvolvimento e a qualidade do software gerado (Gilb 1998; IEEE Software 1987 apud Steinner, 1989).

Ainda na fase de testes, é importante ressaltar a diferenciação de testes unitários e

3 Just-in-Time é uma técnica de produção originada no Japão onde os três principais elementos da manufatura – recursos financeiros, equipamento e mão-de-obra são colocados somente na quantidade necessária e no tempo requerido para o trabalho (Lubben, 1989).

testes integrados. Os testes unitários são teste realizados em cada módulo do sistema e geralmente são feitos pelos programadores. Já os testes integrados dizem respeito aos testes realizados para verificar a integração entre os módulos e podem ser feitos somente pelos programadores como também pelos usuários finais do SI.

Testes unitários testam cada interface que um módulo disponibiliza. O conceito de modularidade na arquitetura de SI influencia sobremaneira a forma como estes testes são realizados. Se um módulo não consegue encapsular suas funcionalidades, ou seja, se é possível executar partes deles sem que se utilize a interface disponível, os testes unitários tem pouca valia. Assim, quanto maior a modularidade que a arquitetura permitir, menor será a dificuldade em realizar testes unitários. Os testes unitários compõe uma dos pilares de metodologias mais recentes como XP ou Lean Development (Teles, 2004).

Seguindo por fim o processo de DSI, uma vez que o contratante testa e aprova o software desenvolvido, o projeto é encerrado ou entra em uma nova fase de refinamento, adição de novas funcionalidades ou correção de erros. Esta fase também é considerada por muitos autores como um processo à parte, chamado de manutenção do sistema (Steinner, 1990). Vale ressaltar que, em metodologias ágeis, o ciclo de aprovação por parte do usuário é constante, e não apenas no final do projeto, ou seja, a cada funcionalidade implementada, o usuário é chamado para testar e validar o que foi desenvolvido (Teles, 2004).

Ao longo da análise dos vários modelos, pode-se verificar que existiam dois tipos de desenvolvedores: um que seria mais experiente responsável pela análise do problema e pela

arquitetura a ser empregada e outro que estaria mais ligado mais às tarefas de codificação e de teste. Dentro das metodologias mais tradicionais, O primeiro é mais comumente conhecido como analista de sistemas enquanto que o segundo seria o programador (Steenis, 1981). Entretanto, as metodologias mais modernas não distinguem estes profissionais pelas funções que eles executam e sim pela sua experiência. Ou seja, em metodologias ágeis, o desenvolvedor deve levantar requisitos, projetar e codificar e os mais experientes terão o papel de Coordenação (Teles,2004).

O modelo pode ser visualizado através da notação de Diagrama de Atividades em UML (Furlan, 1998) na figura abaixo[4]:

4 Este diagrama foi apresentado ao grupo de usuários XP-RIO que reune aproximadamente 300 analistas de sistemas e outros desenvolvedores para discutir métodos ágeis e outras metodologias de desenvolvimento. O processo foi apresentado ao grupo que fez várias modificações até que se chegasse a um consenso mínimo de um modelo genérico para as diversas metodologias conhecidas.

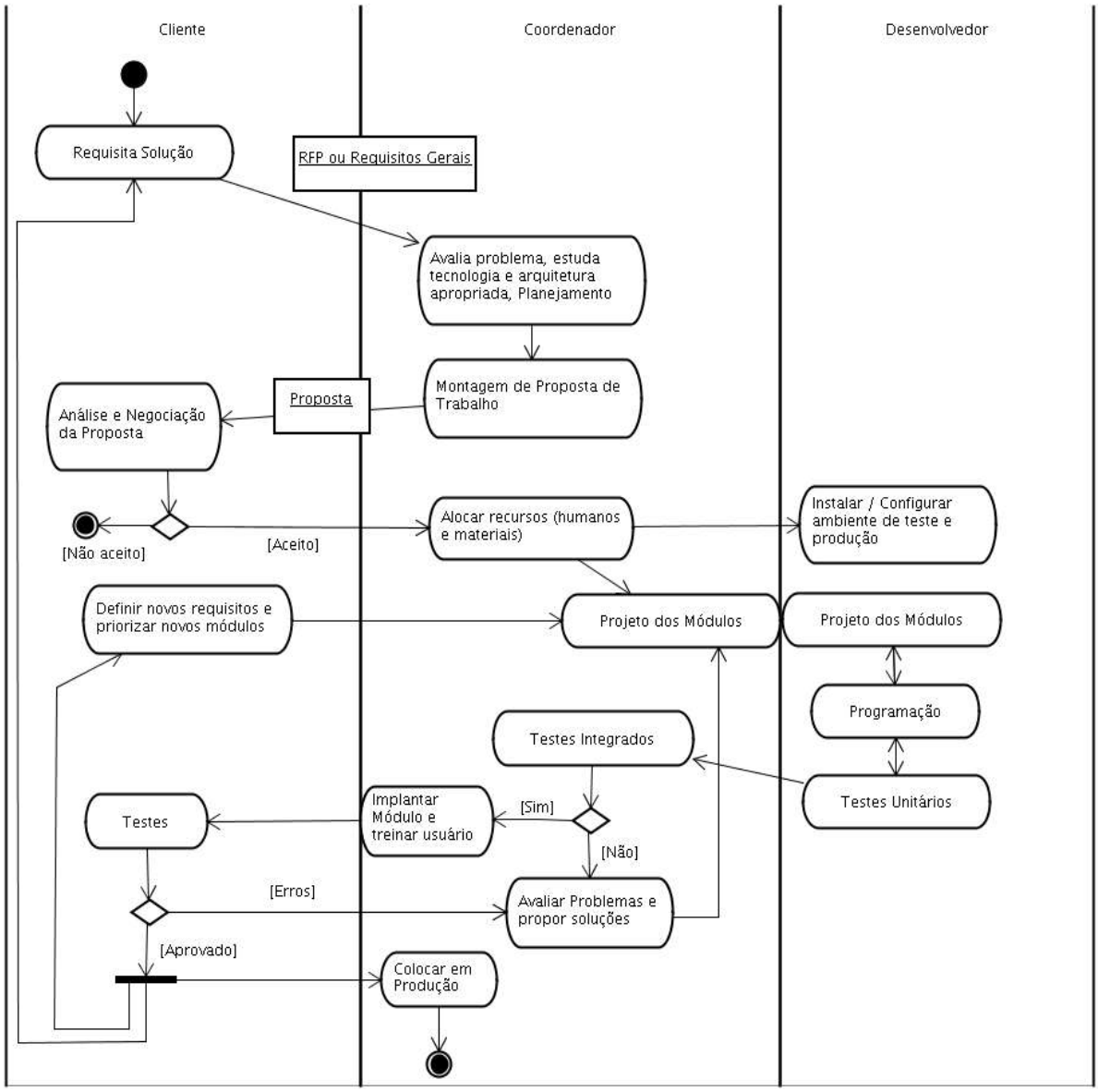


Figura 5: Processo de Desenvolvimento de Sistemas

De posse deste modelo, é possível enumerar diversos indicadores que podem ser utilizados para medir o desempenho do processo de DSI. De forma geral, é possível categorizar estes indicadores em três grupos: indicadores de qualidade , de custo e de produtividade, ou seja, de ordem econômica. No primeiro grupo, estão os requisitos de qualidade que o contratante deseja que o produto final tenha. Já os outros dois grupos são

requisitos que a empresa contratada precisa atingir para que a sua operação seja lucrativa, isto é, que o custo do DSI seja menor que o preço pago pelo contratante e baixo o suficiente para que o contratante possa pagar.

A questão da qualidade em software é um assunto muito abrangente, com diversas matizes. Aqui, será apresentada a definição na ISO/IEC 9126 (Bhatti, 2005), que também é utilizada pela Comissão de Estudos de Qualidade de Software da ABNT - Associação Brasileira de Normas Técnicas e já foi utilizada por diversos autores (Colombo, 2002; Chua, 2004; Hörbst et al, 2005) para a avaliação de arquiteturas e softwares. A norma define as seguintes categorias para os indicadores da qualidade:

1. **Funcionalidade (*Functionality*)**: indicadores que têm como objetivo verificar se o SI atende às necessidades do cliente. São eles:
   1.1. Adequação: verifica se os requisitos funcionais esperados foram definidos corretamente. Indicadores deste tipo estão ligados à fase de “Análise de Requisitos” e de “Projeto dos Módulos”.
   1.2. Acurácia: verifica o quanto dos requisitos funcionais desejados foram efetivamente realizados pelo sistema e se estão previstos.
   1.3. Interoperabilidade: mede como a interação do SI com outros sistemas está transcorrendo.
   1.4. Conformidade: mede o quanto o sistema está de acordo com as normas, leis, padrões e convenções.

1.5. Segurança de acesso: avalia se o acesso aos dados e ao SI são seguros e se existe o controle de quem utiliza o sistema.

2. **Confiabilidade (*Reliability*)**: utilizados para verificar o quanto o sistema está livre de falhas e como funciona a sua tolerância a falhas. Aqui também se avalia a recuperabilidade, isto é, o intevalo de tempo necessário para recuperar informações em caso de falha e durante quanto tempo é possível utilizar o sistema continuamente, sem interrupções.
3. **Usabilidade (*Usability*)**: avalia a facilidade de uso do SI por parte dos usuários, ou seja, qual o grau de dificuldade e quais as exigências requeridas para que um ser humano consiga utilizar o SI. Tem como indicadores secundários a Inteligibilidade (facilidade de leitura de como SI funciona), Apreensibilidade (facilidade de aprender a utilizar o SI) e Operacionabilidade (facilidade em operar o SI).
4. **Eficiência (*Efficiency*)**: mede quão eficaz é o sistema, ou seja, a relação entre a quantidade de recursos computacionais necessários e a quantidade de funcionalidades existentes no sistema.
5. **Manutenabilidade (*Maintenability*)**: mede a dificuldade em dar manutenção no SI, ou seja, qual o grau de complexidade e flexibilidade envolvida na alteração e inclusão de novas funcionalidades no SI. Tem como sub-características: Analisabilidade (facilidade para se analisar o SI), Modificabilidade (esforço necessário para que se consiga modificar o SI) e Testabilidade (esforço necessário para se testar o sistema).
6. **Portabilidade (*Portability*)**: qual o nível de adaptabilidade e de independência entre o SI e

as tecnologias utilizadas por ele, ou seja, qual o grau de facilidade encontrado ao se tentar utilizar o SI em outras plataformas de hardware e software. Avalia também a facilidade de instalação em diversas plataformas.

Os indicadores acima são definidos como externos pois dizem respeito à percepção do usuário em relação ao SI. A norma ISO/IEC 9120 ainda define métricas internas que dizem respeito às características inerentes ao sistema e independentes de sua execução, como por exemplo linhas de código, número de erros encontrados em revisões, etc. Bhatti (2005) sugere, entretanto, que indicadores que não têm efeito na qualidade do SI e não devem ser estudados, isto é, a sua utilização só seria recomendada se um indicador interno influencie uma métrica de qualidade.

O segundo grupo de indicadores diz respeito a requisitos de ordem econômica. Eles visam avaliar as prerrogativas da empresa que desenvolve o software, sob o ponto de vista de custo e de produtividade do processo de DSI. Favaro (1996) indica que a empresa deve buscar a qualidade sem perder, entretanto, o foco no cliente e na sua lucratividade. Ao fornecer um serviço de qualidade a um preço que o cliente não esteja disposto a pagar ou oferecer um nível de qualidade além do que o cliente precisa pode trazer graves conseqüências para a saúde financeira da empresa.

Alguns métodos também levam em conta o custo financeiro ao analisar a escolha de uma arquitetura de SI. Dentre eles, tem-se, por exemplo, o modelo CBAM (Cost Benefit Analysis Method) e ArchOptions baseado em Opções Reais (Bahsoon, 2003). Assim, é

importante que se calcule o custo para desenvolver uma solução utilizando uma dada arquitetura;

- infra-estrutura de software básico e de hardware
- ferramentas de desenvolvimento
- e de mão de obra.

O segundo ponto que influi no processo de DSI diz respeito a produtividade dos desenvolvedores. Aumentar a produtividade, ou seja, a eficiência da mão de obra, é um dos principais objetivos da empresa que desenvolve software. Por definição, produtividade refere-se ao total de recursos consumidos na produção de um determinado bem ou serviço por unidade de trabalho ou de capital empregado. Assim, é importante descrever os principais métodos descritos na literatura que buscam melhorar a produtividade no DSI.

Um dos principais indicadores que expressam a produtividade na produção do software é a reusabilidade. Segundo Coad (1993), este indicador procura medir quanto do código de um software foi reusável, isto é, qual o grau de utilização de um mesmo modelo ou código fonte para as diversas funcionalidades do software. O autor afirma que quanto maior for a reusabilidade, menores serão os recursos consumidos na produção de um software. A afirmação de Coad é corroborada por diversos autores que também citam que o reuso de código além de aumentar a produtividade também melhora a qualidade do software. (Banker e Kauffman, 1991; Basili, Briand e Melo, 1996; Chen e Lee, 1993 *apud* Rothenberger, 1998)

Em resumo, com base no modelo do processo de DSI, é possível avaliar os diversos

requisitos e indicadores envolvidos. Ao entender quem são os *clientes* do processo (as empresas contratadas e contratante) e quais os *requisitos* que cada cliente espera do processo, o empresário ou gerente da software-house pode definir estratégias e planos de ação para minimizar seus custos e maximizar os benefícios tanto para o contratante quanto para a empresa que desenvolve o SI. Nas próximas seções, o conceito de estratégia empresarial será revisado e como a tecnologia de Web Services pode contribuir para alterar os indicadores vistos nesta seção.

2.4. Estratégias Empresariais para MPEs

Nos tópicos anteriores, as técnicas, o ambiente das MPEs e o processo de desenvolvimento de software foram analisados com o intuito de delimitar o universo estudado. Entretanto, uma visão mais macro de como a empresa funciona pode se mostrar necessária para que os diretores de uma MPE possam buscar estratégias acertadas para sua realidade.

Henderson (1989 *apud* Huang, 2004) define estratégia, dentro da ótica da administração de empresas, como "uma busca deliberada por um plano de ação que irá desenvolver uma vantagem competitiva de um negócio e suas composições". Porter (1985) sugere quatro tipos genéricos de estrategias que podem ser adotadas para se ganhar vantagem competitiva. Essas estratégias são relacionadas ao escopo das atividades do negócio, podendo

ser estreitas ou amplas, e também ao grau em que a empresa quer diferenciar os seus produtos, como podemos ver na figura 6.

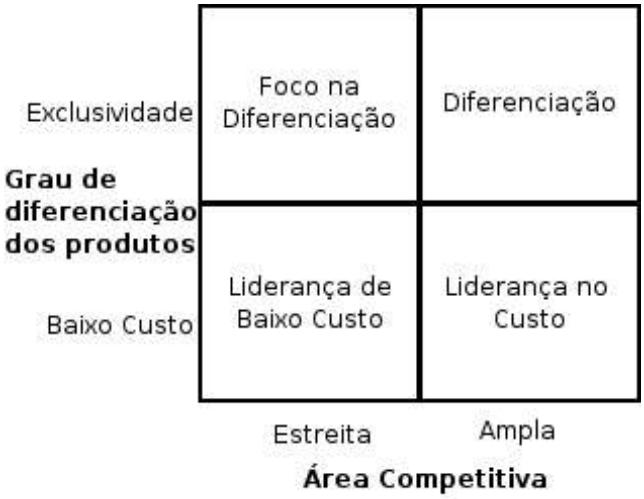


Figura 6: Quatro estratégias de Porter (1985)

Uma outra tipologia das estratégias foi definida por Miles e Snow (1978) que colocam o ambiente externo como principal fator na escolha dos gerentes pela estratégia a ser adotada para sua empresa. Dentro desse modelo, as estratégias podem ser categorizadas em quatro grupos:

- Prospectiva: onde o objetivo é conquistar novos mercados a partir da inovação e da maior abertura a correr riscos.
- Defensiva: é o oposto da Prospectiva, tendo por objetivo a estabilidade, e o atendimento dos clientes atuais, sem a preocupação de inovar ou crescer.

- Analisadora: Procura inovar apenas na periferia, mantendo suas atuais linhas de produtos estáveis e sem inovação. O objetivo é se arriscar apenas em novos mercados sem alterar a sua receita tradicional.
- Reativa: esta não é bem uma estratégia, pois a empresa apenas reage a ameaças imediatas e não tem planos definidos a longo prazo. Segundo os autores, esta é a "estratégia" mais perigosa para o futuro da empresa, pois estará sob ataque das empresas que adotarem estratégias prospectivas.

Dentro do enfoque deste trabalho, é importante notar que, em cada uma destas abordagens, a utilização da TI é vista como instrumento para implementar as estratégias definidas pelas empresas. Como já foi visto anteriormente, o início do uso de sistemas de informação para auxiliar a administração empresarial teve como objetivo a diminuição do custo de produção e coordenação e de transações internas propiciando a adoção de uma estratégia de baixo custo. Entretanto, as empresas perceberam que poderiam utilizar TI para adicionar valor a seus produtos e serviços e até mesmo para criar novas linhas de produtos baseadas em Tecnologia da Informação. Dessa forma, a TI torna-se instrumental para a implementação de uma estratégia de diferenciação (Daft, 2002; Porter, 2001).

O´Brien (2003) lista também alguns outros papéis estratégicos para SI, além da redução de custos e da diferenciação de produtos:

- Inovação: alterar radicalmente os processos de negócio, utilizando a TI;

- Alianças: criar organizações virtuais com parceiros comerciais, para facilitar a comunicação e difundir informações entre empresas parceiras;
- Barreira: fornecer sistemas proprietários com o intuito de erguer barreiras de entrada para novos entrantes e concorrentes, ou aumentar o custo de troca para outros produtos concorrentes.

A questão da estratégia em TI tem sido muito discutida pelo meio acadêmico. Em 2003, Nicholas Carr (2003), editor-chefe da Harvard Business Review, criou uma grande polêmica, ao afirmar que a TI não era mais importante para os negócios, pois tinha se tornado um *commodity* e, portanto, não gerava mais vantagens competitivas. A sua premissa é que a TI está presente em todas as empresas, e que o que faz um recurso ser verdadeiramente estratégico, atuando como base para a geração de uma vantagem competitiva sustentável, é que ele seja escasso e não ubíquo.

Comparando a tecnologia da informação com as ferrovias e a rede elétrica, Carr afirma que a TI atingiu um grau de maturidade que torna difícil sua utilização como forma de obter vantagem estratégica de longo prazo, da mesma forma que não se consegue hoje em dia obter tal vantagem com o manejo da eletricidade. Ele conclui o artigo sugerindo que as empresas devem gastar menos com TI; que não devem estar sempre a frente em tecnologia, mas sim utilizar soluções já testadas pelos outros; e que devem ter o foco nas vulnerabilidades, e não nas oportunidades que podem advir do uso das tecnologias da informação.

Baseado em uma pesquisa da McKinsey Global Institute de 2001, onde somente seis dos cinquenta e nove setores pesquisados, representando apenas 30% da economia americana, conseguiram introduzir inovações significativas nas suas práticas de negócios utilizando TI, Stewart (2003) corroborou o argumento de Carr, apontando os fracos resultados de produtividade obtidos por empresas que fizeram investimentos em TI.

De fato, a pesquisa citada sugere que algo está errado na forma como projetos de sistemas de informação são concebidos e conduzidos, já que 70% dos projetos avaliados não trouxeram os resultados esperados. Entretanto, diversos autores argumentam que um maior rigor nos índices de produtividade utilizados na pesquisa deveriam ser reavaliados pois muitos deles não conseguem captar os inúmeros retornos de investimento em TI e que se deveria buscar pesquisas orientadas a processos para que se obtivesse medidas mais precisas do retorno gerado pelos investimentos em TI (Mooney, 1996).

Mooney (1996) propõe um modelo de valor agregado para TI indicando que o valor que é agregado para o negócio está relacionado a fatores tecnológicos e organizacionais. Ele afirma que a TI somente agrega valor se ela for acompanhada por alterações nos processos de negócio da empresa. Ou seja, se a implantação de um projeto de TI não vier acompanhado de mudanças que melhorem, automatizem ou disponibilizem informações relevantes, precisas e atualizadas para os processos de negócios envolvidos, não se consegue gerar valor para a empresa.

2.5. Modelos de Negócios e Estratégias baseadas em Web Services

Empresários e diretores de empresas começam a se voltar para a tecnologia de WS, pois percebem que seus benefícios estão baseados em premissas estratégicas para a empresa. As arquiteturas tradicionais, que concentram suas atenções nos aspectos operacionais de projetos de TI, geralmente ignoram ou avaliam inadequadamente as conseqüências estratégicas da decisão de investimento (Ward et al. 1996; Counihan et al. 2002).

O primeiro benefício diz respeito ao alcance do mercado que pode ser atingido pelo uso da tecnologia. Como WS é basedo em padrões abertos, um número muito maior de potenciais clientes podem fazer uso de um serviço que venha a ser comercializado. Booch (2003) salienta que é possível interligar sistemas heterogêneos que antes não poderiam ser interligados com facilidade. Esta afirmação é corroborada por Barry (2003) que afirma que a implementação da arquitetura orientada a serviços utilizando WS traz um grau de interoperabilidade e flexibilidade que não era alcançado com as tentativas anteriores, utilizando tecnologias como CORBA ou DCOM. Até mesmo os padrões existentes anteriormente tais como o EDI (Eletronic Data Interchange) eram muito caros para pequenas empresas pois estas eram obrigadas a implementar sistemas de controle e gerenciamento que não eram compatíveis com a sua estrutura (Bunker, 2002).

Assim, Booch (2003) conclui que o que existe de novo na utilização de WS é o alto potencial de alcance dos serviços que pode trazer um grande impacto para o modelo econômico que predomina na indústria de desenvolvimento de software. Isto porque as

empresas podem utilizar a sua infraestrutura atual para implantar uma nova arquitetura orientada a serviços (SOA).

A utilização de WS também pode facilitar estratégias prospectivas. Com a tecnologia dos WS, será possível divulgar e distribuir os softwares pela web sem que o comprador precise de procedimentos complexos de instalação para utilizar os serviços disponibilizados. A nova arquitetura de WS pode estimular a criação de novos modelos de negócios, ou seja, novos produtos e serviços podem ser fornecidos, de forma a promover uma melhor integração com elementos do ambiente externo da organização, baseada na troca de informações que antes estavam isoladas em “silos” dentro das próprias empresas (Lim, 2003).

É importante ressaltar que a figura de quem provê e de quem consome WS não está unicamente atrelada à visão da empresa desenvolvedora de software e da empresa que apenas compra soluções de TI. Com esta tecnologia, uma empresa cujo foco não seja de TI pode vender para parceiros, serviços que ela própria utiliza. Para a empresa que não têm sua atividade fim na TI, é possível agora empregar a tecnologia de WS para vender suas aplicações internas para outras empresas com necessidades de serviços similares. Assim, os departamentos de TI poderão deixam de ser vistos como centros de custos, para tornarem-se centros de lucros (Lim, 2003).

Uma outra das vantagens dos WS é a possibilidade de utilizá-los em conjunto com sistemas legados o que pode facilitar estratégias defensivas ou analisadoras. Geralmente, os sistemas legados são antigas aplicações de software existentes nas organizações, que são

vitais para o funcionamento do negócio (Sayles, 1996 *apud* Siqueira, 2002). Conforme explicado acima, o uso de sistemas legado gera dificuldades para a adaptação da infraestrutura de TI a novas demandas do negócio (Brode, 1995 *apud* Kruchten, 2003). Muitas vezes, as regras de negócio embutidas nestes sistemas não são inteligíveis nem para analistas mantenedores do código, devido às inúmeras alterações que o software sofre ao longo de seu ciclo de vida (Gavino, 2000 *apud* Siqueira,2002). Assim, uma tecnologia que possa adicionar novas funcionalidades a sistemas legados, sem que seja necessário retirá-los de produção ou decifrar milhares de linhas de código de difícil manutenção, pode aumentar a adaptabilidade e agilidade de uma empresa, e, ao mesmo tempo, reduzir significamente o custo do desenvolvimento e manutenção de sistemas.

Segundo Clabby (2002), pode-se identificar três categorias de empreendimentos que estarão utilizando WS. A primeira delas engloba empresas que irão implantar os serviços em conjunto com seus sistemas legados. Estes serviços estarão confinados às suas redes internas e terão o objetivo de interligar seus sistemas legados entre si. O aspecto de autenticação e segurança não precisará ser tão rígido, já que os serviços não estarão interagindo com sistemas externos. Para este tipo de empresa, a adoção de WS permitirá que ela possa adicionar novas funcionalidades ao sistema legado de forma gradual.

A segunda categoria está associada a empresas que têm interesse em disponibilizar WS para clientes e fornecedores, e, com isso, melhorar o seu relacionamento com os elos de sua cadeia de suprimentos. Dentro desta categoria, podemos citar a Amazon.com, que vem

disponibilizando WS para seus fornecedores. Atualmente, já existem mais de 50.000 desenvolvedores utilizando os serviços disponibilizados pela empresa (VUNET, 2004).

Na terceira categoria, estão as empresas de software e consultorias que já dispõem de vários produtos em produção e, com isso, têm um potencial imenso a explorar. Estas empresas podem subdividir seus sistemas em pequenos serviços que podem ser vendidos separadamente, utilizando a infraestrutura da Internet. Desta maneira, elas poderão criar novos mercados, compostos por empresas que antes não estavam dispostas a comprar um pacote completo, mas tinham interesse em contratar apenas partes das soluções oferecidas pelas consultorias. Por outro lado, as empresas que irão comprar os WS também obteriam vantagens, pois não será mais necessário fazer grandes investimentos iniciais para começar a fazer uso dos serviços disponibilizados. Assim., poderiam experimentar diversos tipos de solução a um custo bem mais baixo que o de costume. Principalmente para as pequenas e médias empresas, que não dispõe de muitos recursos para avaliar e testar soluções complexas, adotar uma implementação gradual e pontual pode ser uma ótima estratégia.

Como em toda nova tecnologia, existem ainda desafios em WS, que devem ser levados em conta antes que uma empresa decida investir nessa tecnologia. Diversos autores (Lim, 2003; Boncella, 2004) apontam problemas relacionados à segurança, à busca de parceiros, e à capacidade de autenticação, que devem ser analisados cuidadosamente. Muitas empresas da indústria de software consideram a presente falta de solução para estas questões a maior barreira para a adoção em larga-escala da arquitetura de WS (Lim, 2003; Boncella, 2004).

Um outro ponto a ser considerado é a existência de diversos padrões definidos pela W3C que ainda não foram completamente implementados. É o caso do WS-Policy e o WS-Inspection, que dizem respeito à coordenação e integração do fluxo dos WS entre as empresas e em relação aos processos de negócios envolvidos. Da mesma forma, o padrão responsável pela segurança (WS-Security) ainda não está finalizado. Entretanto, muitas organizações já estão reestruturando sua infraestrutura de Internet, introduzindo a tecnologia de certificados digitais, LDAP e HTTPS, a fim de garantir a integridade, confidencialidade e autenticação dos WS (Datamonitor, 2003). Contudo, o ideal é que a segurança já estivesse embutida dentro dos pacotes SOAP. Além de permitir ganhos de produtividade, isto promoveria a compatibilidade e inter-operacionalidade das aplicações de WS, gerando ganhos maiores a longo prazo (Clabby, 2002).

Na questão relativa à descoberta de serviços, pode-se afirmar que as interações entre as empresas ainda não são viáveis por duas razões. Em primeiro lugar, conforme mencionado anteriormente, as padronizações necessárias precisar ser finalizadas, documentadas e distribuídas. Em segundo lugar, os canais apropriados deverão ser criados para que as ligações B2B possam ser estabelecidas. Parte disso é a negociação, criação e modificação de parcerias, já novos modelos de contrato deverão ser formulados para contemplar os novos tipos de serviços que serão prestados (Kreger, 2003).

2.6. Software Livre

Como a implementação de WS descrita neste trabalho somente utilizou ferramentas baseadas em software livre, é importante que se apresente os conceitos e limites relativos a esse novo modelo de desenvolvimento de software.

O termo de software livre surgiu formalmente através da criação de uma organização civil denominada Free Software Foundation, por Richard Stallman, em 1985. Essa organização foi um marco para a chamada “sociedade da informação”, pois materializou uma das mais importantes ações civis organizadas em defesa da liberdade de informação da história recente (Stallman, 1996). Stallman (1996) define software livre da seguinte forma: “software livre se refere à liberdade dos usuários executarem, copiarem, distribuírem, estudarem, modificarem e aperfeiçoarem o software”. Sob o ponto de vista empresarial, interpretações iniciais equivocadas assumiam que o software livre era sempre grátis, ou seja, que quem produzisse software livre não poderia vender e ter ganhos financeiros com os produtos desenvolvidos. Provavelmente, o termo em inglês para software livre (*free software*) deu margem a essa interpretação já que a palavra *free* pode significar tanto grátis quanto livre. Essa interpretação errônea fez com que muitos empresários não dessem importância para o novo modelo de negócio que estava surgindo (Kavanagh, 2004).

Apesar de ser originário de uma cultura *hacker*[5], específica de profissionais e estudantes de tecnologia, o modelo baseado em software livre começou a se mostrar uma alternativa atraente para o mundo de negócios, principalmente para empresas de TI que estavam estranguladas por um modelo monopolista e de alto custo personificado principalmente pela empresa Microsoft Corporation que detinha o domínio de mais de 90% do mercado de software para microcomputadores (US-DOJ, 1994; Kavanagh, 2004).

A atratividade do software livre aumentou bastante com o advento do sistema operacional Linux, criado pelo estudante finlandês Linus Torvald. Baseado totalmente em software livre, o Linux conseguiu mostrar-se eficiente em problemas que os sistemas operacionais Windows não conseguiam resolver ou resolviam de maneira sofrível ou a um custo muito elevado. (Kavanagh, 2004). Um outro exemplo de software livre bem sucedido desse fenômeno está no software Apache Web Server que gerencia e provê acesso às páginas Web na Internet. Sendo criado inicialmente para o Linux pela Apache Foundation, esse software foi sendo desenvolvido de forma espontânea por uma grande comunidade de programadores e analistas e ganhou a confiança das empresas desde o início da Internet comercial. Segundo uma pesquisa realizada em agosto de 2004, quase 70% de um total de mais de 53 milhões de servidores na Internet utilizam o Apache (Netcraft, 2004).

5 O termo "*hacker*" parece ter sido utilizado pela primeira vez pelo o grupo de pesquisadores de Inteligência Artificial do Massachusetts Institute of Technology. Representa as pessoas que seguem um modo de vida baseado em descobrir falhas nos softwares e construir novas utilidades para a Tecnologia da Informação (Levy, 1984).

Comercialmente, existem diversas formas de gerar receita através do software livre. A forma mais direta é a venda de produtos. O software livre pode ser grátis ou pode ser vendido da mesma forma que o software de código proprietário. Entretanto, para que o software vendido seja livre, a empresa que realizou a venda deve fornecer o código-fonte junto com o produto entregue. O código-fonte é o conjunto de instruções que "dizem" para o computador o que ele deve fazer. Nele está a descrição de como o software foi escrito pelos desenvolvedores. A partir desse código, outro programador pode entender como o sistema funciona e, assim, modificá-lo. Com isso, o modelo dá a liberdade para que o cliente possa modificar e acrescentar melhorias sem que seja necessário contratar o fornecedor original (Fitzgerald, 2003).

Outra forma comercial de utilização do software livre está em prover serviços, suporte e treinamento para aplicativos e sistemas que atualmente são livres e distribuídos gratuitamente. A partir deste tipo de serviço, surgiram novas empresas, que se especializaram em instalar e configurar softwares de código aberto e que vêm tendo um sucesso expressivo (Fitzgerald, 2003).

O fenômeno do software livre pode também ser estudado sob o ponto de vista das redes de relacionamento estabelecidas entre firmas, com o objetivo de intensificar sua interação. Balestrin (2004) sugere que tais redes promovem "uma redução espaço-temporal nas inter-relações dos seus atores, como fatores altamente estratégicos para a competitividade das organizações do século XXI". E o processo de desenvolvimento de software livre segue

um modelo de rede horizontal baseado em cooperação. Ao compartilhar o conhecimento, as empresas envolvidas conseguem diminuir os custos de pesquisa e desenvolvimento e os riscos envolvidos na busca das melhores soluções. Conforme afirma o autor, “as redes favorecem a concentração de esforços, sem privar a liberdade de ação estratégica de seus membros” (Balestrin, 2004 pg.242).

Um outro fator importante nas redes de software livre é o fato de que os desenvolvedores podem atuar em conjunto para conquistar novos clientes e promover ações de marketing que individualmente não teriam condições de fazer, por não terem uma marca forte por trás de seus produtos e serviços. Ao se estruturarem como redes, muitas vezes sob a édige de uma organização maior que congrege todos em uma única entidade, os desenvolvedores melhoram sua imagem junto a possíveis clientes e obtém legitimitidade em seu ambiente institucional (Dimmagio e Powell, 1983; Grabher, 1993 *apud* Balestrin, 2004).

Pode-se dizer que o software livre começa a se consolidar como uma solução viável economicamente para as empresas de forma geral. Especialmente para as MPEs, tanto para as que consomem quanto para as que produzem SI, esse modelo pode trazer inúmeras vantagens, devido ao seu caráter colaboracionista e de redução de custos transacionais e financeiros.

Percebe-se claramente um crescimento, mesmo que ainda tímido, do uso de software livre em todo mundo. Esse movimento vem ganhando força especialmente no setor governamental. Por exemplo, a prefeitura da cidade alemã de Munich está a frente da maior implantação de software livre no mundo, substituindo os sistemas da Microsoft por sistemas

abertos em mais de quatorze mil estações de trabalho (TechWeb, 2004). Aqui no Brasil, o governo federal já mostrou claras intenções de utilizar software livre sempre que possível, como pode ser visto no portal de Software Livre do Governo Federal (BRASIL, 2004).

Cerdeira (2004) cita ainda que, juridicamente, o software livre está de acordo com as leis brasileiras e nenhum fator impede as organizações de adotarem este modelo. O autor cita ainda que, devido ao caráter público e comunitário inerente ao modelo, várias prefeituras e estados têm dado preferência ao uso de SL.

Deve ficar claro que a adoção desse modelo não envolve apenas questões técnicas. Existem motivações políticas e ideológicas associadas à liberdade de expressão, ao fluxo de capital e à geração de empregos locais que motivam empresas e governos a optar por essa solução (UNCTAD, 2003; Cerdeira, 2004). Um exemplo desta afirmação está nas ações da organização Free Software Foundation que realiza muito mais operações legais ligadas aos direitos civis e a liberdade de expressão do que ao desenvolvimento de software em si (Stallman, 1996; Kavanagh, 2004).

Investir num modelo e mercado que ainda não estão consolidados pode ser bem arriscado, principalmente para pequenas empresas que dispõem de pouco capital para a pesquisa. Porém, como em todo investimento de risco, a aposta pode também render bons resultados para as empresas e profissionais que estiverem capacitados a utilizar, de forma eficaz e eficiente, esse novo modelo de negócios.

2.7. Conclusão

Para se encontrar os fatores críticos de sucesso na utilização de Web Services em uma MPE especializada em DSI, buscou-se explicar através do referencial teórico qual os quatro aspectos que serviram de base para a pesquisa:

- tecnológico – definir o que é arquitetura de SI, o que é WS e quais são seus benefícios.
- ambiental - da MPE desenvolvedora de sistemas de informação no Brasil
- do processo de Desenvolvimento de SI e seus indicadores de desempenho
- e estratégico - como a MPE deve se utilizar desta análise para buscar um diferencial competitivo e, no caso específico, utilizando software livre.

Enfim, procurou-se mostrar que a busca de novas formas de desenvolver SI se tornou vital para as micro e pequenas empresas que fornecem esse tipo de serviço e que o alto nível de fracassos em projetos de TI abre inúmeras oportunidades para a criação de novos modelos de negócios e de novas metodologias de desenvolvimento que busquem formas mais produtivas de trabalho. A utilização da arquitetura baseada em serviços (SOA) com Web Services, aliada a novas formas de arranjos estratégicos, tais como o uso de Software Livre pode abrir novos horizontes para as MPEs desenvolvedoras de SI.

3. METODOLOGIA DE PESQUISA

"Você não pode entrar no mesmo rio duas vezes". Heráclito (544- 484 a.C)

O objetivo principal deste trabalho é avaliar os fatores críticos de sucesso no desenvolvimento de sistemas de informação baseados em WS executado por uma pequena *software house* e identificar potenciais vantagens e percalços que possam ser encontrados durante o projeto. Ao tentar entender quais as vantagens competitivas e novas oportunidades que esta tecnologia pode trazer, e avaliar as barreiras que uma empresa com poucos recursos financeiros e humanos pode enfrentar ao adotá-la, o trabalho visa gerar subsídios para o processo decisório relativo a projetos de WS em micro e pequenas empresas (MPEs), de forma a aumentar as suas chances de sucesso.

Inicialmente, pretendia-se realizar um estudo sobre de caso de uma empresa que estivesse implementando a tecnologia de WS. Entretanto, o projeto que seria o alvo da pesquisa foi cancelado por motivos financeiros. Em função do grande interesse do pesquisador em estudar este tema, um segundo projeto, ainda em fase de análise de viabilidade, que inicialmente seria desenvolvido sem o uso de WS, foi reformulado para que se pudesse avaliar a nova arquitetura. A grande diferença entre os dois casos foi que, no projeto original, o pesquisador não participaria das decisões, atuando apenas como um

observador externo. Já no novo projeto, ele seria o próprio coordenador do empreendimento e teria a função, portanto, de negociar com os clientes, gerenciar e contratar os desenvolvedores, e definir as estratégias e ações para o desenvolvimento do sistema de informação.

Por conseguinte, procurou-se uma metodologia de pesquisa que permitisse uma participação ativa dos pesquisadores, influindo nas decisões e caminhos a serem seguidos no projeto. Após a avaliação de diversas abordagens, concluiu-se que as necessidades e restrições do projeto de pesquisa aqui considerado são plenamente atendidas pelo método da Pesquisa-Ação.

Neste capítulo, será feita uma explanação do histórico dos métodos de pesquisa-ação e o de fatores críticos de sucesso (FCS) e como eles são realizados. Em seguida, o plano da pesquisa será detalhado e quais serão as suas limitações.

3.1. Pesquisa Ação

O termo “pesquisa-ação” ou *Action Research* foi utilizado pela primeira vez pelo pesquisador inglês Kurt Lewis, em 1946, para descrever uma prática pioneira na pesquisa social, que combinava a geração de teoria com a mudança do sistema social pesquisado. Para ele, a ação do pesquisador é um meio pelo qual se modifica o sistema e se gera conhecimento crítico sobre o mesmo. (Susman, 1978).

As cartas escritas por Lewis, entre 1944 e 1946, expressavam uma profunda

preocupação e urgência em encontrar métodos para lidar com problemas sociais críticos, tais como o fascismo, o anti-semitismo, questões de minorias étnicas, etc. (Marrow, 1969 apud Susman, 1978). O trabalho de Lewin, a partir daí, buscava uma teoria geral que auxiliasse a implementação de mudanças sociais (Baskerville,1996). Na mesma época, um outro grupo também desenvolvia a pesquisa-ação no Instituto Tavistock de Relações Humanas, utilizando o método para tratar desordens psicológicas resultantes da permanência em campos de prisioneiros e campos de batalha (Rapoport, 1970 apud Kock, 1997). A partir de 1950, a pesquisa-ação foi largamente empregada na pesquisa e desenvolvimento organizacional, nas áreas da Educação e da Saúde Pública, e no estudo do trabalho comunitário principalmente na Europa e na Austrália (Coghlan, 2001).

O modelo de pesquisa proposto por Lewin tinha o objetivo de guiar investigações sociais de forma geral. Por conseguinte, o seu nível de abstração permite que seja utilizado em uma vasta gama de projetos. Entretanto, alguns críticos argumentaram que o modelo de Lewin não seria adequado para alguns tipos específicos de projetos, como, por exemplo, os na área de Sistema de Informação (Frank, 2001). Desta forma, vários pesquisadores como Checkland, Wood-Harper e Baskerville refinaram o modelo original para adequá-lo à pesquisa em SI. (Checkland, 1991; Wood-Harper & Baskerville, 1996).

Thiollent define Pesquisa-Ação como:

> um tipo de pesquisa social com base empírica que é concebida e realizada em estreita associação com uma ação ou com a resolução de um problema coletivo no qual os pesquisadores e os participantes representativos da situação ou problema estão envolvidos de modo cooperativo ou participativo". (THIOLLENT, 1986, p.14)

Conforme a definição de Thiollent, a metodologia de pesquisa-ação, ao contrário das análises fundamentadas nas perspectivas positivistas e críticas/materialistas, tem por base os princípios interpretativistas que substanciam o desenvolvimento de uma visão contextualizada dos fenômenos sociais e organizacionais (Klein e Myers, 1999). Vale lembrar que, na medida em que os sistemas de informação envolvem pessoas, regras e normas, funções, interações, etc., além, é claro, de hardware e software, sua introdução nas empresas constitui um fenômeno eminentemente social. Assume-se aqui, portanto, que o conhecimento detalhado sobre tais fenômenos só pode ser obtido através de construções sociais situadas histórica e regionalmente no contexto organizacional onde eles ocorrem (Klein e Myers, 1999).

Além disso, conforme explica Argyris (1985), a pesquisa-ação é um processo colaborativo de questionamento crítico que ajuda no aprendizado de teorias e práticas (apud Grant, 2003). Habermas (1973 apud Baker,2000) sugere que o objetivo primário da pesquisa-ação é aumentar o entendimento dos participantes em relação à interligação entre os problemas sociais e a teoria que está por trás das soluções dos problemas estudados. Segundo o filósofo, é correto afirmar que a relação dinâmica entre a teoria e a prática requer que ambas sejam consideradas de forma conjunta durante o curso do projeto de pesquisa-ação. A interação entre a teoria e a ação prática, através de um processo de reflexão crítica, leva a um entendimento mais profundo do problema e das configurações sociais que o cercam e o constituem (Masters, 1995; Grundy, 1982; Habermas, 1973 apud Baker, 2000). Hult e

Lennung (1980) também ressaltam o enfoque e o desejo de melhoria das competências dos participantes como uma das características fundamentais do processo de pesquisa-ação.

A pesquisa-ação também é uma pesquisa empírica, já que se baseia na experimentação direta dos fenômenos encontrados e nas transformações que ocorrem ao longo do estudo. O método encoraja o pesquisador a experimentar através de sua própria intervenção, e a refletir sobre os efeitos da sua intervenção no problema estudado, e sobre a sua relação deste último com as teorias existentes (Avison, 1999).

Por fim, pode-se afirmar que a pesquisa-ação é fundamentalmente colaborativa. A filosofia positivista encara o pesquisador como o único possuidor do conhecimento, a partir do qual as ações de pesquisa são desenhadas e posteriormente aplicadas em um mundo eminentemente passivo. Em contraste, o pesquisador envolvido em uma pesquisa-ação produz as soluções dos problemas através da colaboração com os participantes do projeto e suas organizações (Ackoff e Emery, 1972 apud Susman,1978).

Baskerville (1996) descreve o que seria um domínio ideal para a utilização de pesquisa-ação resumindo três características distintas do método:

(1) O pesquisador está envolvido ativamente na solução de um problema, de forma a gerar benefícios tanto para o pesquisador quanto para a organização.

(2) O conhecimento obtido pode ser imediatamente aplicado. Não faria sentido a figura de um observador externo, mas sim de um participante ativo que pudesse utilizar novos conhecimentos, com base em uma estrutura conceitual explícita e clara.

(3) A pesquisa é um processo cíclico, ligando a teoria e a prática.

O método de pesquisa-ação mostrou-se, portanto, adequado aos objetivos do presente estudo. Isso porque ele tem por base a participação direta dos pesquisadores no fenômeno investigado, prevendo uma colaboração constante e intensa entre estes e os participantes.

Um outro fator que motivou a utilização do método foi o fato de ele vir sendo utilizado com sucesso em projetos de pesquisa em Sistemas de Informação (SI). A metodologia foi aplicada no estudo na implantação e gerenciamento de SI (Mumford, 1983; Benbasat, 1984; Wood-Harper, 1985; Checkland, 1991; Tolvanen, 1995; Baskerville, 1996; Baskerville & Wood-Harper 1996; Lau, 1997), na análise da personalidade dos desenvolvedores (Kaiser, 1982) e especificamente no processo do desenvolvimento de SI (Ngwenyama & Grant, 1994; Grant, 1999; Baskerville & Pries-Heje, 1999).

Frank (2001), por exemplo, descreve estudos recentes utilizando pesquisa-ação para avaliar a introdução e utilização de Sistemas de Informação em pequenas e médias empresas na Alemanha. Mais recentemente, Grant (2003) estudou o uso de metodologias de construção de sistemas de informação, utilizando, também, a pesquisa-ação. Vale lembrar que Galliers e Land (1987) ressaltam que a modelagem matemática e experimentos em laboratórios têm se mostrado muitas vezes inapropriados para pesquisar sistemas de informação (apud Baskerville 1996).

3.2. Descrição do Método

O modelo original de Lewis era baseado em 6 fases: (1) análise, (2) descoberta de fatos, (3) conceituação, (4) planejamento, (5) implementação da ação e (6) avaliação dos resultados (Baskerville,1996). Já Blum (1955) propõe que a essência da pesquisa-ação é, na realidade, um processo com apenas dois estágios: diagnóstico e terapia. O estágio inicial envolvia a análise colaborativa da situação social pelo pesquisador e pelos participante da pesquisa. Ainda nesse estágio, hipóteses eram formuladas levando em conta a natureza do domínio do problema. O estágio terapêutico era baseado na introdução de mudanças no ambiente estudado, e na subseqüente avaliação de seus efeitos (Baskerville, 1996).

Thiollent (1997) considera que a pesquisa-ação deve ter no mínimo quatro etapas, conforme abaixo:

1. Etapa de Planejamento: é o momento inicial em que o pesquisador reúne as pessoas que estarão envolvidas na pesquisa para discutir o problema a ser enfrentado.
2. Etapa Exploratória: nessa etapa, o pesquisador coleta informações sobre as teorias possivelmente ligadas ao problema, e procura material de cunho empírico sobre o assunto. Ainda nessa fase, o pesquisador deve realizar entrevistas abertas ou semi-estruturadas, com o objetivo de detectar, nas verbalizações dos participantes, os principais problemas vivenciados e esclarecer os pontos que não ficaram bem entendidos na etapa de planejamento.

3. Etapa de Ação: A primeira ação é difundir os resultados das pesquisas para o grupo, para, em seguida, definir objetivos alcançáveis e apresentar propostas de ação. Por fim, deve-se negociar e implementar ações-pilotos, ou seja, ações que sejam modelos para as futuras ações que serão tomadas.
4. Etapa de Avaliação: é nessa etapa que o pesquisador poderá gerar conhecimento, extraindo os "ensinamentos" que poderão ser úteis para outras entidades. Também nessa fase, o pesquisador avalia a efetividade das ações implementadas no contexto organizacional, e suas conseqüências a curto ou médio prazo.

Diferentemente de Thiollent, Susman (1983) define 5 etapas para a pesquisa-ação, subdivindo a quarta etapa do primeiro autor para distinguir claramente a avaliação como um processo de controle distinto da geração de conhecimento.

Em particular, na área de Sistemas de Informação, Checkland é um dos autores mais citados na literatura relacionada a pesquisa ação. Seus trabalhos procuram dar um maior detalhamento ao método e enfatizar o lado humano da pesquisa. No modelo descrito por Checkland (1991, apud McKay, 1999), existem 3 elementos básicos: *intellectual framework* **(F)**, ou seja o contexto de idéias interligadas que representam as teorias relacionadas ao domínio do problema; a metodologia **(M)** que será utilizada nesse contexto; e a área de aplicação **(A)** onde está inserida a questão da pesquisa.

De acordo com Checkland, o processo de pesquisa se inicia com a identificação de um problema pelo pesquisador, no contexto estudado. A partir daí o pesquisador busca

subsídios teóricos para entender melhor o problema, para em seguida planejar ações sobre o caso estudado. O resultado deste planejamento irá resultar em ações práticas, com o intuito de resolver os problemas estudados. Nesse ponto, a pesquisa pode ser finalizada, ou, caso as questões não tenham sido satisfatoriamente respondidas, o ciclo deve ser reiniciado para que novas ações sejam planejadas. O modelo de Checkland pode ser visualizado na figura abaixo:

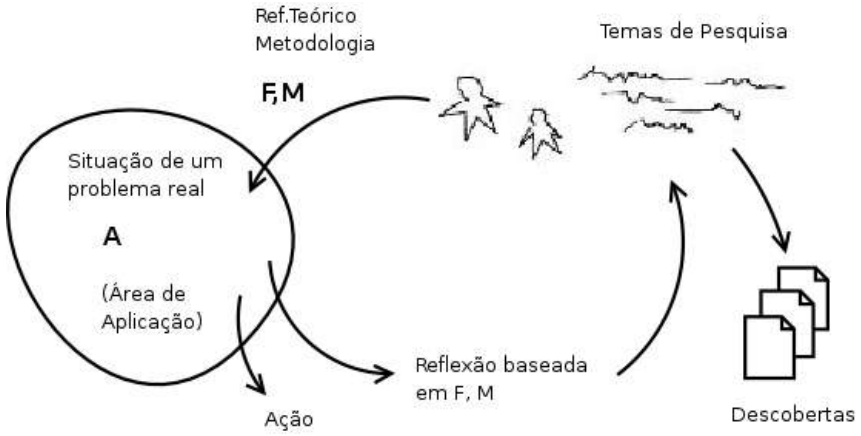


Figura 7: Pesquisa-ação (Checkland, 1991 apud Baskerville, 1996)

McKay e Marshall (1999) estenderam o trabalho de Checkland, e aprimoraram o modelo, argumentando que havia dois processos sendo executados simultaneamente nos projetos de pesquisa-ação: um método com intuito de resolver o problema prático ($\mathbf{M_{PS}}$); e um outro método, que visava a execução e controle da metodologia de pesquisa ($\mathbf{M_R}$). Além disso, os autores consideraram um outro elemento que deveria ser explicitado em qualquer pesquisa-ação: o problema específico (**P**) a ser resolvido pelo método **($\mathbf{M_{PS}}$)**. Com essa modificação, os autores pretendiam dar ainda mais rigor à pesquisa-ação, separando o interesse da pesquisa científica, do método de resolução do problema real. É importante

ressaltar a sutil diferença entre **(A)** e **(P)**: **(A)** é um problema genérico que se quer estudar e é de interesse maior do pesquisador, enquanto **(P)** é o problema específico que se estará estudando, ou seja, **(P)** é uma instância de **(A)**. O modelo estendido proposto por McKay e Marshall pode ser visto na figura abaixo:

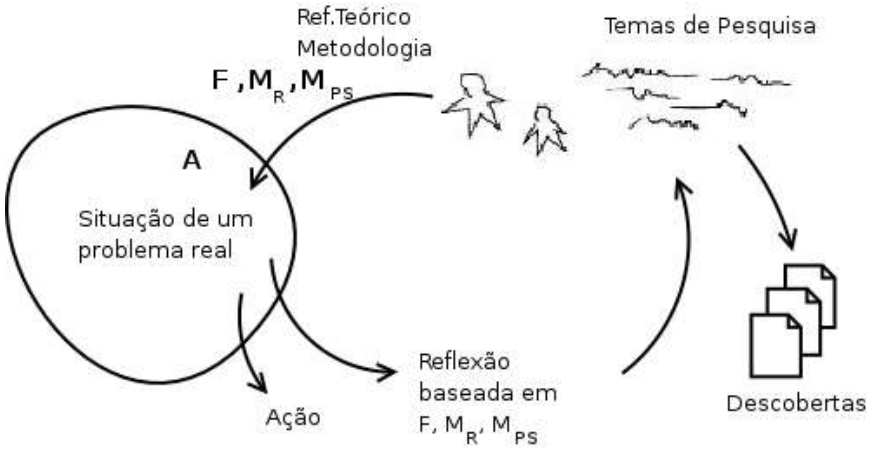


Figura 8: Pesquisa-ação (McKay e Marshall, 1999)

Tendo por base o modelo de McKay e Marshall (1999), pode-se identificar os elementos que foram contemplados na presente pesquisa, conforme a tabela abaixo:

Elemento	***Descrição***
F	Sistemas de Informação, WS, MPEs, Estratégia Empresarial, Software Livre
M_R	Pesquisa Qualititativa; pesquisa-ação baseada no método de McKay e Marshall (1999)
M_{PS}	Metodologia Rápida de Desenvolvimento de Sistemas (MRDS) adaptada para uma arquitetura orientada a serviços (SOA) com WS
A	Fatores Críticos de Sucesso (FCS) na utilização de WS para pequenas empresas especializadas em desenvolvimento de software
P	Desenvolver um sistema de informação contábil-fiscal para pequenas empresas, utilizando WS.

3.3.Fatores Críticos de Sucesso (FCS)

Neste ponto, é fundamental definir o conceito de Fator Crítico de Sucesso (FCS) já que o objetivo da pesquisa é identificar os FCS no desenvolvimento de SI utilizando WS. Segundo Rockhart (1979), fatores críticos de sucesso são fatores chaves que são obrigatoriamente necessários para se atingir um objetivo empresarial, isto é, os fatores que têm que dar certo para que um projeto ou negócio seja bem sucedido (Bullen e Rockhart, 1981, pg.385). Inúmeras pesquisas na área de SI identificaram FCS, inclusive no contexto de DSI (Wasmund, 1993; Greene, 1996; Rothenberger, 1998; Reel, 1999).

Inicialmente, o método utilizado para a análise de FCS era baseado em entrevistas com principais executivos de uma empresa, visando identificar os requisitos que eles entendiam como críticos para o sucesso da organização. Mais recentemente, o método foi expandido por Bullen e Rockhart (1981) em uma metodologia chamada SISP - *Strategic Information Systems Planing* – com o objetivo de utilizar o método para avaliar projetos independentes e não apenas a organização como um todo (Petkov, 2003). Wasmund (2003) ressalta que, através da identificação destes fatores críticos em cada projeto, as organizações aprendem a identificar o que é preciso ser melhorado prioritariamente para se obter sucesso na organização ou em um projeto específico.

De acordo com Wasmund (1993), o método se inicia com a definição de metas ou objetivos que se deseja alcançar. Para cada objetivo, diversos fatores são identificados através

de entrevistas estruturadas, realizadas com as pessoas envolvidas no projeto. A partir daí, o pesquisador procura definir quais são as tarefas que estão ligadas a cada um dos fatores montando uma matriz onde os fatores são colocados em colunas e as tarefas nas linhas. A montagem da matriz visa apenas facilitar o entendimento dos fatores e tarefas envolvidas e para descobrir tarefas que não estejam ligadas a nenhum fator (e consequentemente não precisam ser avaliadas) e fatores que estejam sem tarefas (o que leva o pesquisador a rever a lista de tarefas pois alguma tarefa tem que afetar o fator que não foi afetado).

Por fim, executa-se o projeto ou processo que está sendo estudado e valida-se os FCS identificados no estudo. Em função disso, é possível que se encontre outros fatores e se descarte um fator inicialmente identificado. A figura abaixo resume as etapas do método:

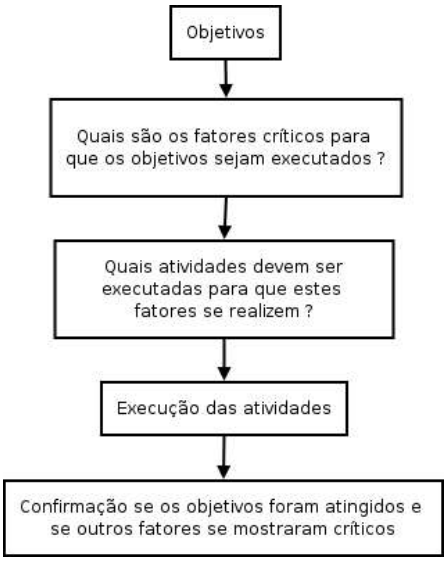


Figura 9: Etapas do método baseado em FCS

Assim, um dos primeiros passos nesta pesquisa foi identificar potenciais FCS de projetos de DSI baseado em WS a partir do referencial teórico e de entrevistas com desenvolvedores que já utilizavam WS. Esses fatores foram associados às tarefas de

desenvolvimento de software identificadas na Figura 5. Após a execução do desenvolvimento do sistema, os FCS foram validados a partir dos resultados encontrados na pesquisa-ação. Esta análise permitiu que se verificasse, dentro do âmbito da pesquisa, se algum fator não havia sido realmente crítico e se outros fatores que não tinham sido identificados inicialmente foram críticos para o sucesso do projeto.

Para identificar se um fator era crítico, buscou-se avaliar se os indicadores de desempenho definidos no referencial teórico associados aos conceitos de Funcionalidade, Confiabilidade, Usabilidade, Eficiência, Manutenabilidade, Portabilidade e Reusabilidade foram superiores aos anteriormente experimentados pelos desenvolvedores que participaram do projeto. Ou seja, se um dos fatores não fossem encontrado, isso poderia inviabilizar o projeto baseado em WS. Os indicadores de Adequação, Acurácia e Conformidade associados a Funcionalidade não foram contemplados pois assume-se, aqui, uma situação ideal em que o levantamento de requisitos foi feito sem erros. Como o usuário final não participou da pesquisa, a funcionalidade do sistema final não poderia ser avaliada.

Seguindo o conceito do modelo de valor agregado (Mooney, 1996), tentou-se identificar melhorias em um ou mais indicadores, em função da alteração do processo de DSI. Tais melhorias sugerem que houve benefícios na adoção dos WS, representando uma adição de valor ao negócio. De posse do resultado obtido, avaliou-se que fatores críticos ou indispensáveis para se alcançar os benefícios encontrados.

3.4. Plano de Pesquisa

Seguindo a metodologia de pesquisa-ação, o primeiro passo foi recolher e analisar o material bibliográfico disponível, ou seja, o *intellectual framework* (F). Este *intellectual framework* foi montado inicialmente a partir do material que dissertava sobre as várias arquiteturas utilizadas no desenvolvimento de SI para, em seguida, explorar os conceitos das arquiteturas orientadas a serviços e, especificamente, Web Services (WS). O foco durante todo o trabalho não estava no detalhamento técnico das arquiteturas em si, mas em como a arquitetura gerava valor e quais os diferenciais competitivos que cada arquitetura trazia para uma organização. Em particular, tentou-se obter as referências bibliográficas que abordassem experiências em pequenas empresas, preferencialmente situadas em território brasileiro.

Concomitante à pesquisa bibliográfica, um projeto real de DSI foi escolhido para que se pudesse avaliar a utilização da tecnologia de WS. O projeto escolhido pretendia integrar os sistemas de faturamento de MPEs com os escritórios contábeis que prestavam consultoria fiscal e contábil às mesmas empresas.

O ponto de partida do projeto de SI foi um trabalho de reengenharia (Hammer, 1990) realizado para um escritório de contabilidade que presta serviço para pequenas e médias empresas e para profissionais liberais aqui denominada ficticiamente de JP Escritório Contábil. O principal processo de negócio desse escritório consiste no recolhimento dos dados de notas fiscais emitidas, de pagamentos realizados por seus clientes (empresas que contratam o serviço contábil) e a contabilização dessas transações e o cálculo de tributos devidos pelos

técnicos contábeis. Ao final de cada mês, o escritório gerava um balanço detalhado da contabilidade e os enviava por correio ou através de mensageiros para cada cliente.

Com o processo de reengenharia dos processos, sugeriu-se que se desenvolvesse um SI para automatizar as diversas fases do processo. O objetivo do sistema era permitir, através da Internet, que os próprios sócios ou funcionários das empresas pudessem digitar suas notas fiscais e pagamentos e, consequentemente, liberar os técnicos contábeis da tarefa de codificar e digitar os valores informados pelos clientes. Este sistema foi denominado GOPE – Gestão Online para Pequenas Empresas.

Percebeu-se desde o início do levantamento que era fundamental criar interfaces que pudessem interagir tanto com o sistema contábil legado, quanto com os sistemas das empresas-cliente. Da mesma forma, dever-se-ia levar em conta que nem todas as empresas teriam o interesse em utilizar a ferramenta dado a pouca experiência que tinham com TI. Assim, para estes casos, o próprio escritório poderia utilizar o acesso via Web para atualizar os dados no sistema contábil legado.

O SI envolveria, portanto, os seguintes atores:

- **Escritório Contábil**: que iria receber os dados do sistema através do SI para seu sistema contábil legado.
- **Empresa Cliente:** que iria digitar ou exportar os dados através do SI.
- **MPE Desenvolvedora (*software house*):** que iria desenvolver o SI e fornecer os serviços aos outros atores. O pesquisador e os outros desenvolvedores contratados

para participar do projeto estavam alocados a essa empresa.

- **Outros Desenvolvedores:** outra empresas ou desenvolvedores que viessem a participar do projeto, integrando seus sistemas ao projeto GOPE.

A figura 8 representa as relações entre os atores e os sistemas envolvidos no projeto que será descrito nos próximos parágrafos.

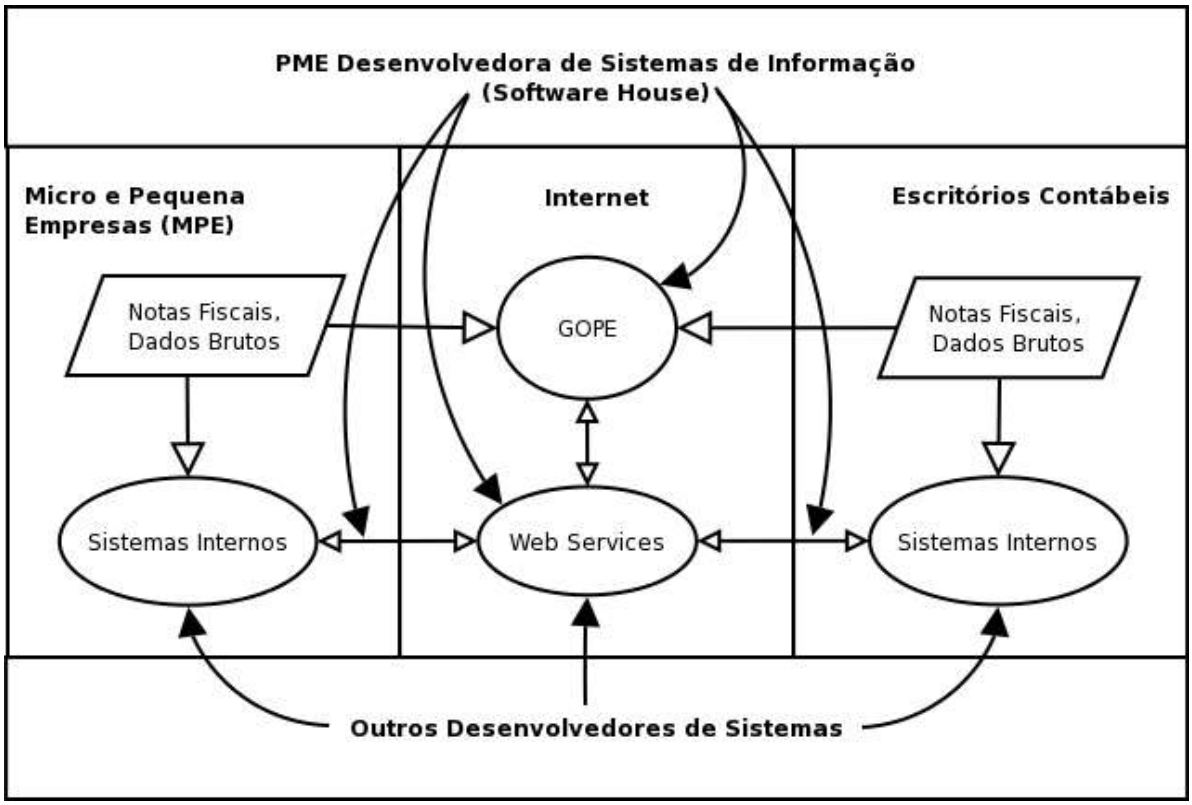


Figura 10: Escopo do Projeto GOPE

A empresa desenvolvedora responsável pelo projeto se chama **JSK** e um dos pesquisadores é um dos sócios. As empresas usuárias do SI são, respectivamente, **JP Escritório Contábil** e uma micro-empresa chamada **Cameo Modas** que fornece os dados

para o escritório contábil mensalmente. As duas empresas usuárias tiveram seus nomes alterados por questões de confidencialidade. As três empresas são localizadas no Rio de Janeiro.

O sistema seria inicialmente desenvolvido utilizando uma arquitetura de 3 camadas baseada na arquitetura Web. Entretanto, para que se pudesse estudar a nova arquitetura orientada a serviços, preferiu-se utilizar WS. Assim, a escolha da arquitetura foi feita em função dos objetivos da pesquisa, para avaliar os impactos da tecnologia em um projeto real.

Como foi explicado no referencial teórico, optou-se por utilizar apenas ferramentas baseadas software livre já que o uso de ferramentas proprietárias iria tornar o projeto inviável devido a alto custo de aquisição destas ferramentas. Assim, todo o projeto foi concebido para ser desenvolvido utilizando software livre (Isso inclui os aplicativos utilizados para troca de correio eletrônico, e a montagem dos modelos, esquemas, gráficos e todo o material de apoio do projeto).

Neste ponto, vale a pena resumir o planejamento geral da pesquisa de acordo com o modelo definido por McKay e Marshall (1999) para se ter uma visão geral do projeto dentro da perspectiva metodológica da Pesquisa-Ação:

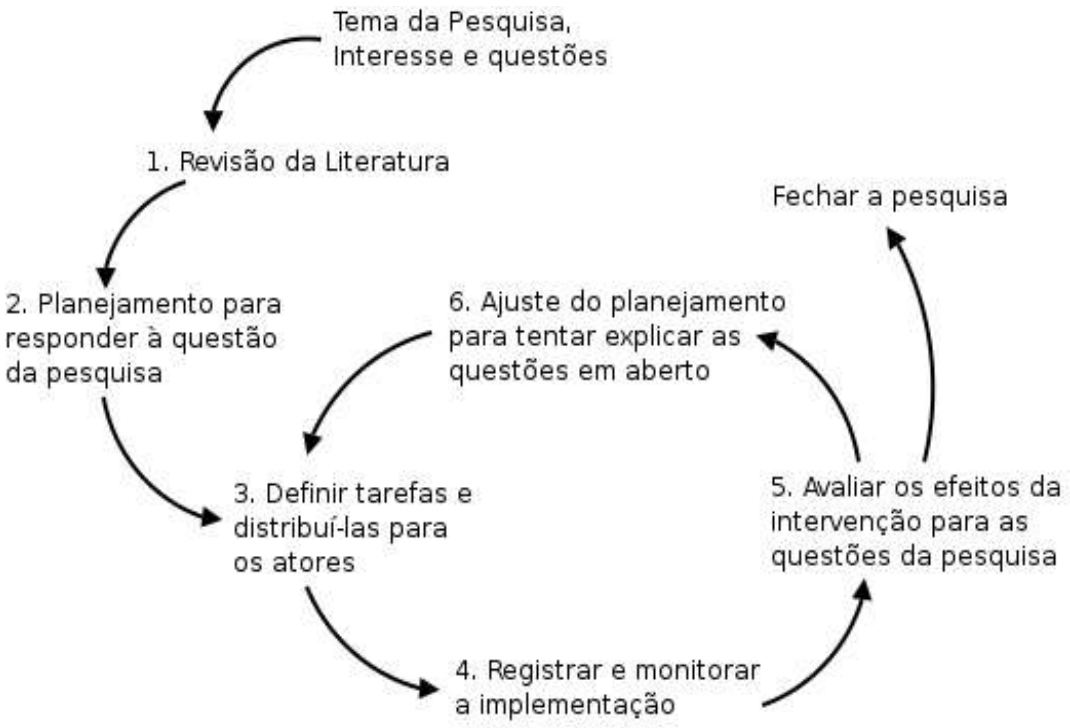


Figura 11: Planejamento Geral da Pesquisa-Ação (adaptado de McKay e Marshall, 1999)

Assim, de posse do conteúdo teórico e com a definição do projeto de DSI a ser estudado, iniciou-se a fase de planejamento onde avaliou-se como a tecnologia de WS poderia ser utilizada para resolver o problema estudado, e que estratégias seriam utilizadas para descrever, com rigor, os passos executados. As seguintes ações foram realizadas nessa fase:

1. A partir do referencial teórico e de entrevistas com outros desenvolvedores que já tinham experiência em WS, foram definidos potenciais fatores críticos de sucesso para o uso da arquitetura de WS por MPEs desenvolvedoras de SI.
2. O planejamento se deu em três etapas:

2.1. O projeto de pesquisa foi explicado detalhadamente para os clientes (JP Escr.Contábil e Cameo Modas) e desenvolvedores e, com o consentimento deles, a pesquisa foi iniciada;

2.2. As ferramentas de desenvolvimento e bibliotecas de classe necessárias para implementar os Web Services foram escolhidas e instaladas.

2.3. O ambiente de teste foi montado na JSK e alguns WS que vinham como exemplos nas bibliotecas de classes foram compilados para verificar se o ambiente poderia ser utilizado corretamente.

2.4. Foram feitos testes locais e na Internet para verificar a conformidade e qualidade dos WS de exemplo segundo os critérios do tópico 2.3 deste trabalho;

2.5. Foram escolhidas as funções a serem implementadas no sistema que poderiam ser utilizadas para validar os fatores críticos de sucesso. Essa escolha foi baseada no equilíbrio entre a utilidade e a complexidade das estruturas de dados e funções requeridas pelos usuários;

3. A partir daí, iniciou-se a análise dos módulos para implementar as funções escolhidas. Vale aqui ressaltar que no desenvolvimento do sistema, utilizou-se uma variação da MRDS (Silveira, 2002) que se baseia em prototipação e análise incremental do sistema.

3.1. Os primeiros WS escolhidos no tópico 6 foram desenvolvidos;

3.2. O processo de desenvolvimento e o software gerado foram avaliados a partir dos critérios de qualidade e produtividade definidos na seção 2.3;

3.3. Foi publicado um site na Internet para divulgar o projeto, convidando outros desenvolvedores a participarem do mesmo. Foram elaborados e disponibilizados pelo pesquisador Tutoriais e HOWTOs[6] como forma de facilitar o aprendizado por parte de outros desenvolvedores;

3.4. Em paralelo, o pesquisador selecionou e contratou dois outros desenvolvedores para participar do processo de DSI;

3.5. Foram disponibilizados WS para que os outros desenvolvedores possam realizar testes;

3.6. O modelo dos WS foi disponibiliado para parceiros (outras empresas especialistas em desenvolvimento de software) e suas sugestões e críticas foram documentadas para que se pudesse reavaliar o planejamento e os resultados do processo.

4. Os WS foram concluídos e avaliou-se os resultados, verificando se os indicadores de qualidade e produtividade do processo de DSI se mostraram melhores em relação à arquiteturas anteriores.

6 HOWTOs: Textos publicados sob forma de tutoriais muito utilizados em projetos de software livre que explicam como realizar as operações cotidianas em um software.

5. Foi verificado se os FCS surgiram no projeto e se outros fatores foram identificados. Diferentemente da figura 11, não se iniciou um novo passo para se realizar novos experimentos.

Ao longo dos passos implementados utilizou-se métodos de coleta de dados próprios a uma metologia interpretativa, incluindo entrevistas com os desenvolvedores, análise dos documentos gerados, mensagens de correio eletrônico (*e-mail*) trocadas entre os participantes da pesquisa e a observação direta e participante. Vale lembrar que Thiollent (1986) define que a observação participante é uma das possíveis técnicas de coleta de dados na pesquisa-ação e que processar a informação e o conhecimento obtido em situações interativas não constitui uma infração contra a ciência social. Segundo Trivinos (1992, p.90), “a observação direta ou participante é obtida por meio do contato direto do pesquisador com o fenômeno observado, para recolher as ações dos atores em seu contexto natural, a partir de suas perspectivas e seus pontos de vista.”

3.5. Limitações da Pesquisa

Em função do método empregado, as conclusões da pesquisa estarão restritas ao contexto em que foi realizada. Assim, não pretende prever resultados ou explicar o desenvolvimento de projetos utilizando Web Services de forma genérica. Espera-se que as conclusões da pesquisa sejam úteis para orientar a atuação de pequenas empresas de desenvolvimento de software, cujos clientes também são, em geral, micro ou pequenas

empresas e especialmente no contexto de projetos de DSI que empreguem ferramentas similares às utilizadas no projeto estudado.

O tipo de conhecimento criado pela pesquisa-ação deriva de um melhor entendimento de um problema complexo. O pesquisador acumula informações sobre uma situação particular e num ambiente específico, para em seguida chegar a conclusões mais gerais a partir das experiências vivenciadas naquele contexto. Embora tais conhecimentos possam ser expressos através de modelos e teorias, é importante ressaltar que o objetivo de uma pesquisa-ação é o entendimento contextualizado de um processo humano complexo, e não a obtenção de uma verdade universal (Baskerville, 1996).

4. RESULTADOS

"O que me interessa não é como são as coisas
mas como as pessoas as percebem."
Epictetus, Filósofo Séc I. DC

Neste capítulo, descreve-se como se deu a pesquisa e quais os resultados obtidos. Como o objetivo deste trabalho é avaliar os fatores críticos de sucesso no processo de desenvolvimento de sistemas de informação que utilizem WS no contexto das MPEs, o primeiro passo foi verificar se era possível utilizar a tecnologia com os recursos financeiros e humanos tipicamente disponíveis para uma MPE. Com este primeiro passo concluído, o próximo passo foi identificar as vantagens e fatores críticos para uma implementação bem sucedida e avaliar se a utilização de WS poderia trazer alguma vantagem competitiva para a empresa desenvolvedora. Tais etapas são descritas, em detalhes, a seguir.

O projeto foi iniciado com 3 pessoas: um analista de sistemas e coordenador do projeto, um programador e uma web-designer. Esperava-se que, numa segunda fase, dois novos desenvolvedores entrassem no projeto. A idéia era, a medida que os primeiros WS fossem sendo construídos e apresentando os primeiros resultados, seria feita uma divulgação do andamento do projeto em grandes canais especializados em software livre e isso poderia atrair novos programadores voluntários (sem remuneração) para participar do projeto. Além

disso, havia orçamento para a contratação de mais um desenvolvedor, caso o projeto se mostrasse viável do ponto de vista tecnológico.

Como não havia nenhum cliente ou organização financiando o projeto, o orçamento mensal era muito pequeno, o que inviabilizava uma equipe trabalhando diariamente. Além disso, todos os participantes estariam envolvidos em outros projetos e não poderiam se dedicar integralmente à pesquisa. Assim, optou-se pelo desenvolvimento à distância no qual cada desenvolvedor trabalharia em casa nos fins de semana ou em horários vagos. Inicialmente, o analista dedicaria 40 horas mensais ao projeto e os desenvolvedores e web-designer, 20 horas cada um.

4.1. Fatores Críticos de Sucesso

Para se definir os fatores críticos de sucesso, buscou-se inicialmente o que artigos e publicações nas áreas de Sistemas de Informação e Administração. Como uma pequena empresa não tem diversos diretores que possam ser entrevistados, foi necessário recorrer à bibliografia sobre DSI e, especificamente, sobre Web Services para que se obtivesse uma relação de potenciais FCS. Como não se encontrou nenhum artigo sobre FCS e Web Services, foi feita uma compilação dos FCS existentes na literatura para o desenvolvimento de sistemas em geral e especificamente para arquiteturas baseadas em Web. Além disso, foram realizadas entrevistas com 5 diretores de pequenas *software-houses* para que eles citassem quais seriam os FCS para o processo de DSI. Para cada entrevistado, eram feitas duas perguntas “Dentre

os fatores abaixo, quais você considera crítico para o processo de desenvolvimento de sistemas de informação?" e "Você acha que existem outros fatores críticos que não estão nessa relação?". O resultado desta análise pode ser visto na tabela abaixo onde se lista os FCS iniciais e como eles foram obtidos:

FCS	Origem
FD - Ferramentas de Desenvolvimento	Entrevista
EQ - Equipe Adequada	Entrevista, Ref. Teórico
FTU - Formalização dos Testes Unitários	Ref. Teórico
DIV - Divulgação dos WS	Ref. Teórico
CI - Controle da Interoperabilidade	Ref. Teórico
MI - Manutenção Incremental	Entrevista

Tabela 1: Fatores Críticos de Sucesso

Parte-se do pressuposto que cada uma destes fatores influenciam diretamente os indicadores de qualidade e custo definidos na seção 2.3, indicadores esses que medem se o processo de DSI estará agregando valor para a empresa. A descrição destes fatores e suas relações com os indicadores são definidas a seguir:

Ferramentas de desenvolvimento adequadas (FD)

Quatro dos entrevistados ressaltaram que a escolha da ferramenta desenvolvimento adequada é fundamental. Ressaltaram que, dependendo do projeto, determinadas ferramentas não poderiam ser utilizadas devido ao seu custo elevado mas, em contrapartida, a escolha de uma ferramenta sem

características mínimas de usabilidade, eficiência e funcionalidade também inviabilizam o projeto.

Equipe adequada de desenvolvedores (EQ)

Os cinco entrevistados disseram que, sem uma equipe adequada e motivada, o processo de DSI não consegue ser realizado. Esse fator também pode ser visto na bibliografia em Reel (1999), Wasmund (1993) e Bohem (1994).

Formalização dos testes unitários (FTU)

Como já foi visto na seção 2.3, os testes unitários têm como objetivo aprimorar a qualidade do software gerado e influenciam diretamente os critérios de confiabilidade pois minimizam os erros que seriam encontrados no SI. Este fator é considerado crítico por Joung (2004) e Wasmund (1993).

Divulgação dos WS (DIV)

Como foi visto no referencial teórico na seção 2.1.4, a publicação e a conseqüente divulgação dos WS nos canais apropriados é parte fundamental da construção de WS como forma de automatizar e de melhorar a qualidade do processo de DSI. Mesmo no caso de WS que serão utilizados apenas em sistemas internos de uma única organização, esta divulgação deveria seguir as padronizações definidas (WSDL e UDDI). Este fator foi considerado crítico em Wasmund (1993), Bohem (1994)

Controle da Interoperabilidade (CI)

Deve-se ter a certeza que os WS estarão dentro dos padrões e que possam ser acessados por qualquer ferramenta. Este controle visa melhorar principalmente os indicadores de portabilidade e manutenabilidade. Este fator foi citado por Booch (2001), W3C (2001), Stal (2002).

Manutenção Incremental (MI)

A implementação deve permitir que se atualize com facilidade um WS sem comprometer o funcionamento do resto do sistema. Este fator influencia os critérios de manutenabilidade e de confiabilidade pois minimiza os impactos de mudanças e confiablidade pois minimiza o tempo em que o sistema tem que ficar indisponível ou “fora do ar”.

Tendo definido os possíveis fatores críticos, o póximo passo foi associar as atividades executadas no processo de DSI (definidas no modelo da figura 5) que influenciariam os FCS. Segundo o método baseado em FCS, como foi visto no capítulo de metodologia, ao se correlacionar como cada atividade influencia cada um dos fatores, pode-se verificar se existem atividades que não são relevantes ou se existem atividades que deveriam ser destacadas no processo. O quadro abaixo mostra como esta correlação foi definida:

Atividades / FCS	***FD***	***EQ***	***FTU***	***DIV***	***CI***	***MI***
4.2. Avaliação das ferramentas adequadas	X	X	X	X	X	X
4.3. Contratação de Recursos Humanos	X	X				
4.4. Adaptar processo de Programação / Codificação	X			X	X	X
4.5. Processo de depuração (*Debugging*)	X		X			
4.6. Identificar oportunidade de reuso	X		X			X
4.7. Automatizar rotinas de testes unitários	X					
4.8. Minimizar atividades de instalação (*Deployment*)	X		X			X
4.9. Estabelecer canais de comunicação		X		X	X	

Tabela 2: Atividades e Fatores Críticos de Sucesso

É muito importante ressaltar que, para cada WS que estava sendo construído, as tarefas 4.4, 4.5, 4.6, 4.7 e 4.8, eram executadas. Diferentemente de metodologias tradicionais em cascata (como foi visto no ref. teórico, na seção 2.3), decidiu-se que cada uma das tarefas seria executada ciclicamente para cada WS. Ao executar parte do processo diversas vezes, procurou-se adquirir conhecimento e aproveitar este conhecimento na construção do módulo seguinte de forma mais produtiva.

Tendo as tarefas listadas, definiu-se também quais seriam os indicadores de qualidade, custo e produtividade (definidos na seção 2.3) que seriam influenciados por cada possível fator crítico de sucesso. Assim, uma segunda tabela foi montada, identificando esta relação:

Fatores Críticos de Sucesso	***F***	***Cf***	***U***	***E***	***M***	***P***	***C***	***R***
FD - Ferramentas de Desenvolvimento	X	X	X	X	X	X	X	X
EQ - Equipe Adequada		X		X	X		X	X
FTU - Formalização dos Testes Unitários		X			X			X
DIV - Divulgação dos WS	X		X				X	
CI - Controle da Interoperabilidade	X					X		
MI - Manutenção Incremental		X	X		X			

Tabela 3: Influência dos Fatores Crítico de Sucesso nos Indicadores de Desempenho

Sendo F – Funcionalidade, Cf – Confiabilidade, U – Usabilidade, E – Eficiência, M – Manutenabilidade, P – Portabilidade, C – Custo e R – Reusabilidade.

A partir da próxima seção, as tarefas efetuadas são descritas, com o destaque para os efeitos no processo de desenvolvimento nos indicadores de desempenho definidos anteriormente e o papel dos FCS na geração destes efeitos.

4.2. Avaliação das ferramentas de desenvolvimento adequadas

O primeiro passo da execução do projeto foi a escolha das ferramentas de desenvolvimento que seriam utilizadas. A primeira escolha foi referente a linguagem de programação. Havia três opções: Object Pascal, C++ e Java. A decisão recaiu sobre o Object Pascal uma vez que os desenvolvedores que iriam participar do projeto já tinham muita experiência com a ferramenta, e a adoção de uma outra linguagem iria implicar em um pesado treinamento ou na contratação de outros profissionais.

Mesmo sabendo que os pacotes de desenvolvimento proprietários seriam de um valor elevado, uma pesquisa de preços foi feita em Novembro de 2003 para as ferramentas Delphi e Kylix. Todos os preços encontrados ficaram acima de R$ 5.000,00 o que tornaria o desenvolvimento inviável. Isso porque era necessário adquirir pelo menos 4 licenças de uso e o valor total iria ultrapassar o orçamento do projeto.

Assim, partiu-se para a avaliação de uma ferramenta baseada em software livre ou, então, grátis. Existiam duas ferramentas baseadas em Pascal a serem testadas: o FreePascal desenvolvido livremente por programadores localizados, em sua grande maioria, na Europa e o Kylix Open Edition, desenvolvido pela Borland Corporation.

As duas ferramentas eram baseadas no paradigma de orientação a objetos e utilizam o conceito de componentização e de bibliotecas de classes (descritos na seção 2.1.2). Ou seja, ambas vinham com poucos recursos e era tarefa dos desenvolvedores incorporar novas funcionalidades através de bibliotecas de componentes disponíveis no mercado. Assim, era necessário pesquisar quais das duas ferramentas tinham disponíveis bibliotecas que dessem suporte a WS. Após pesquisa na Internet e nos grupos de usuários em Delphi/Kylix a biblioteca de classes **IndySOAP** baseada em software livre foi encontrada. Ela tinha sido desenvolvida por uma dupla de programadores experientes e com muito boa reputação no mercado Delphi/Kylix segundo várias revistas e sites especializados em Pascal (SwissDC, 2005). Assim, o Kylix 3 Open Edition foi escolhido.

O próximo passo, então, foi escolher a distribuição do Linux que mais se adequava ao desenvolvimento dos Web Services com o Kylix. Apesar da Borland, criadora do Kylix, indicar algumas distribuições, a equipe de desenvolvimento percebeu que existiam distribuições muito mais recentes que poderiam trazer um melhor aproveitamento do sistema operacional, e melhorar a usabilidade e confiabilidade do DSI. Após alguns testes de performance e usabilidade, escolheu-se o Mandrake Linux como distribuição padrão para o projeto, uma vez que este tinha uma interface visual de melhor usabilidade que as outras distribuições e era mais fácil de ser instalado e configurado pelos desenvolvedores.

Esta fase do projeto se mostrou mais difícil do que se esperava, pois houve várias incompatibilidades entre o Kylix e a distribuição do Linux escolhida. Após três semanas procurando soluções na Internet e em listas de discussões específicas, a equipe conseguiu contornar os problemas de compatibilidade que surgiram. Tendo em vista que outros programadores poderiam passar pelo mesmo problema, a equipe decidiu montar um tutorial na Internet, explicando, passo a passo como instalar o Kylix no Mandrake 10. O tutorial foi bem recebido pela comunidade que desenvolve ou pretende desenvolver aplicações em Kylix, tendo sido consultado por desenvolvedores de vários países, desde os Estados Unidos até o Zaire, mesmo estando disponível apenas em português. Atualmente, o tutorial pode ser acessado na página da web http://www.jsk.com.br/kylix-mandrake.html. Esta e outras atividades de divulgação serão mais detalhadas no tópico 4.9.

O próximo passo foi testar a ferramenta de desenvolvimento e definir como se daria o desenvolvimento dos primeiros protótipos de Web Services. A principal tarefa foi obter na Internet ou construir um programa que fizesse a interface com o servidor Web Apache. O servidor Apache, como foi visto na seção 2.6, é responsável por fornecer as páginas de Internet aos usuários externos e é peça essencial para o funcionamento de uma arquitetura baseada na Web.

As buscas na Internet revelaram uma biblioteca livre chamada WebProvider, desenvolvida na República Tcheca, que funcionava apenas para Delphi e Windows. No entanto, o analista conseguiu converter a biblioteca para o Kylix. Esta atividade exigiu em torno de 1 mês de trabalho. A nova biblioteca foi publicada na Internet para que a comunidade soubesse da conversão e pudesse utilizá-la livremente. O objetivo desta publicação era cativar outros programadores para que eles testassem a biblioteca. Entretanto, poucos desenvolvedores se interessaram pela biblioteca.

Naquele momento, os membros da equipe começaram a perceber que não havia desenvolvedores suficientes que estavam utilizando o Kylix. A comunidade de desenvolvedores não era muito ativa e a própria empresa que desenvolveu o Kylix não lançava uma nova versão ou atualizações há mais de um ano. Isso pode ser constatado nas conversas em listas de discussão específicas de Kylix e com os desenvolvedores do projeto tais como:

> “Tenho um sistema comercial inteiro escrito em kylix (70.000 linhas de código), faz controle de estoque+faturamento+contas a pagar/receber+etc ... e estou pensando

> muito seriamente em abandonar o kylix pela falta de continuidade da borland. Hoje estou preso a uma versao Conectiva-8 (creia !!) para desenvolvimento, porque o bendito não roda em outras versões."
>
> *Me preocupa o fato da Borland ter deixado de lado o Kylix, o que significa que se comercialmente nao for mais interessante para ela a continuidade do projeto .Net ela deixa de trabalhar nele?"

Mesmo com vários destes relatos em várias listas de discussão, decidiu-se continuar a usar a ferramenta no projeto, pois a troca àquela altura iria gerar atrasos consideráveis e poderia até mesmo inviabilizar o projeto.

Novas barreiras técnicas continuaram a surgir ao longo dos testes. Os primeiros testes mostraram que a biblioteca IndySOAP, responsável pela geração dos Web Services, continha vários erros e que ela não tinha sido testada integralmente no ambiente Linux pelos desenvolvedores da biblioteca. Mesmo com o apoio da comunidade de desenvolvedores Kylix e dos próprios desenvolvedores da biblioteca, foi necessário investir 2 meses no mapeamento e correção dos erros para que a biblioteca ficasse apta para uso no Kylix.

Na verdade, os autores originais da biblioteca tinha parado de desenvolvê-la e a própria equipe do projeto teve que alterar o código-fonte original para ajustar a biblioteca para uso e, apesar do código estar disponível e muito bem documentado, os ajustes eram difíceis devido à complexidade da biblioteca. Mesmo após várias correções, a biblioteca IndySOAP ainda apresentava vários problemas. Para guarantir o mínimo de confiabilidade das bibliotecas WebProvider e IndySOAP, foram criados vários exemplos e testes unitários que foram incorporados às bibliotecas.

Um outro componente fundamental foi o SGBD PostgreSQL. Ele foi utilizado para o armazenamento dos dados que eram registrados pelos usuários dos WS. A instalação do PostgreSQL transcorreu sem problemas e os desenvolvedores aprenderam a utilizá-lo com muita rapidez. Nesta etapa, a estrutura de dados que seria utilizada pelo sistema foi criada tendo por base a modelagem feita na etapa de planejamento.

Outra ferramenta de apoio que se mostrou muito estável foi a ferramenta CASE de modelagem UML Jude-take. Este software é desenvolvido no Japão e apesar da falta de documentação na Internet, mostrou-se muito fácil de utilizar. A única deficiência identificada no aplicativo é que não gerava código em Pascal e nem continha interfaces que se pudesse interagir com o modelo de classses, ou seja, não era possível construir rotinas automatizadas que gerassem o código em Pascal. Se este recurso estivesse presente, o processo de criação dos WS seria mais produtivo e traria maior rapidez aos desenvolvedores do projeto pois evitaria a digitação da definição das classes.

A instalação, os testes e os ajustes de todas as bibliotecas e softwares que foram utilizadas no projeto duraram 4 meses que foi bem mais do que o esperado. O modelo geral pode ser visto na figura 11.

Nesta etapa também foram feitos testes de interoperabilidade com o intuito de verificar se a biblioteca era compatível com outros ambientes de desenvolvimento. Este passo era fundamental para se certificar que os WS gerados iriam ser realmente compatíveis com outros sistemas operacionais e ambientes de desenvolvimento pois, sem atingir esta

compatibilidade, os WS não poderiam ser executados por outros softwares, o que iria contra o próprio conceito da tecnologia. Os testes foram feitos inicialmente com Delphi, dotNet e XUL. Delphi é uma linguagem também desenvolvida em Pascal para o ambiente Windows. DotNet é o ambiente mais recente desenvolvido pela Microsoft e XUL é o ambiente de desenvolvimento criado pela Mozilla Fundation e utilizada pelos browsers Netscape, Mozilla e Firefox. Nos três ambientes, os testes foram bem sucedidos.

Infra-estrutura na Internet

Tendo-se decidido pelo ambiente de desenvolvimento, a próxima fase do projeto foi encontrar um provedor na Internet onde se pudesse realizar os primeiros testes. Devido à linguagem adotada, o sistema operacional do servidor disponibilizado pelo provedor deveria ser Linux. Além disso, o uso do Linux apresentava duas vantagens claras; (1) o servidor Apache trabalha de forma mais otimizada no Linux do que no Windows; (2) como a maioria dos projetos Web são desenvolvidos em Linux, haveria mais material disponível na Internet para consulta e aprendizado.

Foram escolhidos três provedores que tinham o menor custo de serviço de hospedagem. Dentro os que tinham suporte à utilização de CGIs (módulos executáveis que rodam no servidor Web) feitos em Kylix. Após uma negociação breve se optou por um provedor que se mostrou mais comprometido em auxiliar à equipe com as configurações customizáveis que eram necessárias ao ambiente utilizados no projeto.

Uma dificuldade encontrada nesta etapa envolvia o *deployment*, ou seja a fase de envio dos módulos executáveis para o servidor web. Apesar da área de suporte técnico do provedor ter sido muito prestativa, esta etapa também durou além do esperado, pois a versão do Linux instalado no servidor não estava preparado para rodar os CGIs criados pelo Kylix. Assim, o provedor teve de fazer um atualização no sistema operacional para suportar os CGIs. Após a atualização, o servidor funcionou sem maiores problemas.

Em resumo, pode-se analisar se os indicadores de desempenho associados a esta atividade tinham sofrido alguma melhora. O único que se mostrou claramente problemático foi o indicador de Usabilidade devido, em grande parte, ao desconhecimento do sistema operacional por parte dos desenvolvedores. Entretanto, esperava-se que este problema fosse superado a medida em que os desenvolvedores fossem se acostumando ao novo ambiente e que novas versões dos softwares utilizados fossem disponibilizadas.

4.3. Contratação de Recursos Humanos

No início do projeto, apenas um programador foi contratado para avaliar as ferramentas e realizar os testes iniciais. Ele foi contratado como free-lancer e trabalhava nas suas horas vagas. O coordenador do projeto passava as tarefas que deveriam ser realizadas mas não definia quantas horas elas iriam tomar. A remuneração era feita a medida em que as tarefas eram realizadas e ao final de cada tarefa, o programador entregava uma planilha com as horas gastas.

Antes da fase de *deployment* ter sido iniciada, o programador contratado avisou que não iria mais participar do projeto pois não estava sem tempo para se dedicar. Na verdade, em conversas informais com ele, pôde-se perceber claramente que ele se sentiu desmotivado por conta dos erros das bibliotecas relatados na seção anterior e por perceber que a ferramenta (Kylix) não teria muito futuro.

Enquanto se buscava outro desenvolvedor, o coordenador criou várias ferramentas de apoio para agilizar o reuso de código e a automatização do envio dos primeiros WS para o provedor, com o intuito de demonstrar a outros desenvolvedores quais eram os potenciais da tecnologia, cativando assim uma maior audiência para o projeto.

Assim, foi possível demonstrar os primeiros serviços para outros desenvolvedores e inciar os primeiros testes de confiabilidade, performance e interoperabilidade. Os resultados desses testes foram satisfatórios e muito promissores. Vale ressaltar que os testes de performance e estresse geraram resultados positivos, apesar do servidor web não ser muio potente.

Conforme o plano de pesquisa, foi construída uma página na Internet para divulgar o projeto e convidar outros desenvolvedores a participarem. Entretanto, poucos desenvolvedores se interessaram em participar sem remuneração. A JSK então fez uma apresentação do projeto e uma oferta de emprego nas listas de discussões especializadas na linguagem Kylix e Delphi com um valor acima do mercado e com a possibilidade do desenvolvedor trabalhar em casa e com prazos mais dilatados para a entrega dos programas.

Aproximadamente 15 programadores se candidataram à vaga. Como não havia um processo de seleção formal e muitos dos candidatos não residiam no Rio de Janeiro, o critério de seleção foi um teste para avaliar a capacidade do desenvolvedor em instalar o Kylix e as bibliotecas de classes a partir do tutorial que foi publicado no site. Esta forma de seleção foi adotada pois o coordenador do projeto tinha que ter certeza que o profissional contratado teria o mínimo de experiência e capacitação tanto com o Kylix quanto com o Linux.

Somente dois desenvolvedores conseguiram completar o teste e foram contratados para criar novas rotinas. Inicialmente, as tarefas passadas eram apenas para consumir os WS, ou seja, interagir com os WS que já estavam prontos para disponibilizar o *front-end* (telas e relatórios)para os usuários finais (a JP Escritório Contábil e a Cameo Modas). Entretanto, antes de entregar as primeiras tarefas, um dos desenvolvedores desistiu do projeto por ter se envolvido em outras atividades. Nesse ponto, a equipe começou a se perguntar por que os desenvolvedores não estavam motivados para participar do projeto, já que seriam remunerados. Assim, os desenvolvedores que não conseguiram terminar a instalação do ambiente de desenvolvimento foram novamente contactados e precebeu-se que o processo de implantação e utilização do Kylix e do Linux era a principal barreira conforme pode-se ver em alguns comentários feitos pelos desenvolvedores quando se perguntou porque eles não quiseram seguir no projeto:

> “Não que nao tenha tido vontade de continuar, pelo contrário, havia sim um grande interesse, mas como respondi na primeira questao, nao consegui fazer a instalação do pacote soap no kilyx”.
>
> “Creio que o maior complicador foi o tempo disponível, principalmente para

> configuração e testes do Kylix. Posteriomente você colocou um tutorial melhor, mas eu tive um final de ano complicado que inviabilizou a continuidade dos testes"

A grande maioria dos desenvolvedores não conseguia sequer instalar o ambiente Kylix e, mesmo depois da instalação, a ferramenta apresentava problemas e incompatibilidades com determinadas distribuições (versões do Linux) utilizadas. Ficou claro que estes fatores desmotivavam os desenvolvedores uma vez que ele não acreditavam ser possível obter os índices de produtividade que estavam acostumado no ambiente Windows. Além disso, alguns deles estavam abandonando o Object Pascal para utilizar Java ou C#, pois consideravam que a "linguagem não tinha futuro".

Devido à falta de desenvolvedores, o pesquisador alterou o planejamento inicial do projeto buscando programadores de outras linguagens para não atrasar mais ainda o prazo de entrega do SI. A utilização de outras linguagens estava prevista apenas para o final do projeto como forma de testar a interoperabilidade dos WS. Devido ao problema de mão de obra, optou-se por desenvolver o *front-end* da aplicação em outras linguagens. Assim, foram feitos convites para listas de discussão em Java, PHP e XUL. A comunidade que mais deu retorno foi a de PHP, enviando mais de 40 currículos. Apenas dois foram selecionados para fazer o teste com os Web Services, pois já tinham trabalhado com a arquitetura.

Um outro fator interessante foi a entrada de desenvolvedores não remunerados. Durante as apresentações sobre o projeto que foram feitas nas listas de discussão, quatro programadores se ofereceram para trabalhar sem remuneração. A esses voluntários, foram passadas tarefas de teste das rotinas e algumas rotinas periféricas que não requeriam um alto

grau de coordenação com os outros membros da equipe. A motivação dos voluntários, de acordo com entrevistas realizadas pelo pesquisador, era a possibilidade de aprender Kylix, Web Services ou Linux. Apesar de não terem muitas horas para se dedicarem ao projeto, o resultado do trabalho destes programadores se mostrou muito produtivo apesar de ter sido direcionado a tarefas que não eram essenciais ao desenvolvimento.

Uma *designer* especializada em Web também foi contratado em regime *free-lancer* para começar a estruturar como seria o front-end para o usuário, ou seja, as telas e relatórios com os quais o usuário final iria interagir. O coordenador já conhecia a *web designer* de outros projetos e ela foi a única pessoa que trabalhou do início ao fim do desenvolvimento do SI.

Apesar dos inúmeros problemas de gerenciamento de recursos humanos, foi possível seguir com o projeto. Os módulos básicos foram eventualmente concluídos, apesar de inúmeras correções no cronograma. Ao final do projeto, de um total de quatorze desenvolvedores, apenas oito entregaram códigos que puderam ser aproveitados no SI.

4.4. Adaptar o processo de programação/codificação

Desde a etapa de planejamento, todos os desenvolvedores tinham ciência que o processo de codificação seria diferente do usualmente feito em arquiteturas baseadas em Cliente/Servidor. A própria estrutura dos programas e módulos executáveis seria diferente, baseada em CGI e na divisão entre WS e a interface visual. Assim, revisar e adaptar a forma

de programar parecia fundamental para o desenvolvimento do SI.

Os módulos escolhidos para serem codificados foram o de Sessão, Usuários, Sócios, Empresas, Clientes, Cadastro de Tipos de Notas Fiscais, Cadastro de Centro de Custos, Cadastro de Despesas e Notas Fiscais. A modelagem realizada na etapa de planejamento, procurou identificar cada classe como um serviço, sendo cada serviço um módulo independente (um módulo executável CGI).

A primeira classe a ser desenvolvida foi a **Sessão**. Ela representava a sessão, isto é, a indicação de que um usuário se conectou ao sistema. Esta classe, ou serviço, seria responsável por todo o controle de acesso e autenticação dos demais serviços. Para cada usuário que se registrasse no sistema, seria gerado um SID (*Session IDentification*) que seria válido durante o tempo em que um dado usuário estivesse utilizando os recursos do sistema.

Essa estrutura de dados baseada em SID era prevista na própria especificação do protocolo SOAP. Contudo o projetista, após consulta na Internet, verificou que muitas implementações do protocolo não haviam incluído essa especificação. Assim, optou-se por incluir o SID como um parâmetro adicional em todas as chamadas aos WS. Desta forma, o mecanismo de controle ficaria aparente na própria descrição dos métodos dos WS, o que poderia facilitar o entendimento do próprio mecanismo de autenticação por parte dos desenvolvedores que fossem utilizar o WS.

O WS.Session demorou aproximadamente um mês para ficar pronto consumindo 24 homens/hora. Sua definição e estrutura podem ser visualizadas em http://www.jsk.com.br/cgi-

bin/Session.cgi/wsdl/ISession.

Com o serviço de controle de sessão implementado, o próximo passo foi construir o WS que gerenciava os usuários que teriam acesso ao sistema. Com esse WS, seria possível cadastrar e atualizar os dados dos usuários. O WS de sessão era apenas de leitura, ou seja, ele não gravava dados enviados pelo cliente. Já com este novo WS, foi possível testar a gravação de dados nos WS e foi possível também avaliar o acesso ao banco de dados PostgreSQL em operações de gravação intensas. Também foi possível testar questões relativas a segurança e a acesso que ainda não tinham sido contempladas no módulo Session como por exemplo: que tipo de usuário pode criar outros usuários ? Um usuário deve poder alterar seus dados tais como email, senha, etc ?

Neste ponto, percebeu-se que este mesmo controle de usuários que estava sendo implementado no projeto GOPE, poderia ser utilizado com facilidade por qualquer outro futuro projeto ou poderia substituir o controle de acesso de um sistemas já pronto, minimizando assim o volume de código a ser escrito e a ser mantido. Isso se devia ao baixíssimo acoplamento que o WS tinha para ser utilizado. Essa conclusão demonstrou que, a longo prazo, o baixo acoplamento e o reuso do código (com consequente aumento de produtividade) poderia ser feito em muito maior escala que o anteriormente feito com outras arquiteturas o que traria bons resultados para a empresa.

Após a implementação do serviço de gerenciamento de usuários, deu-se início aos primeiros WS que continham a lógica do processo de negócio específico do sistema GOPE.

Para cada WS construído, um programa ad-hoc era gerado para testar cada um dos métodos da classe. Neste ponto, a construção dos módulos se mostrou muito produtiva, pois a própria arquitetura induzia e facilitava o programador a realizar os testes. Não havia nenhuma configuração extra a ser feita ou qualquer outra interferência entre o WS e o programa de teste. Isso facilitou a programação enormemente e os testes integrados dos quatro primeiros WS foram feito com muita agilidade.

Inúmeros problemas foram encontrados ainda nesta fase ainda associados às bibliotecas Webprovider e IndySOAP. Como os criadores dessas bibliotecas não estavam mais provendo suporte e atualizando os softwares, a própria equipe da JSK tinha que descobrir, corrigir e contornar os erros que surgiam ao longo do desenvolvimento. Percebeu-se, então, que o projeto original, tinha se desmembrado em três: o projeto GOPE, o projeto WebProvider e o projeto IndySOAP. Foi necessário deslocar os poucos recursos disponíveis para modificar e testar as bibliotecas para somente então continuar o projeto principal. Apesar do atraso ter sido muito desgastante para a equipe, houve pontos positivos: como os próprios desenvolvedores tiveram que editar o código fonte das bibliotecas, eles ficaram muito mais familiarizados com as funcionalidades dessas bibliotecas e ao final do processo, aumentaram a sua produtividade nas tarefas de programação.

Também nessa fase, ficou evidente que a montagem dos WS utilizando as ferramentas de programação tinha que ser bastante meticulosa, envolvendo uma variedade de parâmetros e procedimentos que tinham que ser memorizados e repetidos pelos

desenvolvedores. Decidiu-se, então, montar um *wizard* para automatizar as tarefas repetitivas. O *wizard* é um programa que auxilia na geração de tarefas repetitivas através de uma interface amigável, que vai instruindo o usuário sobre quais os passo que devem ser seguidos. O *wizard* levou uma semana para ficar pronto, mas agilizou e minimizou vários erros que estavam sendo cometidos ainda na fase de implementação.

Uma vez que os principais WS estavam prontos, os desenvolvedores da JSK começaram a participar mais ativamente na construção de uma interface gráfica para se testar a funcionalidade dos WS numa aplicação corriqueira de cadastro. Para tal, buscou-se um desenvolvedor externo que não estivesse familiarizado com os WS para que se pudesse analisar como seria a interação de um outro desenvolvedor com os WS. O desenvolvedor escolhido foi um dos voluntários que trabalhavam sem remuneração.

O objetivo era avaliar se o possível fator de sucesso DIV (Divulgação do WS) era realmente crítico para o processo. Ao avaliar quais as dificuldades que esse desenvolvedor teria para utilizar o WS, e verificando se o WSDL seria suficiente para que o desenvolvedor pudesse entender como trabalhar com o serviço, conforme dizia a literatura (Iyer et al, 2003), seria possível interpretar se esse fator beneficiaria o processo. Se o WSDL não fosse suficiente para um desenvolvedor externo e, se a empresa que disponibilizasse o WS tivesse que explicá-lo através de outros tipos de documento, uma das vantagens descritas na literatura não seria comprovada pois o processo deixaria de ser automático.

A solicitação passada para o programador forçava-o propositalmente a utilizar

somente a definição WSDL. Um exemplo foi incluído abaixo para que se exemplifique a simplicidade do processo:

> “Preciso que vc faça uma tela onde se pede usuario/senha e o programa acessa o WS.Session para validar se o login está correto. O detalhamento do WS pode ser visto em http://www.jsk.com.br/cgi-bin/Session.cgi/wsdl. Para vc se logar, utilize o usuario/senha: LAZARO/lazaro05”

Mesmo com poucas instruções sobre como realizar a tarefa, o programador conseguiu consultar as funções do sistema, sem necessitar de auxílio de nenhuma documentação extra, exceto o endereço do WSDL. Com isso, verificou-se que não seria necessário gerar uma documentação extensa explicando a interface do WS (como seria necessário em outras APIs) já que o próprio protocolo WSDL era suficiente para um desenvolvedor. Ou seja, os indicadores de usabilidade e de produtividade se mostravam claramente melhores que os encontrados em outras arquiteturas no tocante à divulgação das interfaces (API) para os desenvolvedores.

Devido à falta de programadores em Kylix, iniciou-se a tentativa de construir os front-ends em outras linguagens utilizando os WS que já estavam prontos. A utilização de outras linguagens estava prevista apenas para a fase inicial de testes de interoperabilidade, mas como havia muita dificuldade na obtenção de mão de obra, o coordenador do projeto decidiu experimentar outras linguagens que funcionassem no Linux.

A primeira tentativa foi com a linguagem PHP. O primeiro problema encontrado nesta linguagem foi na interpretação das requisições SOAP dos WS por parte do PHP. Sempre que o PHP tentava se comunicar com o WS, a biblioteca IndySOAP não reconhecia a

requisição, retornando mensagens de erro. Através das ferramentas de depuração, conseguiu-se finalmente descobrir que o problema estava no PHP que não montava as requisições corretamente.

Após algumas correções feitas, o programador contratado conseguiu acessar os WS com sucesso. Entretanto quando tentou-se fazer a instalação dos programas que ele desenvolveu no ambiente de DSI da empresa, verificou-se que a versão do PHP que era incompatível com a instalada. Segundo o programador contratado, somente a versão 5 do PHP tinha suporte ao WS. Em função desse problema, os testes de PHP não foram em frente.

Em Java, a situação foi pior pois era necessário instalar o software Tomcat e várias bibliotecas extras, o que tomaria muito tempo da equipe. O Tomcat era necessário para enviar, receber e gerenciar as requisições HTTP necessárias à utilização dos WS e as bibliotecas eram necessárias para dar implementar o protocolo SOAP. Além disso, para dificultar ainda mais o processo, o servidor da Internet teria que passar por várias adaptações razoavelmente complexas. Por essas razões, o desenvolvimento do Java não foi adiante.

A linguagem que se mostrou muito fácil de ser implementada foi XUL. XUL é uma linguagem que tem partes em javascript e partes em XML utilizada pelo projeto Mozilla e é executada nos browsers que se valem desta tecnologia. Com uma modificação muito simples no servidor Web, os programas rodaram sem problemas e a programação se mostrou muito fácil. A equipe XUL foi formada por 2 programadores, um remunerado e outro voluntário. As telas foram feitas dentro do prazo previsto e podem ser acessadas em http://www.jsk.com.br/xul/login.xul

4.5. Processo de depuração (Debugging)

O processo de depuração ocorreu em duas etapas. A primeira etapa ocorreu no momento de desenvolvimento, quando o programador precisava acompanhar como o programa estava sendo executado para, em seguida, intervir na execução do código para corrigí-lo ou otimizá-lo. A segunda etapa ocorreu durante a execução do sistema, ou seja, quando ele já estava em funcionamento e detectava-se algum problema. Neste caso, o desenvolvedor devia ter condições de obter detalhes mais específicos sobre o erro e em que condições ele ocorreu. Quanto mais eficiente fosse o processo de depuração, mais rápido o desenvolvedor conseguiria analisar o programa e corrigí-lo.

A preparação da depuração foi algo que trouxe muitos problemas para desenvolvimento desde o início. Em primeiro lugar, o fato de um Web Service ser executado remotamente tornava difícil a utilização de ferramentas clássicas de depuração que eram feitas

para analisar programas executados localmente.

Assim, foi necessário que se criasse uma estrutura que gerasse pacotes SOAP manualmente que pudessem servir de entrada para o utilitário de depuração. Desta forma, a equipe teve que alterar a biblioteca WebProvider para que ela simulasse a execução do servidor Web para que, então, o depurador pudesse ser executado.

Nessa etapa do projeto, começou-se a notar vantagens na escolha da linguagem, e, principalmente, da arquitetura baseada em software livre. Como a biblioteca WebProvider já tinha sido esmiuçada pela equipe para que ela fosse adaptada ao ambiente Kylix, quando surgiu a necessidade de incluir novas funcionalidades nela, o processo de desenvolvimento foi bem mais rápido. Em apenas dois dias, o projetista conseguiu implementar a funcionalidade de depuração. Percebeu-se então que, se os softwares utilizados fossem proprietários, o projeto poderia ter sofrido um revés considerável pois o custo para depurar o sistema poderia crescer enormemente e depender-se-ia do fornecedor para incluir as novas funcionalidades que fossem demandadas.

Com a mesma solução da depuração desenvolvida, foi possível também implementar um histórico das operações efetuadas para cada WS, coletando os pacotes SOAP recebidos e enviados para futuras análises e diagnósticos. Esses históricos, chamado de *log* no jargão dos desenvolvedores, se mostrou muito útil, pois permitia analisar processamentos e chamadas aos WS após terem ocorrido, ou seja, de forma assíncrona.

A verdade é que própria tecnologia de WS baseada no protocolo SOAP facilitou

muito a implementação desse histórico. Como o resultado dos *logs* eram arquivos textos facilmente legíveis, a interpretação deles foi facilitada enormemente. Podia-se, inclusive, enviar os *logs* para que outros desenvolvedores depurassem os erros em seus próprios ambientes.

4.6. Identificar oportunidades de reuso

A preocupação com a reusabilidade foi uma constante durante todo o projeto. Conforme visto nas referências bibliográficas (Banker e Kauffman, 1991; Basili, Briand e Melo, 1996; Chen e Lee, 1993 apud Rothenberger, 1998), a reusabilidade era um dos fatores que mais aumentava a produtividade no DSI. Assim, era fundamental verificar se a utilização de WS poderia se beneficiar de técnicas de reuso.

A principal dúvida que se tinha no início do projeto era se os WS teriam os mesmos recursos de reusabilidade que a linguagem utilizada para desenvolvê-lo continha. Como o Kylix é baseado na linguagem Object Pascal e se fundamenta nos três principios de linguagens orientadas a objetos – herança, polimorfismo e encapsulamento - esperava-se que os WS pudessem se beneficiar das mesmas vantagens que a linguagem oferecia, principalmente no tocante à herança e ao polimorfismo.

Assim, durante o planejamento da pesquisa, dois WS foram eleitos para que se tentasse validar o conceito de herança: o WS.Empresa e o WS.Cliente. Baseado Na modelagem de classes que tinha sido feita antes do início do projeto, constatou-se que um

Cliente era sempre uma Empresa, ou seja, continha todos os atributos de uma empresa, além de atributos específicos de um cliente. Assim, fez-se primeiro a construção do WS.Empresa e depois verificou-se a possibilidade de se herdar as propriedades do primeiro com o WS.Cliente, aproveitando todo o código já havia sido escrito para o primeiro. O experimento foi bem sucedido, motivando bastante os desenvolvedores. O resultado demonstrava o potencial de reuso de código das ferramentas utilizadas com base na arquitetura de Web Services.

Além disso, mensalmente, os desenvolvedores eram chamados pelo coordenador do projeto para participarem de debates via email com o objetivo de discutir oportunidades de alterar ou criar novas classes com o objetivo de aumentar o reuso do código. O resultado desses debates se mostrou muito proveitoso pois foram realizados vários ajustes nas bibliotecas IndySOAP e WebProvider com o objetivo de automatizar o processo de criação de WS. Com isso, foi possível agilizar o trabalho dos desenvolvedores e aumentar a confiabilidade do software. Os desenvolvedores perceberam que estas mudanças diminuiram o número de falhas durantes os testes pois, com a automatização, o desenvolvedor preenchia menos parâmetros, minimizando, assim, a possibilidade de erros.

A capacidade de herança dos WS facilitou muito o processo de DSI, tornando-o mais mais produtivo. Sempre que um desenvolvedor realiza melhorias ou acrescentava novas funcionalidades às classes primárias (que eram “pais” do web services finais), estas novas funcionalidades eram “herdadas” por todos os WS que já estavam prontos sem que fosse

preciso alterar uma linha de código sequer.

Uma outra técnica denominada *refactoring* foi experimentada e amplamente utilizada durante o desenvolvimento dos WS. Com o *refactoring*, ou em português, refatoração, procura-se aumentar a eficiência e a confiabilidade do software sem alterar a sua funcionalidade. Segundo Martin Fowler (*apud* Teles, 2004, pg. 238), essa técnica é utilizada durante todo o ciclo de vida do desenvolvimento e "tem como objetivo alterar o software de modo a não alterar o comportamento externo do código, mas apenas melhorar a sua estrutura interna. É uma forma diciplinada de limpar o código de modo a minimizar a inserção de defeitos no sistema".

Assim, com o objetivo de melhorar a eficiência e a manutenabilidade dos WS, o planejamento previa a análise de como a confecção dos módulos de Web Services se comportava com a refatoração e se haveria algum problema ou impecilho. A aplicação dessa técnica também foi bem sucedida devido às características inerentes de modularidade dos WS, que facilitavam os testes após cada refatoração. Os primeiros módulos (Session, Empresa e TipoNF) foram "refatorados" pelo menos 3 vezes cada um. Em cada passo, foi possível otimizar o código e reutilizar funcionalidades que se mostravam comuns a todos os módulos. Com isso, a codificação de novos WS se tornava mais rápida, pois mais funcionalidades eram "herdadas" dos códigos já prontos. O resultado final era uma maior produtividade por parte dos desenvolvedores, que conseguiam implantar mais funcionalidades, escrevendo menos código.

Ainda nesta atividade, a manutenabilidade era um outro indicador que se pretendia avaliar. Para analisá-la, verificou-se como o reuso de código facilitava ou dificultava a manutenção dos WS. Neste ponto, o resultado foi considerado neutro, ou seja, com pontos positivos e negativos. O ponto negativo residia no *deployment* pois sempre que fosse preciso alterar uma classe ou WS pai, era necessário recompilar também todas as classes herdadas. Como isso, o desenvolvedor tinha que controlar manualmente quais WS tinham que ser recompilados e retransmití-los para o servidor Web. Em experiências com outras arquiteturas, a recompilação não era necessária e esse fato foi um ponto negativo para o indicador de modificabilidade. Em contrapartida, a testabilidade era facilitada pois, no caso de alteração de uma classe pai, era possível compilar e testar apenas uma das classes filhas, mantendo as outras funcionando sem a alteração realizada.

4.7. Automatizar rotinas de Testes unitários

A idéia inicial para esta fase era construir ferramentas e wizards que automatizassem o teste com os WS já que este fator é citado na literatura como benéfico para o desenvolvimento de componentes em SI (Teles, 2004). Entretanto, como não haviam ferramentas disponíveis que testassem os WS de forma automática e devido ao atraso decorrente das outras atividades do projeto, optou-se por construir programas customizados para cada um dos WS sem que se buscasse uma estrutura automatizada e genérica.

O processo era simples: para cada WS, escrevia-se um teste unitário que iria chamar

todas as funções que o WS disponibilizava através da sua interface. Com isso, a cada alteração ou refatoração feita, se executava novamente o programa de teste para revalidar o WS.

Mesmo sem o automatismo, pôde-se perceber que os testes eram muito fáceis de serem construídos. A modularidade e a interface bem definida inerente dos WS, facilitavam a construção de programas que cobrissem grande parte das funcionalidades de cada serviço.

Os testes unitários conseguiram detectar vários erros durante a fase de projeto. Para efeito de comparação, dois WS (TipoNF e CCusto) foram deliberadamente construidos sem a utilização de testes unitários, e foram os que apresentaram os maiores índices de falhas. Quando os dois serviços foram submetidos ao processo de teste unitário, diversos erros foram encontrados e corrigidos. Os testes unitários também foram fundamentais para facilitar a refatoração dos módulos pois, os testes eram realizados a cada modificação e esse fato aumentava a confiabilidade dos WS gerados.

4.8. Minimizar atividades de Instalação (Deployment)

Em projetos anteriores da equipe, baseados na arquitetura cliente-servidor, o processo de instalação e correção de módulos sempre se mostrou problemático, sendo considerado uma das grandes deficiências daquela arquitetura. A cada alteração que era feita, era necesário copiar os módulos executáveis para o cliente. Muitas vezes, dependia-se do pessoal de TI do cliente para realizar tarefas críticas nesse processo e frequentemente

observava-se falhas na sua execução.

Outro problema decorrente da arquitetura cliente-servidor era a dependências da aplicação de componentes do sistema operacional ou de módulos de outros fornecedores. A escolha do Kylix também levou em consideração essa questão, tendo-se buscado uma ferramenta que tivesse o máximo de independência possível do sistema operacional.

A equipe esperava minimizar esses problemas com a utilização da arquitetura SOA, já que os WS poderiam ser construidos de forma mais modular e totalmente independentes uns dos outros. O objetivo era que cada WS pudesse ser executado sem que dependesse de outros, ou seja, um determinado WS não teria conhecimento nem teria qualquer dependência com WS que ele mesmo não fizesse uso. A única exceção era o WS.Session, que fazia a autenticação no SI. Era importante que toda a autenticação fosse centralizada e, por conta disso, todos os outros WS eram dependentes do WS.Session.

Os resultados finais se mostraram satisfatórios, pois, para se implantar ou atualizar um WS era necesário apenas a cópia de dois arquivos para o servidor de aplicação. Assim, não era necessário intervir no sistema operacional ou executar rotinas extras para se fazer a implantação de um WS. Em comparação com outras tecnologias, tais como Java ou dotNet, onde são necessários copiar vários arquivos em estrutuuras de diretórios pré-determinadas, a utilização do Kylix se mostrou muito eficaz e produtiva.

Muitas vezes durante o projeto, os WS foram atualizados no servidor enquanto dois ou três usuários estavam conectados sem que houvesse queda ou perda da conexão. Até

mesmo o WS.Session foi atualizado nessas mesmas condições. Portanto, a implementação dos WS se mostrou bem resolvida nos quesitos de manutenabilidade.

Um problema encontrado foi o tamanho dos módulos gerados pela ferramenta de desenvolvimento. Cada WS ficou, em média, com 1,8Mb, e isso tornava a cópia para o servidor da Internet bastante lenta, atrasando, algumas vezes, os testes integrados. Havia também a preocupação por parte dos desenvolvedores de que o tamanho dos módulos pudesse influir na performance da execução dos WS. Foram feitos, então, testes de *stress*, para verificar se a resposta do servidor seria satisfatória. A estimativa era que 50 acessos simultâneos seriam muito mais do que o suficiente para validar a velocidade do servidor. O servidor suportou bem as requisições e, apesar das respostas ficarem lentas a partir do trigésimo acesso, os WS responderam a todas as requisições sem erros.

4.9. Estabelecer canais de comunicação

Devido a essência do modelo de negócios baseados em WS e da escolha de ferramentas baseadas em software livre, o estabelecimento de canais de comunicação com desenvolvedores, clientes e parceiros era uma fator que se mostrou fundamental, desde o início do estudo da arquitetura. Já nas primeiras etapas do projeto, estavam programadas diversas atividades com o objetivo de difundi-lo e facilitar o aprendizado dos próprios desenvolvedores e motivar novos projetos baseados na mesma tecnologia. A cada etapa

concluída, um novo tutorial era disponibilizado a comunidade sendo ao todo cinco tutoriais desenvolvidos.

Os primeiros tutoriais relativos à integração de aplicativos feitos em Kylix com o servidor Apache e à instalação do Kylix e bibliotecas básicas foi muito bem aceito na comunidade. Para divulgar os tutoriais, foram feitos anúncios das listas de discussão brasileiras especializadas em programação e em Linux e foram incluidos nos tutoriais cabeçalhos que permitiam que os serviços de busca como o Google e o Yahoo incluíssem o conteúdo do tutorial em seus bancos de dados.

Como forma de verificar quanta pessoas estavam utilizando o site, um mecanismo de contagem foi incluído. Mais de 5.000 acessos foram contabilizados entre setembro de 2004 e janeiro de 2005, e, mesmo estando escrito em língua portuguesa, o site foi visitado por vários países tais como França, Hungria, Zaire e Ilhas Maldivas.

Uma das atividades desta fase foi consultar alguns leitores dos tutoriais para saber suas opiniões sobre WS e sobre o projeto GOPE. A partir de questionários estruturados, verificou-se que os leitores tinham por objetivo aprender a utilizar as ferramentas Kylix e WebProvider não para aprender WS especificamente.

Como os tutoriais não estavam trazendo o resultado esperado, uma lista de discussão foi criada no site YahooGrupos. Esse serviço permite que qualquer pessoa crie comunidades virtuais sob forma de listas de discussão com o intuito de trocar opiniões sobre um assunto específico. A lista, denominada webservices-br, tinha como objetivo reunir gerentes e

desenvolvedores que já trabalhavam ou tinham interesse em Web Services para que eles pudessem trocar opiniões sobre a arquitetura e divulgar seus trabalhos e WS. Entretanto, a lista não teve muita procura reunindo apenas 55 pessoas.

De todos os contatos estabelecidos, apenas um veio trazer resultados positivos para a pesquisa: um desenvolvedor que havia desenvolvido um WS de consulta ao CEP, através do qual o usuário podia obter o endereço equivalente a um dado CEP. Isso proporcionou uma boa oportunidade para experimentar o intercâmbio de WS entre empresas diferentes e avaliar quais o impacto dessa interação no projeto GOPE.

Após o contato com o autor do serviço, a funcionalidade de CEP foi incorporada ao sistema ou seja, o sistema GOPE passou a utilizar um WS externo a ele, podendo-se testar o efeito de se usar serviços de diferentes fornecedores para montar uma aplicação.

Nesse teste, pôde-se observar um problema de interoperabilidade muito interessante. Como o WS de CEP foi desenvolvido em C# (linguagem do ambiente dotNet da Microsoft), o programador tinha a possibilidade de definir várias estruturas de dados como "*array of anytype*", ou seja, uma matriz de qualquer tipo. No entanto, isso viola as regras estabelecidas no WSDL, que exige que o tipo de dados contido numa estrutura seja definido. Assim, foi necessário assumir que o tipo retornado pelo serviço de CEP seria uma *string* (cadeia de caracteres), sem, entretanto ter certeza que o tipo retornado seria sempre este. Sabia-se que, se o desenvolvedor do CEP, começasse a enviar outro tipo de dado, o WS cliente iria retornar erro pois o WSDL não seria alterado. Dessa forma, a interoperabilidade e a confiabilidade

estavam claramente comprometidas devido a uma falha de implementação do ambiente dotNet que permitia esse tipo de construção.

Ainda dentro do esforço de divulgar os resultados do trabalho e cativar novos desenvolvedores para que utilizassem e testassem os WS, a equipe disponibilizava toda alteração, exemplos e ferramantas criadas para a biblioteca IndySOAP para a lista de discussão (indy-soap-public@yahoogroups.com), onde os desenvolvedores se reuniam. Tamanhas foram as contribuições enviadas para a lista que o criador da biblioteca convidou o coordenador a fazer parte da equipe principal de desenvolvedores através do email abaixo:

> I's really great to see your contributions to IndySoap. As a matter of fact, Chad and I would like to invite you to become a core member of the IndySoap team.
>
> This would mean that: you would be able to book code in and out of the version control system, you would be on the indy-soap maintainers email list (though it has been quiet for a long time now) and you would become a moderator of the public indy-soap mailing list. Are you interested?

Um outro recurso que se mostrou muito útil para o coordenador do projeto GOPE foi a lista que discutia especificamente problemas de interoperabilidade entre as diversas plataformas que implementavam SOAP. A partir dela foi possível tirar dúvidas de padronização dos protocolos SOAP e WSDL e foi possível verificar que outros desenvolvedores estavam reclamando dos mesmos problemas de interoperabilidade, ou seja, outros desenvolvedores ao redor do mundo também estavam tendo dificuldade em compatibilizar WS escritos em plataformas distintas. Esta lista pode ser vista http://groups.yahoo.com/group/soapbuilders/

Um dos maiores impecilhos para a divulgação de WS foi falta de um diretório UDDI brasileiro onde os WS desenvolvidos pudessem ser publicados ao público em geral. Tentou-se, em várias fases do projeto, inscrever os WS do projeto GOPE no site Xmethods que tinha como objetivo cadastrar e listar os WS que estivessem disponíveis na Internet. Entretanto, estas tentativas não foram bem sucedidas e, apesar de inúmeros contatos, os proprietários não souberam explicar o motivo do problema. No fechamento da pesquisa, a empresa SUN Microsystems havia criado o seu diretório UDDI para a divulgação dos WS de forma gratuita. Entretanto, não houve tempo hábil para testar o serviço.

Um evento alheio à pesquisa em si mas muito interessante ocorreu durante o desenvolvimento do projeto. Em função da divulgação da utilização de software livre, duas empresas interessadas em desenvolver SI baseados em software livre entraram em contato com a JSK após lerem os tutoriais criados. Em função deste contato, um novo projeto foi iniciado e a JSK conseguiu um novo cliente. Apesar do novo projeto não utilizar especificamente Web Services, o uso de software livre alterou o ambiente externo da empresa de forma positiva, criando um novo mercado para a JSK que antes era só tinha capacitação para desenvolver SI sob plataformas proprietárias.

4.10. Considerações Finais

Através da análise dos resultados, pode-se verificar que todos os fatores definidos inicialmente se mostraram realmente críticos para o desenvolvimento do sistema GOPE.

Principalmente a escolha da ferramenta mostrou-se um fator decisivo para o projeto uma vez que influiu praticamente em quase todos os indicadores de qualidade.

Na tabela abaixo, pode-se observar um resumo de como os indicadores de qualidade foram influenciados pelo novo processo utilizando Web Services. Os sinais positivos (+) indicam melhora nos indicadores, os negativos (-) indicam piora e os neutros (o) indicam que não se conseguiu identificar nenhuma alteração aparente ou as vantagens e desvantagens não definiram uma melhora no indicador:

Fatores Críticos de Sucesso	***F***	***Cf***	***U***	***E***	***M***	***P***	***C***	***R***
FD - Ferramentas de Desenvolvimento	**-**	**+**	**o**	**+**	**-**	**+**	**+**	**+**
EQ - Equipe Adequada		**+**		**+**	**-**		**o**	**o**
FTU - Formalização dos Testes Unitários		**o**			**+**			**+**
DIV - Divulgação dos WS	**+**		**+**				**+**	
CI - Controle da Interoperabilidade	**+**					**+**		
MI - Manutenção Incremental		**o**	**+**		**+**			

Tabela 4: Resultado dos Indicadores de Desempenho

Todos os fatores definidos inicialmente se mostraram críticos mas, no decorrer da pesquisa, o acesso ao código fonte das bibliotecas também se mostrou crítico. Sem a possibilidade de visualização e alteração do código fonte das bibliotecas de classes responsáveis pela geração dos WS, o projeto não seria concluído já que as ferramentas disponíveis originalmente não atendiam diversos critérios de qualidade que eram fundamentais para o projeto GOPE.

Outro fator que atrasou o projeto e quase fez com que ele não tivesse terminado, foi a seleção da equipe. Como não se encontrou profissionais capacitados suficientes, o projeto demorou mais do que esperado e só foi concluído porque o próprio coordenador pôde trabalhar mais horas do que o planejado.

O custo do projeto foi de R$25.600,00 e envolveu oito pessoas entre programadores, *web designers*, consultores e fornecedores durante um ano e meio. O pesquisador trabalhou 16 horas semanais enquanto os outros desenvolvedores trabalharam 4 horas semanais em média.

O projeto, até a conclusão da presente pesquisa, gerou mais de 40.000 linhas de código, entre código Pascal, HTML, XUL e PHP. Foram gerados e publicados oito Web Services. Além disso, a equipe deu manutenção em mais de 200.000 linhas de código das bibliotecas auxiliares Indy, IndySOAP, Zeos e WebProvider.

Como produtos suplementares, a biblioteca de classes WebProvider foi construída, e uma versão da biblioteca IndySOAP específica para Kylix foi disponibilizada na Internet para a comunidade de desenvolvedores.

Embora o projeto tenha sido criado com o intuito básico de gerar resultados para a pesquisa acadêmica, pretende-se dar prosseguimento às atividades de desenvolvimento para gerar o sistema GOPE como produto final em sua versão comercial integral.

5. CONCLUSÕES

Nesta pesquisa, que durou dois anos para ser concluida, foi possível avaliar a arquitetura orientada a serviços (SOA) e a utilização de Web Services em uma micro empresa especializada em desenvolvimento de software. Através da revisão de literatura, procurou-se estabelecer os conceitos da tecnologia de WS e quais estratégias poderiam ser adotadas por uma micro empresa para se beneficiar das vantagens que a tecnologia oferecia.

A partir da análise interpretativa inerente ao método da pesquisa-ação, foi possível definir os fatores críticos de sucesso para a implementação da tecnologia de WS e, apesar do projeto ter durado bem mais do que o esperado, conseguiu-se atingir o resultado esperado. O resultado da pesquisa trouxe diversas conclusões interessantes, de ordem técnica e sociais, que podem contribuir para a aprofundamento da pesquisa nesta área.

Primeiramente, a pesquisa demonstrou que, através da análise dos indicadores de qualidade e custo, a utilização de Web Services contribuiu para a melhoria dos processos de DSI na empresa estudada pois pôde-se perceber melhoras em grande parte dos indicadores. Mesmo os indicadores que não apresentaram bons resultados se mostraram deficientes em função das ferramentas de desenvolvimento adotadas, e não em função da tecnologia de WS em si.

Em relação à interoperabilidade, grande bastião da arquitetura SOA, pode-se perceber claramente que existe uma grande distância entre o material bibliográfico e as

práticas e ferramentas analisadas. Como foi visto no pequeno exemplo do WS de CEP na seção 4.4, os padrões não estão tão sedimentados assim, pois permitem interpretações diferentes por parte de quem desenvolve os WS. Assim, esse “relaxamento” dos padrões por parte de alguns fabricantes pode trazer os mesmos problemas já encontrados em arquiteturas anteriores. O problema de interoperabilidade em WS também foi citado nos trabalhos de Stal (2002) e Milinski (2004).

Conforme a literatura estudada, o uso de WS não se efetiva completamente até que estes serviços possam ser reusáveis e compartilhados entre as pessoas e organização (W3C, 2002). Assim, o pesquisador esperava encontrar outras empresas e parceiros que se interessassem em utilizar os WS desenvolvidos para que se pudesse analisar em que grau o reuso de WS entre empresas poderia afetar ou beneficiar o desenvolvimento do SI. Entretanto, nenhum resultado consistente pôde ser obtido, pois a única empresa que estava disponibilizando seus WS não se interessou em integrar os dois sistemas.

Esta constatação também foi citada em recente pesquisa da EDC (2005), que relatou que as empresas, em mais da metade das entrevistas, não tiveram vantagens econômicas na produção de Web Services ou somente tiveram implantações confinadas em uma única unidade de negócio. A pesquisa do EDC afirma ainda: “se um WS estiver confinado em apenas uma aplicação ou em um único departamento, ele tem o mesmo valor que uma antiga aplicação legada tinha. Os aspectos de geração de valor e a redução de custos que podem surgir na utilização de WS advem exatamente do reuso deste WS em vários departamentos ou clientes” (EDC, 2005).

A utilização de software livre (SL) se mostrou viável no caso investigado e contribuiu em vários aspectos para o projeto de DSI, sugerindo que a sua adoção pode ser benéfica para PMEs especializadas em DSI. Entretanto, ficou claro para todos os desenvolvedores envolvidos no projeto e para o pesquisador que o uso de software livre não é um caminho de mão única. Diferentemente do software proprietário, que não pode ser alterado e já conta com funcionalidades pré-determinadas pelo fabricante, o software livre está em constante mudança, e esta mudança depende intimamente da interação do usuário com a comunidade que criou o software. Uma dos maiores aprendizados desta pesquisa foi a verificação de que o gerente ou empresário que for utilizar SL deve planejar e reservar recursos para possibilitar esse engajamento, por parte de sua equipe, no desenvolvimento das ferramentas que eles mesmos estarão utilizando. Pode-se concluir, assim, que o software realmente não é grátis, e sim livre, pois existe um custo inerente ao uso para que se possa

testar, aprimorar e distribuir os aprimoramentos, sejam eles sob forma de código, de documentação ou de exemplos de como utilizá-lo.

Em contrapartida, este processo de interação com o SL trouxe um entedimento das ferramentas utilizadas no desenvolvimento do SI muito mais profundo que os anteriormente experimentados com ferramentas de código fechado. Ao se ter acesso ao código fonte e contato direto com os autores do SL, os participantes da pesquisa se aprofundaram no conhecimento de XML, WS e da arquiteura SOA o que facilitou o desenvolvimento do SI de forma geral.

A partir deste trabalho, fica patente que o DSI ainda tem muitos caminhos a percorrer. Quem se aventura nessa seara tem que vir preparado para encontrar muitos problemas ainda não levantados pela pesquisa acadêmica e que ainda não são tratados com facilidade pelas ferramentas encontradas no mercado. Existem novos e diversos caminhos que podem ser trilhados e que podem trazer vantagens competitivas para as empresas desenvolvedoras de software.

Assumir que a TI já se tornou um *commodity*, como sugere Carr (2003), seria o mesmo que afirmar que a válvula eletrônica utilizada nos primeiros televisores já seria suficiente para os consumidores, pois o importante era transmitir imagens, não importando muito a sua eficiência.

A partir de estudos como estes, pode-se entender que o DSI ainda está na época das válvulas e percebe-se que seus principais componentes, os componentes humanos, são bem

mais complexos que átomos e elétrons. Metaforicamente, descobrir como construir "transistores" a partir da interação homem-máquina não é tarefa fácil. Enquanto o aspecto humano não for levado em conta e estudado com mais atenção, principalmente nos aspectos de usabilidade e de motivação dos desenvolvedores, a pesquisa científica pode não atingir os resultados que a sociedade espera.

5.1. Trabalhos Futuros

Considerando que a tecnologia de WS e a arquitetura orientada a serviços (SOA) possa vir a ser o futuro do desenvolvimento de sistemas, tal como foi a arquitetura cliente-servidor e tantas outras, novas pesquisas nesta área parecem ser fundamentais.

Pesquisas quantitativas, que avaliem o tamanho do mercado de WS no Brasil, podem ajudar pesquisadores a mapearem como os WS estão sendo utilizados e se os fatores críticos de sucesso encontrados nesta pesquisa também são aplicáveis em outras empresas.

Estudos avaliando como as empresas estão comprando e utilizando os WS já prontos podem complementar esta pesquisa, mostrando, assim, o outro lado da questão, isto é, de quem apenas consome os WS e não os fabrica. De forma mais geral, estudos tratando especificamente da utilização de WS entre empresas parecem ser importantes para avaliar a questão da interoperabilidade e do reuso de código, pilares da arquitetura SOA. Dentro ainda deste assunto, a avaliação dos protocolos UDDI e WS-Inspect que não foram avaliados nesta pesquisa podem se mostrar interessantes.

A questão da segurança na utilização de WS também é assunto vasto e devem ser estudada a fundo pois é de fundamental importância para o desenvolvimento da tecnologia. Além disso, a interação de ferramentas CASE para modelagem de processos também podem ser estudados com o objetivo de se entender como esta interação pode trazer de benefícios para as empresas.

6. BIBLIOGRAFIA

ABOWD, G. et al. **Recommended Best Industrial Practice for Software Architecture Evaluation. Technical Report.** CMU/SEI-96-TR-025. 1996.

ALBERTIN, Alberto Luis. **Comércio eletrônico. Modelo, aspectos e contribuições de sua aplicação.** São Paulo : Ed.Atlas, 1999.

ANDREWS, W. **SOA Has Impact on Application Development Outsourcing.** Gartner Group. Disponível em <http://www.gartner.com>. Acesso em 18/04/2004.

AVISON, D.; LAU, Francis; MYERS, Michael; NIELSEN, P. Axel. **Action Research.** Communications of the ACM, Vol. 42. Jan 1999.

BAHSOON, R. EMMERICH W. **Evaluating Software Architectures: Development, Stability, and Evolution.** Proceedings of ACS/IEEE Int. Conf. on Computer Systems and Applications. Tunisia, 2003.

BAKER, C. Richard. **Towards the increased use of Action Research in accounting information System.** Accounting Forum, Dec.2000, Vol.24 Issue 4, pg.336 (2000)

BALDWIN, C.Y. e CLARK, Kim B. **Managing in an age of modularity.** Harvard Business Review. Set/Out 1997.

BALESTRIN, A. e VARGAS, L.M. **A dimensão estratégica das redes horizontais das PMEs: Teorizações e Evidências.** Revista de Administração Contemporânea - Edição Especial 2004. ANPAD, 2004.

BARRY, Douglas. **Web Services and Service-Oriented Architectures.** Elsevier Publishing, New York. 2003. pg.76.

BASKERVILLE, R.L.; WOOD-HARPER, T.A. **A critical perspective on action research as a method for information systems research.** Journal of Information Technology (1996) 11,235-246.

BASKERVILLE R.L.; Pries-Heje, J. **Grounded action research: a method for understanding IT in practice.** Accting., Mgmt. & Info. Tech. 9 1–23 (1999)

BASKERVILLE R.L. **Investigating information systems with action research.** Communications of AIS, Volume 2, Article 19. (1999)

BASS K. Clements P. e Kazman, R. **Software Architecture in Practice.** Addison-Wesley Professional, 2a edição (2003)

BHATTI, Shahid N. **Why Quality? ISO 9126 Software Quality Metrics (Functionality)**

Support by UML Suite. ACM SIGSOFT Software Engineering Notes Page 1 March 2005

BOEHM, Barry. **An Experiment in Small-Scale Application Software Engineering.** IEEE Trans. Software Eng. pg. 482-493. 1981.

BOEHM, Barry. **Software Architectures: Critical Success Factors and Cost Driven.** Proceedings of the 16th International Conference on Software Engineering. pg.365. 1994.

BOOCH, Grady. **Web Services: The Economic Argument**. Software Development Magazine. May, 2003. Disponível em <http://www.sdmagazine.com/documents/s=1478/sdm0111d/0111d.htm>. Acesso em 06/2003.

BONCELLA, Robert. J. **Web Services and Web Services Security.** Proceedings of the Americas Conference on Information Systems. August, 2004.

BRASIL. República Federativa - Governo Federal. **Portal de Software Livre.** Disponível em <http://www.softwarelivre.gov.br> Acesso em 3/08/2004.

BRODE, M. e STONEBRAKER, M. **Migrating legacy systems.** San Francisco: Morgan Kaufmann Publishing. 1995.

BRODBECK, A.F. **Alinhamento Estratégico entre os Planos de Negócio e de Tecnologia de Informação: um Modelo Operacional para a Implementação.** Tese de Doutorado, 2001. Universidade do Rio Grande do Sul – PPGA. 2002.

BUNKER, Debora.J e MacGregor, R.C. **The Context of Information Technology and Eletronic Commerce Adoption in Small/Medium Enterprises: A Global Perspective.** Eighth Americas Conference on Information Systems. 2002.

CARR, Nicholas. **IT doesn't Matter.** Harvard Business Review. Maio, 2003.

CERDEIRA, P.C. MONIZ, P. P. **Copyleft e Software Livre: uma opção pela razão – eficiências tecnológica, econômica e Social.** Revista da ABPI, n.70, 2004.

CHEN, Minder. **Factors affecting the adoption and diffusion of XML and Web services standards for E-business systems**. International Journal of Human Computer Studies, Volume 58. Pg.259-279. 2003.

CHECKLAND, Peter. **Achieving 'Desirable and Feasible' Change: An application of Soft Systems Methodology.** The Journal of the Operational Research Society. Vol.36 No.9. 1985.

CHUA, B.B. & Dyson, L.E. **Applying the ISO9126 model to the evaluation of an elearning system.** In R. Atkinson, C. McBeath, D. Jonas-Dwyer & R. Phillips (Eds),

Beyond the comfort zone: Proceedings of the 21st ASCILITE Conference (pp. 184-190). Perth, 5-8 Dezembro, 2004.

CLABBY, J. **Web Services Explained - Solution and Applications for the real world.** New York: Prentice Hall PTR. 2002.

CLOETE, E. Courtney, S. **Small Businesses' Acceptance and Adoption of E-Commerce in the Western-Cape Province of South-Africa.** The Eletronic Journal on Information Systems in Development Countries. 2002.

COAD, Peter. YOURDON, E. **Projeto baseado em Objetos.** Editora Campus, 1993.

COLOMBO, Regina. GUERRA A.C. **The evaluation method for software product.** ICSSEA - International Conference Software & Systems Engineering and their Applications. Paris - França. 2002.

COMPUTERWORLD. **Pequenas e Médias Empresas devem investir mais em 2003.** Disponível em <http://www.computerworld.com.br>. Acesso em 19/03/2003.

CONSENZA, José Paulo. **A evolução da escrituração contábil através dos tempos. Uma revisão história da contabilidade contemporânea com base na literatura contábil.** Dissertação de Mestrado. Rio de Janeiro: UERJ, 1999.

COUNIHAN, A. Finnegan, P. Sammon, D. **Towards a framework for evaluating investiments in Data Warehousing.** Information Systems Journal, 12, 321-338. 2002.

CRONHOLM, Stefan. **Understanding practices of AR.** 2nd European Conference on Research Methodology for Business and Management Studies. 20-21 March.2003

DATAMONITOR. **Real Web Services.** Special report on Web Services. 2003. Disponível em <http://ww.datamonitor.com>. Acesso em 30/10/2003.

EDEN, Colin; HUXMAN, Chris; **Action Research for Management Research.** British Journal of Management, Vol. 7, 75-86 (1996)

EDC; **Web Services Development Survey, Spring 2005.** Disponível em <http://www.evansdata.com/n2/surveys/webservices/2005_1/webservices_05_1_xmp3.shtml> Acesso em 21/06/2005.

DAVID, Matthew. **Problems of participation: the limits of action research.** Int.J.Social Research Methodology, 2002. Vol

FAVARO, J. **When the pursuit of quality destroys value.** IEEE Software. May, 1996.

FRANK**,** Ulrich, FRAUNHOLZ, Bardo. **Applying Action Research to Designing, Introducing and Evaluating Information Systems in Small and Medium sized**

Enterprises (SMEs): Prospects and Critical Success Factors. The 8th European Conference on Information Technology Management (2001).

FITZGERALD, Brian; KENNY, Tony. **Open Source Software in the Trenches: Lesson from a Large-Scale OSS implementation.** Twenty-Fourth International Conference on Information Systems (2003).

FURLAN, José David. **Modelagem de objetos através da UML – the unified modeling language.** São Paulo : Makron Books, 1998.

GARLAN et al. **Architectural Styles, Design Patterns, and Objects.** IEEE Software, pp.43-52, Jan/Fev 1997

GHEZZI, C., Jazayeri, M., Mandrioli, D., **Fundamentals of Software Engineering**, Prentice Hall, Inc., 1991.

GIANESI, Irineu G.N. Corrêa, H.L. **Administração estratégica de serviços: operações para a satisfação do cliente.** São Paulo: Atlas, 1996

GEFEN, D. KEIL, M. **The impact of developer Responsiveness on Perceptions of Usefulness and Ease of Use.** ACM SIGMIS Database for Advances in Information Systems. Vol.29 No.2. 1998.

GRANT, Delvin. NGWENYAMA, Ojelanki. **A report on the use of action research to evaluate a manufacturing information systems development methodology in a company.** Information Systems Journal 13, 21–35. (2003)

GREW, Paul. **ISO Certification for XP at LogicaCMG. XP123 – Exploring XP Programming.** Disponível em <http://www.xp123.com/xplor/xp0406a/index.shtml>. Acesso em 05/2005.

HAGEL, J.; BROWN J.S. **TI Flexível, a melhor estratégia**. HSM Management 43. Abril, 2004.

___. **Your Next IT Strategy.** Harvard Business Review, 79. pp.105-113. 2001.

HAMMER, Michael. CHAMPY, James. **Reengenharia – Revolucionando a empresa.** Ed.Campus. 1990

HOLANDA, Victor; RICCIO E.L. **A utilização da pesquisa ação para perceber e implementar sistemas de informações empresariais.** In: 13 th Asian Pacific, 2001, Rio de Janeiro. (2001)

HORBST, Alexander; Fink, K.; Goebel, G. **The ISO/IEC 9126-1 as a Supporting Means for the System Development Process of a Consumer Health Information Web**

Service. HINZ 2005: Fourth Health Informatics Conference; pages: [50-53]. Brunswick East, Vic.: Health Informatics Society of Australia, 2005.

HP. Hewlett-Packard Company. **Building an Adaptative Enteprise.** Public White Paper. Disponível em <http://www.hp.com>. Acesso em 06/2004.

HUANG, C.D, HU, Q. **Integrating Web Services with competitive strategies: A Balanced Scorecard Approach.** Communications of the Association for Information Systems. Vol.13. 2004.

HULT, Margareta; LENNUNG S. **Towards a definition of Action Research: A note and bibliography.** Journal of Management Studies, Vol. 17 Issue 2, p241, 10p. (1980)

IBGE. **As Micro e Pequenas Empresas Comerciais e de Serviços no Brasil: 2001.** IBGE, Coordenação de Serviços e Comércio – Rio de Janeiro: IBGE. 2003.

IYER, B; FREEDMAN, J; GAYNOR, M. e WYNER G. **Web Services: Enabling Dynamic Business Networks.** Communications of the Association for Information Systems. Volume 11, 2003. pg.525-554.

JAKOVLJEVIC**,** Maria. ANKIEWICZ, Piet. **Action research in an Information Systems Design context: Exploring a variety of instructional strategies and techniques to enhance technological problem solving.** Action Learning, Action Research & Process Management (ALARPM) 6th World Congress. 2003.

JOUNG, Philip. **Bulletproof Web application deployments: best practices in testing.** Web Services Journal, v4 i1 p14, Jan 2004.

JUDE. **A Java/UML Object-Oriented Design Tool.** Disponível em <http://objectclub.esm.co.jp/Jude/jude-e.html>. Acesso em 11/2003.

KAVANAGH, J.F. **Resistance as motivation for innovation: Open Source Software.** Communication of the AIS. pg.615-628. Vol.13. 2004

KLEIN, H.Z.; MYERS M.D. **A set of principles for conducting and Evaluating Intepretive Field Studies in Information Systems.** MIS Quarterly. Vol.23 N.1 p.67-94. Março 1999.

KOCK, N.F. **Action Research: Lessons Learned From a Multi-Iteration Study of Computer-Mediated Communication in Groups.** IEEE transactions on professional communication, VOL. 46, NO. 2, JUNE 2003.

_____. **The Effects of Asynchronous Groupware on Business Process Improvement.** Tese de Doutorado. University of Waikato, Hamilton, New Zealand (1997)

KREGER, Heather. **Fullfilling the Web Services Promise.** Communications of the ACM. Junho de 2003.

KRUCHTEN, Philippe. **Using the RUP to evolve a legacy system.** The Rational Edge, Jun/2003. Disponível em <http://www-106.ibm.com/developerworks/rational/library/389.html>. Acesso em 25/03/2004.

KÜSTER, Edison. **Metodologia para o organização do processo operacional das empresas de pequeno porte, visando a implantação de sistemas de informação.** Dissertação de Mestrado. UFSC, 2001.

LEVY, Margi; Powell, P. Yetton P. **SME: Aligning IS and the strategic context.** Journal of Information Technology (2001) 16, p.133-144.

LAURINDO, F.J. et al. **O papel da TI na estratégia das organizações.** Revista Gestão e Produção. v.8, n.2, pg. 160-179. Ago, 2001.

LEVY, Steven. **Hackers.** Anchor/Doubleday 1984.

LEYMAN J.; ROLLER D. e SCHMIDT, M.T. **Web Services and Business Process Management.** IBM Systems Journal, Vol.41, No.2, 2002

LIM, B., WEN, H. J.. **Web Services: An analysis of the technology, its benefits, and implementation difficulties**. Information Systems Management Journal. Spring, 2003.

LUBBEN, Richard. **Just-In-Time: uma estratégia avançada de produção.** São Paulo: McGraw-Hill, 1989.

MACHADO, Solange A.; Pizysieznig Filho, J. et al. **MPEs de Base Tecnológica: conceituação, formas de financiamento e análise de casos brasileiros.** Relatório de Pesquisa SEBRAE, 2001.

MCCUTCHEON, G. & JUNG, B. (1990). **Alternative Perspectives on Action Research.** Theory Into Practice, vol. 29, no. 3, pp. 144-151.

MCKAY, Jude. Marshall, Peter. **2 x 6 = 12, or Does It Equal Action Research ?** Proc. 10th Australian Conference on Information Systems, 1999.

MCTAGGART, Robin. **Is Validity really a issue in AR. Studies in Cultures, Organizations and Societies**, Vol.4 pg.211-236. 1998.

MILES R. & Snow C. **Organizational Strategy: structure and process.** New York, McGraw Hill Book Company, 1978

MILINSKI, O.Z. OOLERMAN, M.C. **Second Generation Web Services-Oriented Architecture in Production in the Financial Industry.** OOPLSA 2004, Vancouver, Canada. 2004.

MOONEY, J. GURBAXANI, V. **A process oriented framework for assessing the business value of Information Technology.** ACM SIGMIS Vol.17 Issue 2, 1996.

MORRIS, Charlie. **SOAP, the Simple Object Access Protocol.** Web Developers Library. Disponível em <http://www.wdvl.com/Authoring/Languages/XML/Soap/>. Acesso em 12/01/2005.

MUSTONEN-OLLILA, E., LYYTINEN, K. **How organizations adopt information system process innovations: a longitudinal analysis.** European Journal of Information Systems (2004) 13, 35–51

NETCRAFT. **August 2004 Web Server Survey.** Disponível em <http://news.netcraft.com/archives/2004/08/01/august_2004_web_server_survey.html> Acesso em 27/08/2004.

NIST. Standards for ELECTRONIC DATA INTERCHANGE (EDI). Disponível em <http://www.itl.nist.gov/fipspubs/fip161-2.htm> Acesso em 12/01/2005.

NOGUEIRA NETO, Mário S. et al. **Aplicação de Sistemas Integrados de Gestão em Pequenas e Médias Empresas.** Anais – III Simpósio de Administração da Produção, Logística e Operações Internacionais – FGV/SP, 2000.

O´BRIEN, James. **Sistemas de Informação e as Decisões Gerenciais na Era da Internet.** São Paulo: Saraiva, 2003.

OLIVEIRA, Edson. **Contabilidade Informatizada.** São Paulo: Atlas, 1997.

OLSON, T. G., N. R. Reizer, & J. W. **A Software Process Framework for the SEI Capability Maturity Model.** Documents and Checklists for a Software Process Framework for CMM. 1994.

PADUAN, Roberta et al. **Tesouro Escondido.** Revista EXAME. São Paulo, v.37 n.13, 2003.

PARNAS, D.L. **On the Criteria To Be Used in Decomposing Systems into Modules.** Communications of the ACM. Vol.15. N. 12. 1972.

PATTON, Jeff. **Unfixing the Fixed Scope Project: Using Agile Methodologies to Create Flexibility in Project Scope.** ADC, vol. 00, p.146, Agile Development Conference 2003.

PETKOV, D. Fry, G.S. **Assisting small information technology companies identify Critical Success Factors in Web Development Projects.** AIS - Americas Conference on Information Systems 2003.

PORTER, M.E. **Competitive Advantage**. The Free Press, New York NY. 1985.

POURKOMEYLIAN, Pouya.**Knowledge Creation in Improving a Software Organization.** Proceedings of IRIS 23. Laboratorium for Interaction Technology, University of

Trollhättan Uddevalla, 2000.

POWELL, T.C., DENT-MICALEFF. **Information Technology as Competitive Advantage: The role of Human, Business and Technology Resources.** Strategic Management Journal, Vol.18:5, pg.375-405. 1997.

PRESSMAN, R. S. **Software Engineering – A Practitioner's Approach.** 5a Edição, McGraw-Hill series in computer science, 2001.

REEL, J. S. **Critical success factors in software projects.** IEEE Software v.16 (1999)

ROBINSON, Viviane M.J. **Current Controversies in AR.** Public Administration Quarterly. Fall, 93. Vol.17. Issue 3, p.263. (1993)

ROTHENBERGER, M. Kulkarni, U. Dooley, K. **Critical Success Factors for Software Reuse Projects.** The 19th Americas Conference on Information Systems. Vol. 19 (1998)

RUSS, Melissa L, McGREGOR, John D. **A Software Development Process for Small Projects.** IEEE Software. September/October 2000.

SAYLES, Jonathan. **COBOL and the business program paradigm.** MicroFocus. Disponível em <http://www.microfocus.com>. Acesso em 30/03/2004.

SCHUPP, S. Zalewski, M. Ross, K. **Rapid performance prediction for library components.** ACM SIGSOFT Software Engineering Notes , Proceedings of the 4th international workshop on Software and performance WOSP '04, Volume 29 Issue 1, 2004.

SEBRAE. **Estudos e Pesquisas: Legislação Básica da Micro e Pequena Empresa.** Disponível em <http://www.sebrae.com.br/br/aprendasebrae/estudosepesquisas.asp>. Acesso em 30/08/2004

SILVEIRA, Denis, SCHMITZ, Eber. **Uma Metodologia de Desenvolvimento de Sistemas de Informações em Empresas de Pequeno e Médio Porte.** ENANPAD, 2002.

SIQUEIRA, José Roberto B. **Source Inspector – uma ferramenta para extração de regras de negócio em sistemas legados.** Monografia (Mestrado em Informática). UFRJ /IM / NCE. Rio de Janeiro. 2002.

SLEEPER, B. **Defining Web Services.** San Francisco: The Stencil Group. 2001.

SOFTEX. **A Indústria de Software no Brasil – 2002.** Fortalecendo a economia do crescimento. Capítulo Brasil do projeto: Slicing the knowledge-based economy in India, China and Brasil: a tale of the three software industries. 2002.

SORUMGÅRD, Sindre. **Verification of Process Conformance in Empirical Studies of Software Development.** Tese de Doutorado. Department of Computer and Information Science. The Norwegian University of Science and Technology. 1997

STAIR, R., **Princípios de Sistemas de Informação – Uma Abordagem Gerencial.** Segunda Edição, Editora LTC, 1998

STAL, Michael. **Web services: beyond component-based computing**. Communications of the ACM. Volume 45, Issue 10, 2003.

STALLMAN, Richard; **The Free Software Definition.** Disponível em <http://www.gnu.org/philosophy/free-sw.html>. Acesso em 4/12/2003.

STEENIS, H.V. **How to plan, develop and use Information Systems.** Dorset House Publishing. 1990.

STEWART, Thomas. **Does IT Matter ? An HBR Debate. Harvard Business Review.** June, 2003.

SUSMAN, Gerald.I; EVERED, Roger. D. **An Assessment of the Scientific Meerits of Action Research.** Journal of Management Studies, May80, Vol. 17 Issue 2, p241, 10p.

SWISSDC, Swiss Delphi Center. **Interview with Chad Z. Hower.** Disponível em <http://www.swissdelphicenter.ch/en/chadhower.php>. Acesso em Setembro/2005.

TECHWEB. **Munich Presses On With Microsoft-to-Linux Move.** Disponível em <http://www.techweb.com/wire/story/TWB20040812S0007>. Acesso em Agosto/2004.

TELES, Vinícius Manhães. **Extreme Programming.** Novatec Editora. 2004.

THIOLLENT, Michel. **Pesquisa-Ação nas Organizações.** Editora ATLAS. 1a Edição 1997.

_____. **Metodologia da Pesquisa-Ação.** 2 a ed. São Paulo: Cortez Editora, 1986.

THONG, James. **An Integrated Model of Information Systems Adoption in Small Businesses.** Journal of Management Information Systems, Vol. 15, Issue 4. 1999.

TRIPP, David. **Socially Critical Action Research.** Theory in practice. Summer,1990. Vol.29. Issue 3. pg.158

US-DOJ. **Microsoft agrees to end unfair monopolistic practices.** United States of America – Department of Justice. Boletim informativo 94-387. Disponível em <http://www.usdoj.gov/opa/pr/Pre_96/July94/94387.txt.html>. Acesso em 25/04/2005.

UNCTAD - United Nations Conference on Trade and Development. **E-Commerce and Development Report 2003. Free and open-source software: Implications for ICT**

policy and development. pg.110. 2003.

VITHARANA, P. **Risks and Chalenges of Component-Based Software Development.** Communications of the ACM. August 2003/Vol.46,No.8.

VUNET. **50,000 developers use Amazon web services.** Disponível em <http://www.vnunet.com/News/1154546>. Acesso em 25/04/2004.

WARD, J. and Griffiths, P. **Strategic Planning for Information Systems.** John Wiley & Sons Ltd, Chichester, UK. 1996.

WASMUND, M. **Implementing critical success factors in software reuse.** IBM Systems Journal. Vol. 32 N. 41993.

WESTBRIDGE Corp. **Web Service Usage Survey.** Disponível em <http://www.westbridge.com>. Acesso em 01/2004.

World Wide Web Consortium – W3C. **Web Services Activity Statement.** Disponível em <http://www.w3.org/2002/ws/Activity> . Acesso em 05/2003.

_____. **Web Services Architecture - Working Draft.** Disponível em <http://www.w3.org/TR/2003/WD-ws-arch-20030514> . Acesso em 05/2003.

ANEXOS

ANEXO I – Exemplo do protocolo SOAP

Neste exemplo, pode-se visualizar como é internamente a chamada da função WS.Session.Login, onde se envia três parâmetros: Login, Pwd (Senha) e Machine:

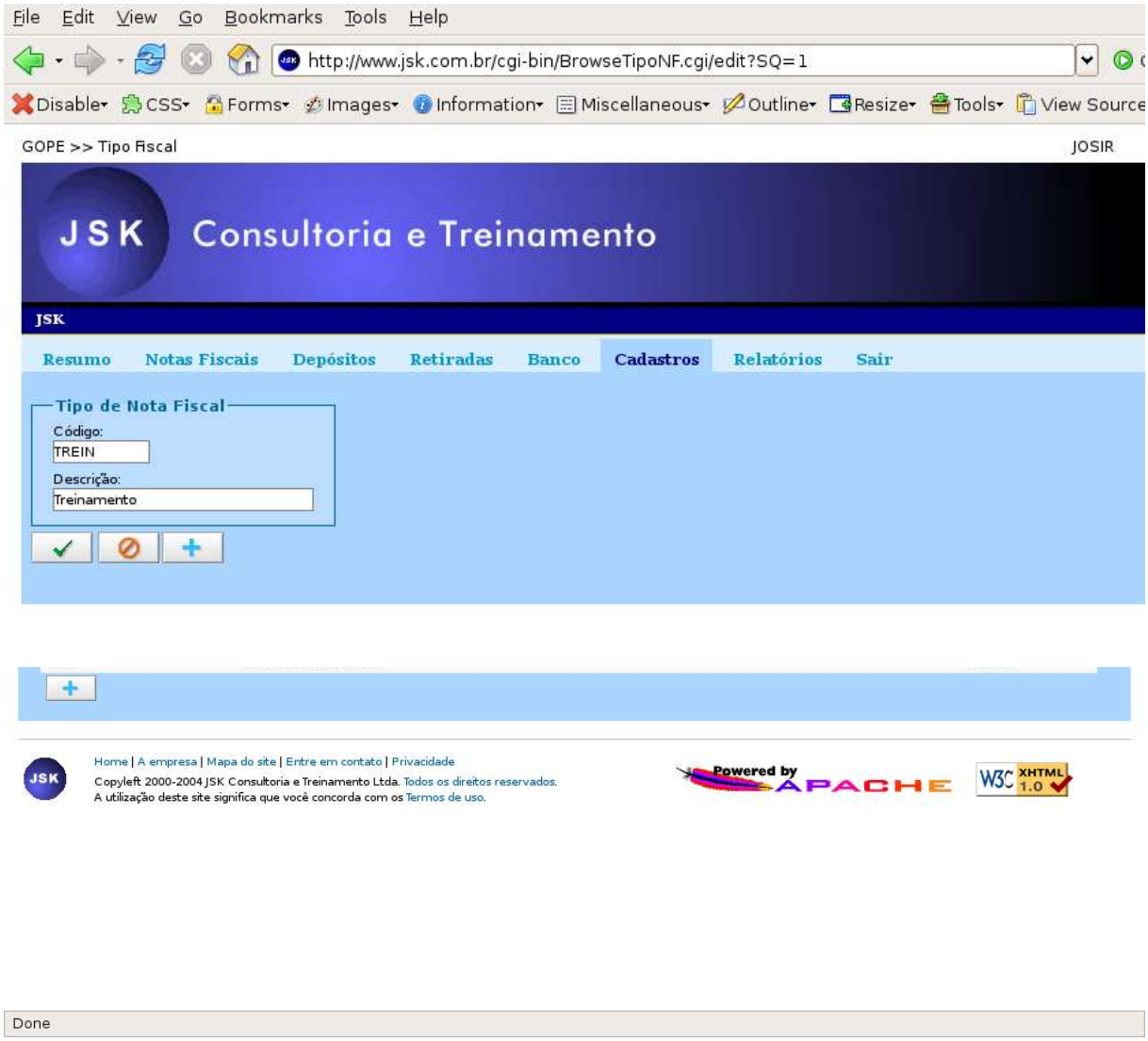
```
<soap:Envelope
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:ns="urn:nevrona.com/indysoap/v1/">
<soap:Body>
<ns:Login soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<Login xsi:type="xs:string">josir</Login>
<Pwd xsi:type="xs:string">xuxu33</Pwd>
<Machine xsi:type="xs:string">200.141.125.126</Machine>
</ns:Login>
</soap:Body>
</soap:Envelope>
```

ANEXO II – Exemplo de WSDL – WS.Empresa

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="IEmpresa" xmlns="http://schemas.xmlsoap.org/wsdl/"
targetNamespace="urn:nevrona.com/indysoap/v1/"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:tns="urn:nevrona.com/indysoap/v1/"
xmlns:ns="http://schemas.xmlsoap.org/soap/encoding/"><types><xs:schema
targetNamespace="urn:nevrona.com/indysoap/v1/">

<xs:simpleType name="longint" base="xs:int"></xs:simpleType>
<xs:complexType name="TGOPESimpleList">
  <xs:complexContent>
    <xs:restriction base="ns:Array">
      <xs:attribute
                ref="ns:arrayType"
                arr:arrayType="tns:TGOPESimpleInfo()"
                xmlns:arr="http://schemas.xmlsoap.org/wsdl/">
      </xs:attribute>
    </xs:restriction>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="TGOPESimpleInfo">
  <xs:sequence>
        <xs:element name="SQ" type="xs:string"></xs:element>
        <xs:element name="Codigo" type="xs:string"></xs:element>
        <xs:element name="Descricao" type="xs:string"></xs:element>
  </xs:sequence>
</xs:complexType>
</xs:schema>
</types>

<message name="Criar">
  <part name="SID" type="xs:string"></part>
  <part name="Nome" type="xs:string"></part>
  <part name="Cidade" type="xs:string"></part>
  <part name="Estado" type="xs:string"></part>
  <part name="Login" type="xs:string"></part>
</message>

<message name="CriarResponse">
```

```
  <part name="SQEmpresa" type="xs:int"></part>
</message>

<message name="GetSQ">
  <part name="SID" type="xs:string"></part>
  <part name="Nome" type="xs:string"></part>
</message>

<message name="GetSQResponse">
   <part name="return" type="xs:int"></part>
</message>

<message name="GetNome">
   <part name="SID" type="xs:string"></part>
   <part name="SQEmpresa" type="xs:int"></part>
</message>

<message name="GetNomeResponse">
   <part name="return" type="xs:string"></part>
</message>

<message name="GetCidade">
          <part name="SID" type="xs:string"></part>
          <part name="SQEmpresa" type="xs:int"></part>
</message>

<message name="GetCidadeResponse">
          <part name="return" type="xs:string"></part>
</message>

<message name="SetCidade">
          <part name="SID" type="xs:string"></part>
          <part name="SQEmpresa" type="xs:int"></part>
          <part name="Cidade" type="xs:string"></part>
</message>

<message name="ListResponse">
          <part name="Lista" type="tns:TGOPESimpleList"></part>
</message>

<portType name="IEmpresa">
   <operation name="Criar">
          <input message="tns:Criar"></input>
          <output message="tns:CriarResponse"></output>
```

```
  </operation>
  <operation name="GetSQ">
        <input message="tns:GetSQ"></input>
        <output message="tns:GetSQResponse"></output>
  </operation>
  <operation name="GetNome">
        <input message="tns:GetNome"></input>
        <output message="tns:GetNomeResponse"></output>
  </operation>
  <operation name="GetCidade">
        <input message="tns:GetCidade"></input>
        <output message="tns:GetCidadeResponse"></output>
  </operation>
  <operation name="SetCidade">
        <input message="tns:SetCidade"></input>
        <output message="tns:SetCidadeResponse"></output>
  </operation>
  <operation name="PermiteAcesso">
        <input message="tns:PermiteAcesso"></input>
        <output message="tns:PermiteAcessoResponse"></output>
  </operation>

.....

  <service name="EmpresaService">
        <port name="EmpresaServiceSoap" binding="tns:EmpresaServiceSoap">
              <soap:address location="http://www.jsk.com.br/cgi-bin/Empresa.cgi">
              </soap:address>
        </port>
  </service>
</definitions>
```


File Edit View Go Bookmarks Tools Help
http://www.jsk.com.br/cgi-bin/BrowseTipoNF.cgi/edit?SQ=1
Disable CSS Forms Images Information Miscellaneous Outline Resize Tools View Source
GOPE >> Tipo Fiscal
JOSIR
JSK Consultoria e Treinamento
JSK
Resumo Notas Fiscais Depósitos Retiradas Banco Cadastros Relatórios Sair
Tipo de Nota Fiscal
Código:
TREIN
Descrição:
Treinamento
Home | A empresa | Mapa do site | Entre em contato | Privacidade
Copyleft 2000-2004 JSK Consultoria e Treinamento Ltda. Todos os direitos reservados.
A utilização deste site significa que você concorda com os Termos de uso.
Powered by APACHE
W3C XHTML 1.0
Done

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo