UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE CIÊNCIAS E TECNOLOGIA
CURSO DE PÓS-GRADUAÇÃO EM INFORMÁTICA

DISSERTAÇÃO DE MESTRADO

WIDEWORKWEB – UMA METODOLOGIA PARA O DESENVOLVIMENTO DE APLICAÇÕES WEB NUM CENÁRIO GLOBAL

Vanessa Farias Dantas

Francilene Procópio Garcia
(Orientadora)

Campina Grande – PB
Julho - 2003

UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE CIÊNCIAS E TECNOLOGIA
CURSO DE PÓS-GRADUAÇÃO EM INFORMÁTICA

WIDEWORKWEB – UMA METODOLOGIA PARA O DESENVOLVIMENTO DE APLICAÇÕES WEB NUM CENÁRIO GLOBAL

Vanessa Farias Dantas

Francilene Procópio Garcia
(Orientadora)

Dissertação submetida à Coordenação do Curso de Pós-graduação em Informática da Universidade Federal de Campina Grande, como parte dos requisitos necessários para obtenção do grau de mestre em Informática.

Área de concentração: Ciência da Computação.

Linha de pesquisa: Engenharia de Software.

Campina Grande – PB
Julho – 2003

Agradecimentos

A Deus, pela saúde, perseverança e iluminação em todos os momentos.

Aos meus pais, Valnice e Francisco, por acreditarem em mim e demonstrarem sempre todo o seu amor e orgulho.

Ao meu irmão e grande amigo, Vinicius, por sua ajuda e apoio.

À minha orientadora, Francilene, por sua paciência e compreensão, e por me ajudar a entender o valor de uma boa pesquisa.

A todos os meus amigos, em especial Ana Paula, Pasqueline, Rodrigo Rebouças, Fred e todos os integrantes do PET, por contribuírem direta ou indiretamente com palavras e gestos para minha felicidade e paz de espírito durante a realização deste trabalho.

A todos aqueles que torceram e oraram por mim para que eu superasse todos os obstáculos e atingisse esta meta.

Resumo

A evolução das tecnologias de comunicação observada nas duas últimas décadas tem levado a comunidade de software a buscar novas abordagens de desenvolvimento. Projetos com times geograficamente distribuídos desenvolvendo software para mercados culturalmente heterogêneos tornaram-se viáveis, e, com a crescente disseminação da Internet, a produção de aplicações baseadas na Web para os mais diferentes propósitos passou a ser um negócio lucrativo.

Este trabalho apresenta uma análise detalhada sobre o Desenvolvimento Global de Software e o Desenvolvimento de Aplicações Web, indicando suas principais vantagens e seus desafios. Conforme indicado ao longo do texto, as particularidades dessas abordagens não são contempladas pelas práticas tradicionais da Engenharia de Software.

Finalmente, é proposta uma metodologia de desenvolvimento simples e objetiva para projetos com times distribuídos trabalhando juntos na produção de Aplicações Web transacionais para diferentes mercados. A metodologia *WideWorkWeb* é baseada nos princípios dos Processos Agile e na experiência de diversas empresas de software, e representa uma importante contribuição para todos aqueles interessados nessas novas abordagens de desenvolvimento de software.

Abstract

Communication technologies evolution observed in the last two decades has driven the software community into new development approaches. Projects with geographically distributed teams developing software for culturally heterogeneous markets have become viable, and, with Internet's growing dissemination, producing web-based applications for the most different purposes has turned into a profitable business.

This work presents a detailed analysis about Global Software Development and Web Application Development, pointing out their main advantages and challenges. As indicated in the text, traditional Software Engineering practices do not consider these approaches particularities.

Finally, a simple and objective software development methodology is proposed to be used in projects where distributed teams work together producing transactional Web Applications for different markets. The *WideWorkWeb* methodology is based on Agile Processes principles and on some software companies experiences, and represents an important contribution to everyone interested in these new software development approaches.

.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

1. Introdução

A evolução dos meios de comunicação observada nas últimas quatro décadas, e caracterizada pelo surgimento das redes de computadores e de tecnologias de transmissão de dados multimídia, provocou mudanças no cenário de desenvolvimento de software.

O propósito original da Internet era possibilitar a troca de dados entre pequenos grupos de cientistas. Além de restrita ao meio acadêmico, essa era uma tecnologia bastante cara. Entretanto, novas pesquisas e investimentos ao longo dos anos têm tornado a Internet acessível à comunidade em geral, por preços cada vez menores.

Com a popularização da tecnologia, a Internet passou a ser utilizada não só como meio de disseminação de informações, mas também como plataforma de comunicação e desenvolvimento, e como canal para a realização de negócios.

Essa mudança de contexto possibilitou o surgimento de duas novas modalidades de desenvolvimento de software: o Desenvolvimento de Aplicações Web e o Desenvolvimento Global de Software.

O Desenvolvimento de Aplicações Web mostrou-se uma excelente oportunidade de negócio para as empresas de software, que poderiam oferecer serviços a usuários finais de todo o mundo. As aplicações atuais servem a uma variedade de propósitos, desde a troca de mensagens em listas de discussão e a participação em jogos on-line, à realização de negócios de comércio eletrônico e transações bancárias [GINIGE, 2001a].

Mas, para que isso fosse possível, foi necessária uma mudança nas características e na arquitetura das aplicações Web. Enquanto as primeiras aplicações se resumiam a um conjunto de poucas páginas com conteúdo estático e, principalmente textual, as aplicações atuais são maiores e mais complexas, com páginas que apresentam conteúdo multimídia e dinâmico, e permitem acessos a bancos de dados e comunicação com sistemas legados [MURUGESAN, 2001].

A rápida multiplicação da oferta de aplicações semelhantes tornou a Web bastante competitiva, e aspectos como confiabilidade, usabilidade e segurança passaram a ser considerados importantes diferenciais de qualidade [OFFUTT, 2002].

Os desenvolvedores também tiveram que se ajustar à velocidade da Web, realizando projetos em espaços de tempo menores, de apenas alguns meses, e adaptando suas aplicações com freqüência para acompanhar as mudanças tecnológicas constantes, e atender às crescentes necessidades do mercado [GLASS, 2001].

Por outro lado, as empresas de software perceberam que poderiam usar a Internet e as tecnologias de comunicação (telefone, fax, vídeo e áudio-conferência, etc) para a execução de projetos à distância.

A idéia do Desenvolvimento Global de Software é que uma aplicação seja construída de maneira concorrente por times dispersos geograficamente, que podem até estar ligados a diferentes empresas, dependendo da parceria de negócios firmada.

As vantagens dessa abordagem são muitas [HERBSLEB, 2001]: o trabalho em paralelo de times seguindo diferentes fusos-horários pode acelerar o projeto; os gastos com mão-de-obra podem ser reduzidos com a contratação de profissionais de países com taxas de remuneração mais baixas; as empresas podem formar parcerias com o objetivo de explorar mercados maiores; as diferentes competências necessárias para o projeto podem ser recrutadas onde quer que elas estejam.

Mas, para que isso funcione harmonicamente, esforços adicionais na gestão do projeto são necessários para sincronizar o trabalho dos times. O controle de versões, o uso compartilhado de ferramentas e a integração dos resultados gerados são os grandes desafios. Além disso, a comunicação entre as equipes pode ser afetada por questões culturais, como idioma e convenções sociais [CARMEL, 1999].

Outro aspecto do Desenvolvimento Global de Software é a produção de aplicações para usuários de todo o mundo. Com a Internet, as fronteiras geográficas deixaram de ser um obstáculo para os negócios, e os mercados locais se transformaram em mercados globais. Havia agora um canal eficiente e econômico para distribuir software para todo o mundo [MURUGESAN, 1999].

Como conseqüência, os desenvolvedores passaram a considerar aspectos culturais na concepção das aplicações, tentando ajustá-las ao perfil dos diferentes usuários. A adaptação do software para diferentes plataformas e idiomas tornou-se uma condição para a entrada no mercado global. Aspectos de usabilidade, como o uso de cores e de ícones, e a escolha de estilos de navegação, também são muito importantes [LUONG, 1995] [GRIBBONS, 1997] [BECKER, 2001].

As semelhanças entre o Desenvolvimento de Aplicações Web e o Desenvolvimento Global de Software estão relacionadas principalmente aos requisitos para as aplicações desenvolvidas. Em ambos os casos, os usuários são bastante heterogêneos, com necessidades culturais próprias e diferentes configurações de hardware e software. Por isso, questões como portabilidade, reuso e manutenibilidade devem ser consideradas desde a concepção do software, para que diferentes versões sejam geradas e atualizadas com esforços e custos reduzidos.

O uso combinado das duas abordagens também constitui uma boa alternativa de negócio para as empresas. Uma vez que a produção de Aplicações Web complexas, como as de comércio eletrônico, por exemplo, requer o envolvimento de profissionais com diferentes competências que nem sempre estão reunidas numa única região, ou mesmo

numa única empresa, a solução pode ser a participação remota de outros profissionais e a formação de parcerias entre organizações.

Ainda considerando as aplicações de comércio eletrônico, é bastante desejável que o maior número possível de usuários possa ter acesso ao sistema, o que implica na necessidade de oferecer versões da aplicação adaptadas às características de um público-alvo bastante heterogêneo, com aspectos culturais (idioma, formato de representação de datas e horários, convenções sociais, etc) que variam de uma região geográfica para outra. A proximidade dos desenvolvedores com os usuários permite a percepção dessas particularidades e facilita a realização de adaptações na aplicação durante e após o desenvolvimento (atividades de manutenção e suporte), constituindo, assim, outra boa razão para o desenvolvimento distribuído.

Outra circunstância que pressupõe o uso conjunto das duas abordagens ocorre quando, devido a uma fusão entre empresas localizadas em diferentes regiões, as unidades resolvem utilizar o desenvolvimento distribuído para continuarem desenvolvendo Aplicações Web sem o deslocamento geográfico dos membros ou dos equipamentos.

É curioso observar ainda que o Desenvolvimento Global de Software só é possível graças ao uso de Aplicações Web que permitem a comunicação e a troca de resultados entre os times. Afinal, ferramentas especiais para a gestão remota de artefatos e para a interação entre as equipes são fundamentais para o acompanhamento e o sucesso dos projetos.

1.1 Motivações

Tanto os projetos de Desenvolvimento de Aplicações Web como os de Desenvolvimento Global de Software apresentam particularidades que devem ser consideradas na escolha da metodologia de desenvolvimento a ser seguida.

No caso das Aplicações Web, o processo deve permitir ciclos de desenvolvimento rápidos, para atender às pressões do *time-to-market*, e deve dar ênfase à fase de manutenção, para acompanhar a mudança de requisitos. Também é importante garantir o atendimento aos requisitos não-funcionais da aplicação (confiabilidade, usabilidade, segurança, etc).

Uma vez que os projetos são desenvolvidos por equipes pequenas e multidisciplinares, envolvendo profissionais de áreas como design, software e computação gráfica [GINIGE, 2001c], é fundamental que o processo escolhido não seja muito burocrático nem complicado, para que possa ser facilmente seguido por aqueles que não estão acostumados com as práticas da Engenharia de Software.

O principal desafio do Desenvolvimento Global de Software é manter os times distribuídos geograficamente sincronizados. Para isso, é preciso integrar pessoas de empresas diferentes, que podem não se conhecer, procurando contornar problemas de comunicação, diferenças culturais e falta de espírito de equipe.

Além disso, as estratégias para a divisão de tarefas e para a integração de código devem ser cuidadosamente planejadas, bem como a realização de testes remotos. O controle das versões do software é um aspecto importante.

Para que as aplicações possam ser disponibilizadas para o mercado global, é preciso que o software seja projetado para o reuso, de modo que as adaptações para as diferentes culturas sejam feitas rapidamente e sem custos extras. Essas atividades devem estar previstas no processo de desenvolvimento adotado.

Como pode ser observado, tanto no caso do Desenvolvimento de Aplicações Web como no Desenvolvimento Global de Software, ocorrem mudanças no ciclo de vida do software. Novas atividades passam a fazer parte do processo, enquanto outras continuam sendo realizadas, mas de maneira diferente.

Quando se trata da escolha do processo de produção do software, os desenvolvedores das duas abordagens têm uma opinião comum: as metodologias tradicionais da Engenharia de Software, como o Modelo Cascata, a Prototipação e o Modelo Espiral [PRESSMAN, 1995], não são adequadas às suas necessidades. Isso porque, além de não incluírem atividades importantes, apresentam uma definição de fases pouco adequada às necessidades do mercado, que precisa de resultados rápidos e que não costuma ter requisitos definidos e estáticos no início do projeto.

A burocracia excessiva dos processos tradicionais, com a produção de um grande número de artefatos, também é contestada pelos desenvolvedores, que precisam de metodologias que possam ser seguidas mais facilmente, e que não tenham impacto sobre o cronograma dos projetos.

Essa é a opinião de diversos autores [AYOAMA, 1998] [BATTIN, 2001] [DESHPANDE, 2001] [EBERT, 2001] [FAVELA, 2001] [GAO, 1999] [GLASS, 2001] [GINIGE, 2001a][GINIGE, 2001b] [HUTCHENS, 1997] [MOCKUS, 2001] [MURUGESAN, 1999] [PRESSMAN, 1998][PRESSMAN, 2001] [TAKAHASHI, 2000], que recomendam a busca de modelos alternativos que reflitam em suas atividades as necessidades de cada área.

É fundamental saber escolher ainda as métricas que serão utilizadas para acompanhar o projeto, e as ferramentas que auxiliarão os desenvolvedores durante a concepção da aplicação. Esses aspectos devem ser antecipados no modelo.

1.2 Objetivos da Dissertação

O objetivo principal desse trabalho é analisar as particularidades do Desenvolvimento de Aplicações Web e do Desenvolvimento Global de Software e apresentar uma proposta de modelo de desenvolvimento para projetos em que a combinação de fatores como complexidade das aplicações desenvolvidas, importância da proximidade com os mercados-alvo, e pressões de mercado para a formação de parcerias levam as empresas a adotarem simultaneamente condutas das duas abordagens.

Para isso, serão gerados dois resultados intermediários: (1) uma análise do cenário de Desenvolvimento Web, considerando as características das aplicações e as necessidades dos desenvolvedores e (2) um mapeamento das mudanças provocadas pelo Desenvolvimento Global de Software nos aspectos de negócio, de desenvolvimento e gerenciais do ciclo de vida das aplicações.

Juntamente com a metodologia proposta, serão sugeridas também métricas e ferramentas de desenvolvimento para dar suporte ao projeto.

Devido a restrições de tempo, a validação do modelo proposto a partir de uma experiência prática não fará parte do escopo desta dissertação, podendo ser realizada futuramente em outros trabalhos. Entretanto, para fundamentar as condutas propostas na metodologia, serão analisados relatos da experiência de grandes empresas como Fujitsu, Motorola e Alcatel, dentre outras, em projetos de desenvolvimento de software por times distribuídos, e também relatos de experiências com o Desenvolvimento de Aplicações Web.

A idéia é reunir as melhores práticas das duas abordagens e propor os ajustes necessários para que essas condutas possam ser combinadas harmonicamente numa única metodologia, considerando um cenário específico semelhante ao vivenciado pelas empresas atuais. Essa análise também evidenciará possíveis falhas das metodologias utilizadas atualmente e lacunas a serem preenchidas pela metodologia *WideWorkWeb* apresentada neste trabalho.

1.3 Relevância

Uma vez que o Desenvolvimento de Aplicações Web e o Desenvolvimento Global de Software são abordagens que estão sendo adotadas em diversos projetos recentes [BATTIN, 2001] [BECKER, 2001] [EBERT, 2001] [GAO, 1999], acredita-se que esse trabalho será de grande interesse para a maioria dos desenvolvedores de software.

A análise das características das Aplicações Web e de seu desenvolvimento poderá ser usada para avaliar com precisão os padrões de qualidade necessários e os riscos envolvidos. Além disso, a descrição de como modelos alternativos de processos, como os métodos Agile [AMBLER, 2002a], podem ser adaptados ao Desenvolvimento de Aplicações Web proporcionará uma variedade maior de opções para a escolha dos desenvolvedores.

Já o mapeamento das mudanças envolvidas no ciclo de vida das aplicações desenvolvidas de maneira distribuída ajudará as empresas a conhecerem as vantagens e desvantagens do Desenvolvimento Global de Software, além de servir como base para uma análise mais concreta das mudanças necessárias para a formação de times virtuais.

Este conjunto de considerações sobre o Desenvolvimento Global de Software será de particular importância para as empresas de software brasileiras como um todo, pois, apesar de existirem diversas fontes de informação disponíveis em outros idiomas, principalmente em inglês, ainda não existem publicações em português sobre o assunto.

A apresentação da metodologia de desenvolvimento proposta representará também uma oportunidade para a realização de novas pesquisas e aprofundamentos no assunto, seja na forma de projetos de iniciação científica ou dissertações de mestrado ou doutorado, visto que essas são áreas de desenvolvimento em expansão e que possuem uma grande riqueza de aspectos a serem analisados.

1.4 Organização da Dissertação

O Capítulo 2 apresenta ao leitor o cenário do Desenvolvimento de Aplicações Web. São analisadas as características das aplicações, bem como sua influência sobre a arquitetura dos sistemas.

Também são descritas as particularidades de seu desenvolvimento, e a adequação de certas metodologias atuais, como os Processos Agile eXtreme Programming e Scrum, às necessidades dos desenvolvedores.

No Capítulo 3, o cenário explorado é o do Desenvolvimento Global de Software. Após considerar as vantagens dessa abordagem, serão apontadas as mudanças necessárias no ciclo de vida do software, bem como os riscos envolvidos nesses projetos.

Ao longo do texto, são indicados os problemas encontrados e as soluções adotadas por grandes empresas, como Alcatel, Motorola, Bell Labs e Fujitsu, na realização de projetos globais.

O Capítulo 4 apresenta uma análise resumida das melhores práticas para o Desenvolvimento de Aplicações Web e o Desenvolvimento Global de Software discutidas nos Capítulos 2 e 3, e aponta razões para o uso conjunto das duas abordagens, justificando a definição de uma metodologia comum.

O Capítulo 5 apresenta, de forma simples e objetiva, a metodologia *WideWorkWeb*, proposta para o Desenvolvimento de Aplicações Web no contexto do Desenvolvimento Global de Software. São apontadas diretrizes para a realização das atividades e para a produção de artefatos ao longo do projeto, bem como métricas de acompanhamento.

No Capítulo 6, são discutidas técnicas para a realização de testes em Aplicações Web, e são sugeridas ferramentas de desenvolvimento para dar suporte ao modelo proposto, de modo a guiar o desenvolvedor em suas escolhas.

Finalmente, no Capítulo 7, encontram-se as conclusões do trabalho, com uma avaliação sobre as contribuições geradas e a indicação de trabalhos futuros a serem realizados para apoiar os profissionais de software no Desenvolvimento de Aplicações Web e no Desenvolvimento Global de Software.

Além dos capítulos, há ainda dois anexos que contêm detalhamentos sobre assuntos importantes. O Anexo A apresenta uma visão geral sobre a Internacionalização e Localização de software, indicando os principais aspectos que devem ser levados em

consideração na hora de decidir adaptar uma aplicação para mercados heterogêneos. Já o Anexo B contém modelos que ilustram alguns dos artefatos da metodologia *WideWorkWeb*, e podem ajudar os desenvolvedores a entender melhor a importância de cada documento.

Capítulo 2

2. Desenvolvimento de Aplicações Web

O crescimento da Web trouxe mudanças para diversos setores. Agindo como um instrumento da globalização, a Web contribuiu para reduzir as distâncias entre os povos e criou infinitas possibilidades de negócios para as empresas, que passaram a dispor de um canal universal para a troca de informações e para a divulgação e comercialização de produtos e serviços, sem limitações de tempo ou espaço.

Mais que uma rede de computadores, a Web tornou-se uma plataforma de comunicação e desenvolvimento. Para aproveitar as novas oportunidades de negócios, as empresas passaram a investir em Aplicações Web complexas e com uma arquitetura bem diferente das primeiras aplicações desenvolvidas [MURUGESAN, 2001] [GINIGE, 2001a][OFFUTT, 2002].

Originalmente, a Web foi criada para servir como um ambiente que possibilitasse a troca de informações entre pequenos grupos de cientistas. Nesse contexto, questões como performance e escalabilidade das aplicações não eram críticas. Além disso, o conteúdo manipulado era essencialmente textual e estático, e as aplicações funcionavam em caráter isolado (*stand-alone*).

Usadas na realização de processos críticos para os negócios, as Aplicações Web atuais precisam ser robustas e projetadas para suportar um grande número de usuários simultâneos. Performance e integração com Bancos de Dados e sistemas legados das empresas são fatores determinantes para o sucesso. O conteúdo manipulado é multimídia e dinâmico, e sua concepção envolve a participação de profissionais de diferentes áreas do conhecimento.

A complexidade e a importância dessas características em uma Aplicação Web variam de acordo com seu propósito. Athula Ginige e San Murugesan [GINIGE, 2001a] classificam as Aplicações Web em sete categorias segundo seu uso (Tabela 2.1 – Categorias de Aplicações Web). É importante destacar que uma Aplicação Web pode apresentar características de várias categorias.

Categoria	Descrição	Exemplos
Informativa	Conjunto de aplicações cujo objetivo é divulgar informações. São baseadas em conteúdo multimídia, que pode ser estático ou dinâmico. Essas aplicações demandam atualizações constantes, e a forma de apresentação do conteúdo também é um diferencial.	Jornais e classificados on-line (http://www.nytimes.com http://www.uol.com.br/fsp) Catálogos de produtos (http://www.philips.com.br) Livros eletrônicos (e-books) (http://www.ebooks.com)
Interativa	Corresponde às aplicações em que o usuário interage com o sistema, fornecendo informações e obtendo respostas adequadas (feedback), contribuindo para uma personalização dos produtos e/ou serviços.	Formulários de autenticação e páginas com conteúdo personalizado (http://www.uol.com.br http://www.yahoo.com) Jogos on-line (http://www.cyberjogos.com)
Transacional	Engloba aplicações que processam os dados dos usuários e realizam transações, principalmente financeiras. A segurança dos dados e a confiabilidade são aspectos críticos nessas aplicações.	*Sites* de comércio eletrônico (http://www.americanas.com.br http://www.amazon.com) Bancos on-line (http://www.bradesco.com.br http://www.bancodobrasil.com.br)
Workflow	Conjunto de aplicações projetadas para permitir o acompanhamento de processos do mundo real. As informações precisam ser atualizadas em curtos intervalos de tempo, de modo a fornecer a visão mais próxima possível da realidade.	Sistemas de planejamento e monitoramento on-line (http://www.detrans.com.br) Sistemas para acompanhamento de encomendas (http://www.correios.com.br)
Comunidades on-line	São aplicações utilizadas para permitir a comunicação entre pessoas, estejam elas dispersas geograficamente ou não. Os grupos costumam ser formados por pessoas com perfis semelhantes, que discutem interesses comuns.	Grupos de *chat* e listas de discussão (http://yahoogroups.com http://www.meugrupo.com.br) Leilões on-line (http://www.ebay.com)
Portais Web	São aplicações que oferecem uma grande variedade de serviços aos usuários (notícias, anúncios, correio eletrônico, etc).	Shopping centers eletrônicos (http://www.uol.com.br http://www.yahoo.com.br)
Ambientes de trabalho colaborativo	Conjunto de aplicações integradas que possibilitam que times de desenvolvedores dispersos geograficamente trabalhem em conjunto na produção de software.	Sistemas distribuídos de desenvolvimento,ferramentas de projeto colaborativas e repositórios de código compartilhados (http://wwwagr.informatik.uni-kl.de/~milos/ http://www.rational.com.br)

Tabela 2.1 – Categorias de Aplicações Web

De um modo geral, as Aplicações Web possuem características especiais que as diferenciam das aplicações convencionais. Aspectos como performance, escalabilidade, confiabilidade e segurança são críticos em Aplicações Web, pois funcionam como diferenciais para atrair usuários num cenário onde as aplicações das empresas concorrentes podem ser acessadas com um simples clique de mouse.

No caso de aplicações desenvolvidas para mercados globais, algumas características precisam ser adaptadas para o perfil dos usuários, como estilos de navegação, interface e plataforma de software.

As características próprias das Aplicações Web também provocam mudanças em sua arquitetura, uma vez que novas camadas de software e novos componentes precisam ser acrescentados para garantir a qualidade dos serviços.

Além disso, o desenvolvimento de Aplicações Web possui algumas particularidades, como a necessidade de times menores e heterogêneos, formados por profissionais de diferentes áreas, que precisam gerar resultados em curtos espaços de tempo. Devido a essas e outras características especiais, alguns especialistas [PRESSMAN, 1998][PRESSMAN, 2000][PRESSMAN, 2001][GLASS, 2001][GINIGE, 2001a][GINIGE, 2001b][DESHPANDE, 2001] alegam que o desenvolvimento Web deveria seguir uma abordagem diferente das metodologias tradicionais da Engenharia de Software. O termo Engenharia Web vem sendo adotado por esses autores para referenciar a nova abordagem.

A seguir, serão descritas as principais características das Aplicações Web que devem ser consideradas pelos desenvolvedores durante sua concepção. Também serão feitas algumas considerações a respeito da escolha da arquitetura das Aplicações Web.

Finalmente, serão analisadas as diferenças entre o desenvolvimento Web e o desenvolvimento tradicional, e os requisitos da Engenharia Web. Também serão apresentados alguns modelos de processos e práticas que estão sendo propostos pela comunidade de software.

2.1 Características das Aplicações Web

Mais que uma coleção de documentos HTML estáticos, as Aplicações Web atuais apresentam conteúdo dinâmico, gerado com o auxílio de scripts interpretados e consultas a bancos de dados remotos. Segundo Offutt [OFFUTT, 2002], o software utilizado para dar suporte a essas aplicações costuma ser distribuído, implementado em múltiplos estilos e linguagens, construído com as mais recentes tecnologias, e baseado em componentes reusáveis.

Como ocorre no desenvolvimento de qualquer software, o atendimento às necessidades dos usuários e a rapidez no lançamento do produto (*time-to-market* reduzido) são fatores muito importantes para o sucesso das aplicações. Entretanto, no caso das Aplicações Web, a qualidade do software, identificada na maioria das vezes por requisitos não-funcionais, adquire uma importância muito maior, já que a concorrência é acirrada e é preciso conquistar o cliente para que ele retorne ao *site* outras vezes.

Uma pesquisa recente [OFFUTT, 2002] indica que, para muitos desenvolvedores, o *time-to-market* deixou de ser a principal preocupação, e fatores como confiabilidade, usabilidade e segurança passaram a ser vistos como determinantes para o sucesso das Aplicações Web. Nesse mesmo estudo, foram apontados ainda os seguintes critérios de

qualidade para as aplicações: acessibilidade, escalabilidade e manutenibilidade. A seguir, algumas considerações sobre cada um desses aspectos:

- **Confiabilidade**

Os desenvolvedores de Aplicações Web devem realizar testes e medições de qualidade para garantir que o software funcionará corretamente na maior parte do tempo. Esse requisito é particularmente importante para aplicações convencionais de tempo real, como controles de tráfego aéreo.

No caso de Aplicações Web, o que se espera é que o usuário possa realizar eficientemente transações de comércio eletrônico ou troca de arquivos. Se a aplicação não for confiável, dados poderão ser perdidos e, conseqüentemente, as empresas perderão também seus clientes e terão prejuízos.

Podem ocorrer falhas na conexão, e é preciso evitar que os dados usados em transações bancárias, ou de compra e venda pela Web, por exemplo, sejam perdidos, ou mesmo que a operação seja realizada apenas parcialmente. Para isso, o desenvolvedor deve procurar garantir a atomicidade das transações, bem como minimizar o tempo de resposta da aplicação e sua taxa de erros. Apresentar mensagens de erro compreensíveis também é uma boa prática para ajudar o usuário.

- **Usabilidade**

No competitivo mundo Web, em que os usuários vêm e vão com a mesma facilidade, a usabilidade – capacidade de permitir que o usuário manipule os recursos do *site* para atingir um determinado objetivo [POWELL, 2000] - é um aspecto diferencial. Os usuários estão cada vez mais exigentes, e sua satisfação deve ser buscada a todo custo.

A usabilidade de uma Aplicação Web corresponde à usabilidade do conjunto de páginas Web que compõem sua interface. Como existem diversas opções de Aplicações Web (na forma de *Web sites*) com serviços semelhantes, os usuários certamente escolherão aquela que seja mais fácil de usar, ou que seja mais adequada a seu perfil.

Becker [BECKER, 2001] aponta alguns fatores estratégicos de usabilidade, que devem ser considerados pelo desenvolvedor na criação de Aplicações Web: perfil dos usuários, *layout*, navegação, identidade visual e diferenças culturais.

Conhecer o perfil dos usuários é o primeiro passo para desenvolver uma Aplicação Web de sucesso. Características como idade, sexo, escolaridade, nacionalidade, profissão e grau de familiaridade com o computador e com a Web devem ser determinantes na definição da interface da aplicação.

O *layout* de uma Aplicação Web corresponde à sua apresentação visual: uso de cores, alinhamento entre os elementos de cada página Web, tamanho do texto e das figuras, etc. A aplicação deve ter uma aparência agradável para que os usuários sintam prazer em usá-la.

Uma boa Aplicação Web deve organizar sua interface de modo que o usuário possa navegar livremente, acessando as páginas Web de acordo com sua necessidade. O desenvolvedor da aplicação deve facilitar esse processo, disponibilizando menus, *links* ou botões que auxiliem na navegação.

Para facilitar o uso da Aplicação Web, Becker recomenda que seja mantida a mesma identidade visual (*look and feel*) entre as páginas Web que compõem sua interface. O objetivo é manter uma consistência entre as cores e a localização de alguns elementos, como menus, entre as páginas para que o usuário não fique confuso.

Graças ao crescimento da Web, a quantidade e a diversidade cultural de usuários de suas aplicações tendem a aumentar cada vez mais. Por isso, criar Aplicações Web personalizadas para cada cultura, considerando idiomas, significado de cores e ícones, e estilos de navegação adequados é fundamental para ser competitivo no mercado global.

- **Segurança**

As Aplicações Web precisam estar protegidas contra ataques de *hackers* e acessos indevidos a bancos de dados com informações sobre os usuários. A privacidade dos usuários deve ser preservada e dados pessoais como número de cartão de crédito, endereço, telefone e renda não devem ser divulgados sem autorização. Algoritmos de criptografia e senhas individuais de acesso são alternativas para evitar que a confiança dos usuários seja perdida.

- **Acessibilidade**

Para que usuários de todo o mundo possam usufruir seus serviços, as Aplicações Web precisam estar acessíveis 24 horas por dia, durante todo o ano, sob pena de perderem oportunidades de negócios. Por isso, investimentos em hardware e robustez das aplicações são fundamentais para reduzir o tempo de indisponibilidade dos serviços. Condutas de tolerância a falhas devem ser consideradas.

Outra forma de evitar que alguns usuários não tenham acesso às Aplicações Web é projetá-las para serem executadas em diferentes *browsers*, e com diferentes taxas de transmissão, uma vez que os usuários globais são heterogêneos. Uma alternativa é disponibilizar diferentes versões da aplicação.

Especialmente no caso de aplicações para comércio eletrônico, em que se espera atingir o maior número possível de usuários, não é desejável que uma parte

deles não tenha acesso à aplicação porque seu *browser* não suporta *frames*, ou porque ele não possui o *plug-in* necessário para visualizar uma determinada animação.

- **Escalabilidade**

Os desenvolvedores de Aplicações Web devem projetar sistemas preparados para suportar um número crescente de usuários. Como o público-alvo dessas aplicações é global, trata-se de milhares, ou até milhões de pessoas usando os recursos da aplicação e/ou realizando transações concorrentemente.

Quando o limite de acessos simultâneos do sistema é atingido, a ocorrência de falhas, como perda de dados e transações incompletas, e a indisponibilidade temporária do sistema são possíveis conseqüências, que devem ser evitadas a todo custo pelos desenvolvedores.

- **Manutenibilidade**

Uma característica importante da Web é a freqüência com que os requisitos das aplicações mudam. As empresas estão constantemente realizando atualizações no conteúdo de seus *sites*, e adaptando suas soluções para tecnologias mais modernas para não ficarem obsoletas.

Como o conteúdo é o aspecto mais importante nas Aplicações Web [GLASS, 2001], uma boa estratégia para facilitar atualizações e adaptações arquiteturais seria isolar a lógica da aplicação de sua forma de apresentação, diminuindo assim o impacto das mudanças [OFFUTT, 2002]. Além disso, essa é uma forma de facilitar o trabalho dos desenvolvedores, já que as equipes costumam ser compostas por programadores (responsáveis pelo software) e por profissionais de design e/ou comunicação (responsáveis pelo conteúdo e pela interface gráfica). O Desenvolvimento Baseado em Componentes reusáveis e o uso de páginas Web geradas dinamicamente ajudam os desenvolvedores nessa estratégia.

- ***Time-to-market***

Apesar dos aspectos de qualidade do software estarem sendo considerados determinantes, o *time-to-market* ainda é um aspecto crítico para as Aplicações Web. As mudanças tecnológicas não param de acontecer, e é preciso acompanhar as necessidades dos usuários e as tendências do mercado para tentar antecipá-las. Muitas vezes, manter-se competitivo significa reformular totalmente uma aplicação, e isso precisa ser feito no menor espaço de tempo possível. No caso de aplicações de caráter informativo, as atualizações de conteúdo precisam ser feitas em questão de minutos.

Toda essa pressão para reduzir o tempo de desenvolvimento acaba tendo impacto sobre a forma como os projetos são realizados. Os processos de

desenvolvimento e os mecanismos de gerência devem ser adaptados para suportar cronogramas reduzidos e mudanças tecnológicas constantes.

Além de todos esses aspectos associados à qualidade, algumas questões legais, culturais e morais devem ser consideradas pelos desenvolvedores de Aplicações Web. As leis para a comercialização de bens e a divulgação de informações variam de país para país, e as empresas precisam estar atentas a essas diferenças para evitar desentendimentos e problemas com os usuários.

2.2 Arquitetura das Aplicações Web

Com a disseminação do uso da Internet, as Aplicações Web passaram a ser usadas não só para divulgar informações, mas também para permitir manipulação de dados e gerenciamento remoto de sistemas, dentre outras facilidades.

Essa oferta adicional de serviços provocou uma evolução da estrutura das Aplicações Web [MURUGESAN, 2001] [GINIGE, 2001a][OFFUTT, 2002]. Antes, elas se resumiam a um conjunto de poucas páginas com conteúdo estático e, principalmente textual. Hoje, suas inúmeras páginas apresentam conteúdo multimídia, gerado dinamicamente através de scripts, e permitem acessos a bancos de dados e comunicação com sistemas legados.

Com o aumento da complexidade das aplicações, e a importância crescente de seus requisitos não-funcionais (confiabilidade, segurança, escalabilidade, etc) como indicadores de qualidade, a arquitetura dos sistemas baseados na Web precisou ser modificada.

A antiga arquitetura cliente-servidor, bastante centralizada e caracterizada pela comunicação direta entre apenas duas máquinas, apresentava limitações que não poderiam ser toleradas em sistemas Web. Dentre elas [OFFUTT, 2002]:

- Problemas de segurança: todos os recursos ficavam concentrados em uma única máquina (servidor), tornando o sistema mais propenso a ataques. No caso dos sistemas transacionais atuais, com operações bancárias e de comércio eletrônico, essa vulnerabilidade poderia representar o fracasso da aplicação.

- Problemas de escalabilidade e acessibilidade: concentrar o atendimento a todas as solicitações de um *site* em uma única máquina significa limitar a quantidade de acessos simultâneos, com a formação de um gargalo.

- Problemas de manutenibilidade: como todos os recursos ficavam concentrados no servidor, não havia separação entre a lógica da aplicação e sua forma de apresentação, dificultando as alterações necessárias.

Como alternativa a esse modelo, surgiu a arquitetura em N-camadas [ALMEIDA, 2003][OFFUTT, 2002][CONALLEN, 1999]. O objetivo era permitir que outras máquinas ou camadas fossem acrescentadas, isolando os diferentes aspectos das aplicações (interface,

lógica do negócio e dados) e descentralizando o processamento. Assim, as limitações do modelo cliente-servidor poderiam ser superadas.

As principais camadas desse modelo podem ser vistas na Figura 2.1 (Arquitetura em N-Camadas): a Interface do Sistema, a Lógica da Aplicação, a Camada de Dados e os Sistemas Legados [ALMEIDA, 2003].

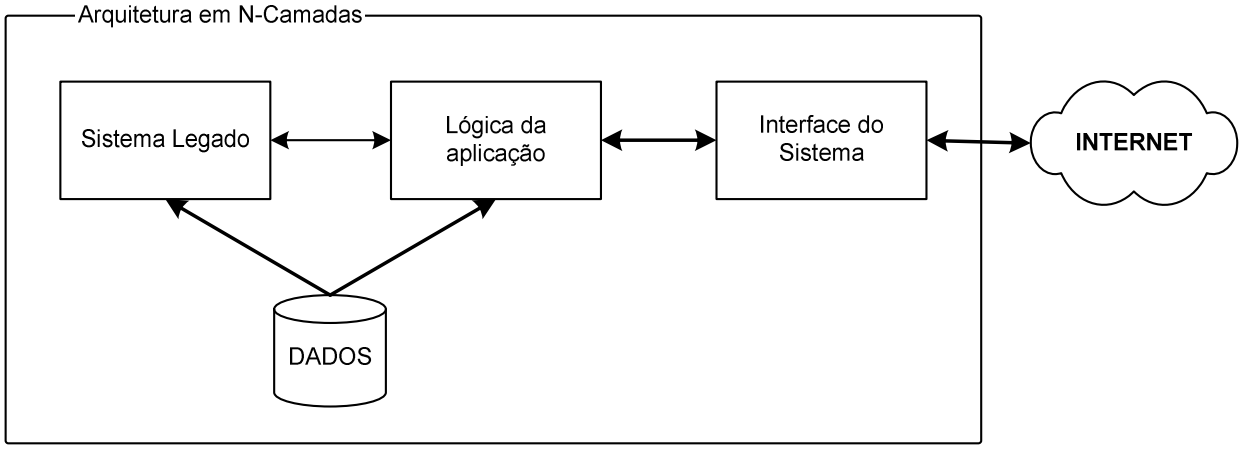

Figura 2.1 – Arquitetura em N-Camadas

A Interface do Sistema permite que os usuários acessem os recursos da aplicação, através da navegação pelas páginas do *site*. A comunicação entre a máquina cliente e o servidor Web utiliza o protocolo HTTP, que garante a robustez e a tolerância a falhas, pois não permite conexões permanentes: a conexão é iniciada quando ocorre uma solicitação do *browser*, e é encerrada tão logo a página Web seja enviada pelo Servidor Web.

Quando uma conexão segura é necessária, o protocolo HTTPS pode ser usado, pois utiliza criptografia por chave pública/privada para os dados transmitidos. Ainda para garantir a segurança dos dados, é possível utilizar um software chamado *Firewall* para bloquear acessos indevidos.

Quando uma solicitação é feita pelo *browser* cliente, o Servidor Web encarrega-se de atendê-la. Se a página pedida for estática (código HTML puro), ela simplesmente será exibida. Caso seja uma página dinâmica, que contenha scripts (Ex: Active Server Pages, Java Server Pages, Servlets, Cold Fusion) ou módulos executáveis (Ex: Microsoft ISAPI ou Netscape NSAPI), o Servidor Web acionará a camada lógica da aplicação para realizar seu processamento.

A existência de uma camada especializada na interface permite que o sistema tenha suporte a vários tipos de dispositivos de acesso, como *browsers*, celulares, interfaces gráficas, etc.

É na camada Lógica da Aplicação que são executadas as regras do negócio pelo Servidor de Aplicação. Essa camada fornece dados para a geração de interfaces e se comunica também com sistemas legados para expandir serviços já existentes.

Separar a lógica da aplicação de sua interface não só facilita a manutenção do sistema, mas também permite uma maior flexibilidade para acréscimo de funcionalidades ao sistema. O uso de linguagens que possibilitem o reuso, como Java, é outra estratégia para simplificar as alterações na aplicação.

A Camada de Dados, também chamada de persistência, corresponde a um conjunto de tecnologias que podem ser usadas para armazenar os dados da aplicação. O acesso aos dados pode ser feito diretamente através de comandos no código das páginas dinâmicas, usando bibliotecas ODBC, JDBC, etc.

Sistema Legado é o nome dado a qualquer sistema já existente que precise ser integrado à Aplicação Web para fornecer dados ou realizar processamentos. Para evitar que o código das aplicações existentes precise ser reescrito, ou que uma tecnologia específica precise ser adotada para integrar a Aplicação Web ao legado da empresa, é recomendável o uso de uma tecnologia conhecida como *Web Services* [ALLEN, 2002][STAL, 2002].

A proposta é usar um conjunto de padrões abertos, composto por XML(*eXtensible Markup Language*), SOAP (*Simple Object Access Protocol*), WSDL (*Web Services Description Language*) e UDDI (*Universal Description, Discovery, and Integration*), para garantir a interoperabilidade entre as aplicações. Essa iniciativa já conta com a aprovação de grandes empresas internacionais, como Microsoft, IBM, Sun Microsystems e Hewlett-Packard, que estão adotando os padrões de Web Services em suas soluções [NICHOLS, 2002]. Exemplos de soluções comerciais que utilizam o padrão Web Services são o **.NET**, da Microsoft, e o **Sun ONE**, da Sun Microsystems.

Na Figura 2.2 (Arquitetura detalhada do modelo N-Camadas) são apresentados em mais detalhes os elementos da arquitetura em N-camadas, bem como indicações de tecnologias atualmente disponíveis no mercado [ALMEIDA, 2003].

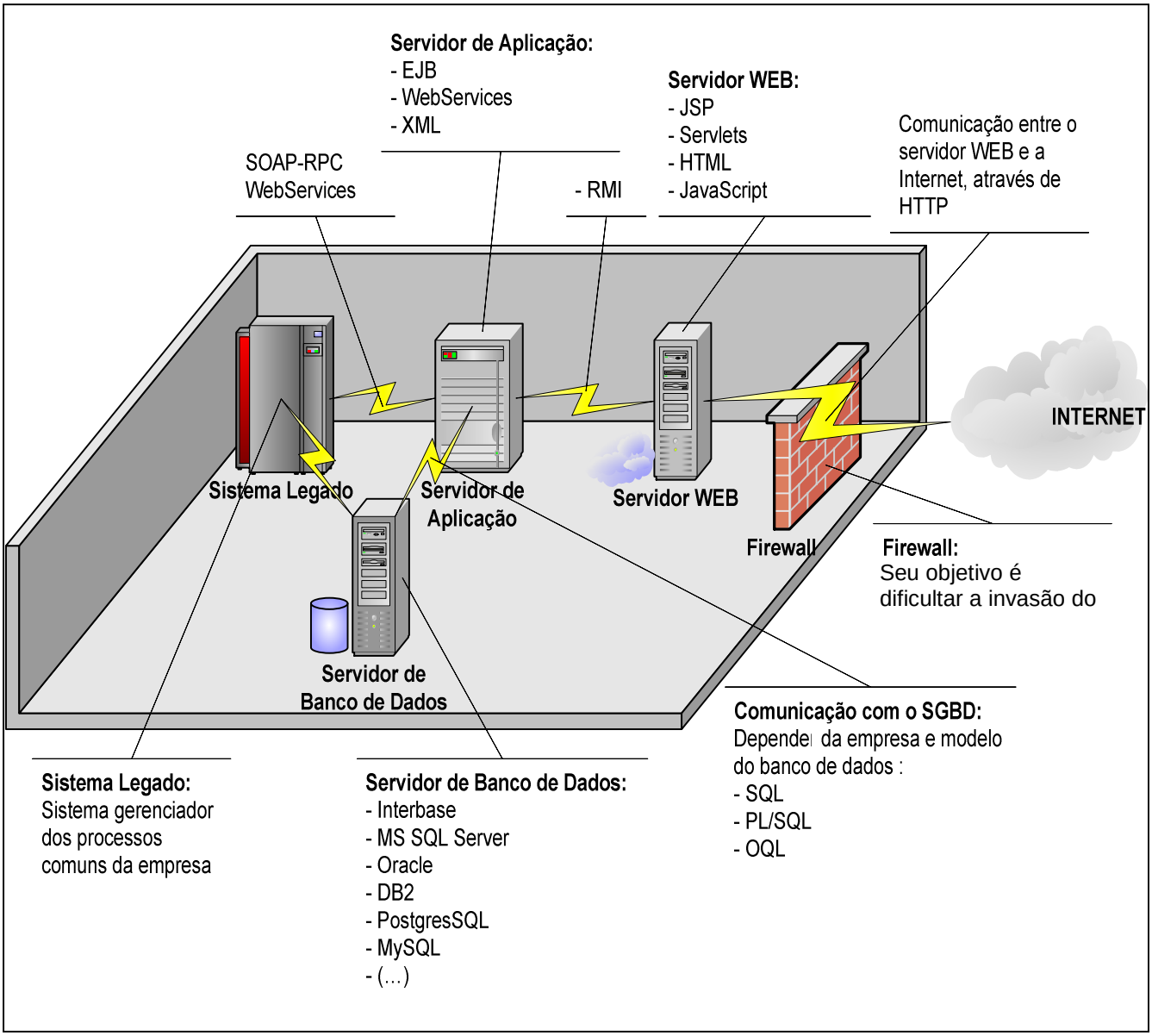

Figura 2.2 – Arquitetura detalhada do modelo N-Camadas

2.3 Engenharia Web – Uma Nova Abordagem de Desenvolvimento

O diferencial das Aplicações Web não está apenas em suas características, mas principalmente nas particularidades de seu desenvolvimento. Vários artigos [PRESSMAN, 1998][PRESSMAN, 2000][PRESSMAN, 2001][GLASS, 2001][GINIGE, 2001a][GINIGE, 2001b][DESHPANDE, 2001] apontam as principais diferenças: requisitos indefinidos e mutáveis, times pequenos e multidisciplinares, ciclos de desenvolvimento menores, e necessidade constante de manutenção.

- **Requisitos indefinidos e mutáveis**

Desenvolver software para a Web significa considerar um cenário extremamente competitivo, onde ocorrem mudanças o tempo todo e as aplicações ficam obsoletas numa velocidade assustadora. Novas tecnologias não param de surgir, e as necessidades dos clientes evoluem num ritmo igualmente acelerado.

A especificação do sistema precisa ser feita rapidamente, mesmo quando os clientes não conhecem os recursos da Web e não sabem exatamente o que querem. Se os desenvolvedores passarem muito tempo analisando os requisitos da aplicação para gerar uma especificação completa, o software corre um sério risco de se tornar inadequado antes mesmo de ser finalizado [LOWE, 2001].

No início do projeto, é provável que o cliente não consiga expressar claramente o que espera da aplicação, principalmente por não conhecer os recursos das novas tecnologias. Nesses casos, um protótipo com a estrutura das páginas do *site* e suas possíveis funcionalidades pode ser gerado rapidamente para guiar o cliente nas decisões funcionais e de interface. O desenvolvimento é então iniciado, e à medida que novas funcionalidades ou mudanças tecnológicas se tornam necessárias, o código vai sendo modificado para acomodar novas demandas.

Essa é uma grande diferença das abordagens tradicionais da Engenharia de Software, que recomendam o início do desenvolvimento apenas quando a especificação do sistema (ou a maior parte dela) já estiver pronta. Tal conduta seria justificada com a redução dos custos de mudança nas fases mais avançadas do desenvolvimento. No caso das Aplicações Web, a mudança nos requisitos é uma questão inerente ao processo, e precisa ser tratada com atenção.

Segundo Jim Highsmith [HIGHSMITH, 2002], “as metodologias tradicionais foram desenvolvidas para construir software em cenários caracterizados por níveis baixos ou médios de mudanças, e resultados razoavelmente previsíveis. Entretanto, o mundo dos negócios não é mais tão previsível, e os requisitos de software mudam numa freqüência que sobrecarrega os métodos tradicionais”.

❖ **Times pequenos e multidisciplinares**

A composição dos times também é diferente no desenvolvimento de Aplicações Web. Como os projetos costumam ser desenvolvidos no espaço de poucos meses, a recomendação é limitar a equipe a um número pequeno de participantes, para diminuir a complexidade gerencial e agilizar a comunicação entre eles [HOHMANN, 2001].

As equipes de desenvolvimento Web costumam ser bastante heterogêneas, pois são constituídas por especialistas em diferentes áreas: engenharia de requisitos; design gráfico; codificação, teste e simulação; projeto de interfaces; armazenamento e indexação de informações; projeto e manutenção de redes de computadores; manipulação de arquivos multimídia; gerenciamento de projeto; etc. Uma vez que os times costumam ser pequenos, é provável que um mesmo desenvolvedor tenha que acumular funções num projeto.

Essa necessidade de profissionais com perfis diversificados torna mais difícil a adequação do desenvolvimento Web aos processos tradicionais da Engenharia de Software, que costumam definir um conjunto de etapas e uma variedade de artefatos pouco significativos para o cenário Web. Como nem todos os profissionais são programadores, nem tampouco engenheiros de software, fica difícil convencê-los a adotar processos complexos, principalmente em projetos de apenas alguns meses.

Quanto mais simples o fluxo de atividades a ser seguido e quanto menor a quantidade de documentação a ser gerada, mais fácil assegurar que o processo será efetivamente respeitado.

A própria conduta para a organização dos times, com um número reduzido de integrantes trabalhando colaborativamente num mesmo espaço físico e num pequeno intervalo de tempo, favorece a substituição da comunicação formal pela comunicação informal, minimizando o uso de documentos. Ou seja, as dúvidas sobre o projeto ou as decisões de planejamento podem ser discutidas em conversas objetivas, sem intermediários e sem margem para interpretações incorretas.

❖ **Ciclos de desenvolvimento menores**

A pressão do *time-to-market* também provoca mudanças no ciclo de desenvolvimento das Aplicações Web. Embora este seja um aspecto relevante na construção de qualquer software, na Web apresentar soluções inovadoras representa um diferencial crítico para as empresas. Como as aplicações passam por constantes mudanças ao longo do desenvolvimento, as empresas não podem esperar que elas sejam finalizadas antes de colocá-las no mercado, pois estarão correndo um sério risco de serem surpreendidas pela concorrência ou por uma evolução tecnológica.

A principal questão não é gerar uma aplicação completa após um longo período, mas ir gerando resultados continuamente, em curtos intervalos de tempo, à medida que vão sendo esclarecidos os requisitos do cliente e do mercado.

Os ciclos de vida tradicionais, compostos por fases seqüenciais que podem se estender por anos, precisam ser quebrados em ciclos de poucos dias ou meses, em que conjuntos de funcionalidades vão sendo gerados e integrados à aplicação existente.

O importante é entregar freqüentemente resultados que agreguem valor ao negócio do cliente, de modo que a evolução da aplicação seja percebida durante todo o processo, e não apenas no final. Essa é uma forma também de permitir que a aplicação vá sendo concebida de acordo com as mudanças dos requisitos, garantindo assim o sucesso do produto final.

Para isso, os chamados processos pesados devem dar lugar a processos mais leves, que sejam menos formais e mais adaptativos a esse cenário em constante mutação [FOWLER, 2000].

- **Necessidade constante de manutenção**

Como a manutenção de Aplicações Web ocorre freqüentemente, seja para atualizar conteúdo, integrar novas funcionalidades ou migrar de uma tecnologia para outra, os desenvolvedores precisam ainda criar soluções flexíveis e adaptáveis para diminuir o impacto dessas modificações no *time-to-market* do produto [LOWE, 2001]. O Desenvolvimento Baseado em Componentes costuma ser a abordagem preferida pelos times para solucionar essas questões [HARMON, 1998] [BRERETON, 2000] [REPENNING, 2001] [GINIGE, 2001a] [LOWE, 2001] [OFFUTT, 2002].

Facilitar a manutenção das Aplicações Web é um grande desafio para os desenvolvedores, uma vez que a mudança constante nos requisitos torna difícil fixar uma arquitetura para a aplicação. Assim, para garantir os requisitos não-funcionais responsáveis pela qualidade da aplicação, os desenvolvedores precisam primar pela adaptabilidade das soluções que vão sendo geradas.

Também é importante destacar que, apesar do desenvolvimento inicial das Aplicações Web ser realizado num espaço de poucos meses, a manutenção continua sendo feita até que a aplicação não esteja mais disponível aos usuários. Assim, a equipe responsável pela manutenção pode não ser a mesma que desenvolveu a aplicação inicial.

Por isso, a produção de uma documentação mínima, com informações sobre o sistema e sobre o projeto, é fundamental para que a transição entre equipes, ou mesmo a entrada de novos participantes no projeto, ocorra rapidamente. A comunicação informal pode funcionar para pequenas aplicações, quando o código ainda é pequeno e seu funcionamento pode ser entendido com facilidade.

Entretanto, à medida que o sistema evolui, é preciso manter uma documentação consistente para que o conhecimento a ser absorvido pelos novos integrantes não dependa apenas da memória dos desenvolvedores mais antigos.

Essas e outras diferenças em relação às abordagens tradicionais de Engenharia de Software criaram uma polêmica entre os especialistas: seria o desenvolvimento de Aplicações Web tão inovador a ponto de exigir uma abordagem completamente nova, que não seguisse as técnicas conhecidas e utilizadas por tantos anos?

Alguns autores [GINIGE, 2001b][PRESSMAN, 2000][DESHPANDE, 2001] comparam a situação atual à crise de software ocorrida no início dos anos 70, quando abordagens como o Modelo Cascata, a Prototipação e o Modelo Espiral [PRESSMAN, 1995] ainda não estavam em prática. Por não disporem de abordagens adequadas às suas necessidades, muitos desenvolvedores da época usavam práticas *ad hoc* de desenvolvimento, sem rigor e sem controles de qualidade.

A crise Web atual ocorre pelas mesmas razões: os desenvolvedores Web julgam as características de seu desenvolvimento tão peculiares que, não tendo uma abordagem adequada, com uma definição correta para as fases e para os artefatos necessários, preferem não adotar nenhuma metodologia em especial e vão realizando o projeto de acordo com os acontecimentos [PRESSMAN, 2001][GLASS, 2001].

A questão é polêmica e ainda não existe um consenso entre os especialistas. As opiniões são bastante conflitantes, e alguns desenvolvedores chegam a considerar o desenvolvimento de Aplicações Web uma espécie de arte, que, portanto, não deve seguir padrões ou formalismos, mas deve continuar sendo realizado de maneira *ad hoc* [GINIGE, 2001a].

Entretanto, a opinião predominante é de que o desenvolvimento de Aplicações Web realmente apresenta particularidades que não podem ser ignoradas, mas deve seguir um processo de engenharia como ocorre com os produtos de software convencionais [PRESSMAN, 1998].

Essa nova abordagem de desenvolvimento recebeu o nome de Engenharia Web, um termo proposto por San Murugesan e adotado em diversas publicações [PRESSMAN, 1998][PRESSMAN, 2000][PRESSMAN, 2001] [GINIGE, 2001a][GINIGE, 2001b] [GINIGE, 2001c][GLASS, 2001][DESHPANDE, 2001].

Segundo San Murugesan, citado em [DESHPANDE, 2001], a Engenharia Web pode ser definida como a aplicação de uma abordagem sistemática, disciplinada e quantificável ao desenvolvimento, operação e manutenção das Aplicações Web.

Novas tarefas precisam ser incluídas no ciclo de desenvolvimento, novas ferramentas são necessárias, e as técnicas tradicionais precisam ser adaptadas para a velocidade da Web, mas muitos dos ensinamentos anteriores ainda são aplicáveis. As novas práticas da Engenharia Web não representam uma negação das conhecidas práticas da

Engenharia de Software, mas sim uma evolução dessa abordagem para um novo cenário com características específicas [GLASS, 2001].

A seguir, serão observadas algumas práticas, presentes em processos de desenvolvimento usados atualmente, que podem ser adequadas às necessidades do cenário Web.

2.3.1 PROCESSOS AGILE

Processos Agile [AMBLER, 2002a] ou Processos Leves [FOWLER, 2000] são metodologias de desenvolvimento adaptativas e flexíveis, e que são indicadas para cenários onde a mudança de requisitos é constante e os resultados precisam ser entregues ao cliente em curtos espaços de tempo.

A proposta é dividir o desenvolvimento de software em ciclos curtos, ou iterações, de apenas algumas semanas, de modo que, ao final de cada ciclo, o cliente receba uma aplicação que agregue valor ao seu negócio. Assim, os desenvolvedores podem acompanhar a mudança dos requisitos no início de cada ciclo, além de ter um feedback contínuo do cliente, reduzindo assim os riscos do projeto.

Enquanto as metodologias tradicionais de desenvolvimento mantêm o foco na geração de documentação sobre o projeto e no cumprimento rígido de processos, a proposta Agile é concentrar as atenções na produção de código e nas relações entre os participantes. A fase de planejamento é reduzida para que os desenvolvedores possam se concentrar em cada iteração em vez de traçar diretrizes para o projeto como um todo.

De um modo geral, os Processos Agile defendem quatro valores fundamentais no desenvolvimento de software: Comunicação, Simplicidade, Feedback e Coragem [AMBLER, 2002a].

- **Comunicação**

  Os processos Agile defendem o fortalecimento da comunicação face-a-face entre os desenvolvedores, de modo que a necessidade de documentação formal para esclarecimento de questões seja minimizada, e, conseqüentemente, o tempo do projeto não seja afetado. Uma vez que as equipes costumam ser formadas por poucos participantes trabalhando fisicamente juntos, a comunicação direta ocorre naturalmente.

  No caso do Desenvolvimento Web, em que os times são multidisciplinares, substituir o uso de modelos formais de documentação pela comunicação informal sempre que possível é uma boa forma de evitar desentendimentos sobre notações e diagramas, que nem sempre são do conhecimento de toda a equipe.

Além disso, a comunicação com o cliente também é intensificada para que suas necessidades sejam conhecidas e as dúvidas sejam esclarecidas tão logo apareçam. Essa é uma prática bastante eficiente para acompanhar a evolução dos requisitos.

- **Simplicidade**

Uma vez que os requisitos estão evoluindo o tempo todo, é recomendável que os desenvolvedores simplifiquem ao máximo o trabalho feito, procurando atender apenas às solicitações imediatas do cliente.

Funcionalidades extras, adicionadas na tentativa de antecipar mudanças, além de aumentarem o tempo de desenvolvimento, a complexidade e os custos, podem incluir defeitos no código ou dificultar mudanças futuras.

O segredo é deixar o código o mais simples possível, de modo que atenda aos requisitos e possa ser facilmente entendido por toda a equipe. Assim, a evolução do software acontecerá naturalmente, à medida que os novos recursos necessários sejam acrescentados. Ou seja, é preciso manter o foco no problema que precisa ser resolvido a cada iteração.

Essa é uma prática bastante útil para reduzir o *time-to-market* das aplicações.

- **Feedback**

A produção iterativa de software, com ciclos curtos e resultados intermediários, permite que os desenvolvedores tenham feedback constante dos usuários. Assim, a qualidade do software resultante pode ser melhorada, e a equipe se sente mais motivada ao perceber que está progredindo. Essa é também uma forma eficiente de reduzir os riscos em projetos com cenários instáveis.

- **Coragem**

Apesar de serem mais simples e menos burocráticos que os processos tradicionais, os Processos Agile não devem ser vistos como pouco rigorosos e fáceis de se aplicar.

Afinal, é preciso haver muita confiança e entrosamento entre as equipes para que um projeto baseado mais na comunicação informal que na documentação possa cumprir prazos e orçamentos.

Além disso, os desenvolvedores precisam ter coragem para adotar a simplicidade, mantendo o foco no presente sem considerar as mudanças que podem ser necessárias para adequar o software no futuro. É preciso ter coragem para crer que a melhor documentação para um projeto de software é o código e que a produção de artefatos adicionais pode ser perda de tempo.

Em suma, é preciso ter coragem para acreditar que práticas tão simples, que mais parecem senso comum, podem substituir com eficiência os modelos tradicionais de desenvolvimento praticados há anos.

Os Processos Agile vêm conquistando adeptos, embora ainda haja poucas evidências de que funcionem na prática [AMBLER, 2002b][MAURER, 2002]. No caso do desenvolvimento de Aplicações Web, os Processos Agile atendem as necessidades de ciclos de desenvolvimento menores, equipes pequenas e requisitos instáveis.

Já no caso da manutenção constante, os desenvolvedores podem encontrar dificuldades por não anteciparem mudanças tecnológicas que tenham impacto sobre a arquitetura do sistema. A manutenção também pode ser dificultada pela falta de documentação descritiva sobre o sistema, já que a equipe responsável pode não ser a mesma que desenvolveu a aplicação inicial.

Outro problema do uso de Processos Agile no desenvolvimento Web é que eles são bastante genéricos e não incluem em seus ciclos de vida atividades relacionadas a questões específicas das aplicações. Nesse caso, algumas condutas sugeridas nos Processos Agile poderiam ser aplicadas em conjunto com técnicas mais apropriadas que atendessem as peculiaridades do domínio do problema. Métricas e testes próprios para a Web também são necessários.

A seguir, uma visão geral sobre duas abordagens Agile discutidas pela comunidade de software nos dias atuais: eXtreme Programming (XP) e Scrum.

2.3.1.1 eXTREME PROGRAMMING (XP)

Das abordagens Agile, *eXtreme programming*, ou XP, proposta por Kent Beck [BECK, 2000], certamente é a mais difundida atualmente. Como o próprio nome sugere, a proposta é tentar simplificar ao extremo as práticas do desenvolvimento de software para aumentar a produtividade [NOYES, 2002].

Além dos quatro valores da filosofia *Agile*, XP defende ainda a adoção conjunta de 12 práticas de desenvolvimento para garantir a qualidade do software e o atendimento dos requisitos do cliente. São elas: planejamento, pequenas entregas, metáfora, projeto simples, testes, refatoramento, programação em pares, propriedade coletiva, integração contínua, 40 horas semanais, presença do cliente, padrões de codificação [BECK, 2000] [HIGHSMITH, 2002] [MAURER, 2002].

- **Planejamento:** as prioridades do negócio e a realidade dos desenvolvedores são consideradas na definição do escopo e do cronograma para cada iteração.

- **Pequenas entregas:** o objetivo é fornecer ao cliente versões incrementais do software em curtos espaços de tempo, chamados iterações, de modo que ao final de algumas semanas ou meses o cliente receba sempre resultados que agreguem valor ao seu negócio. Essa é também uma forma de reduzir os riscos do projeto,

acompanhando a mudança de requisitos, e dos desenvolvedores obterem feedback constante do cliente.

- **Metáfora:** é uma representação geral do funcionamento do sistema, com aspectos sobre sua arquitetura e seus requisitos funcionais e não-funcionais, compartilhada por toda a equipe de desenvolvimento.

- **Projeto simples:** o sistema construído deve ser o mais simples possível, evitando-se que seja acrescentada complexidade extra com recursos não solicitados pelo cliente. A simplicidade do código, sem redundâncias e com um número mínimo de classes e métodos, favorece seu entendimento por todos os desenvolvedores.

- **Testes:** devem ser escritos antes mesmo do código para garantir sua qualidade. Enquanto os desenvolvedores escrevem testes de unidade, os clientes definem testes funcionais para verificar se o sistema apresenta as características pretendidas. Os testes devem ser automatizados de modo que a adequação do software produzido em relação aos requisitos iniciais possa ser verificada. Essa é uma vantagem sobre a documentação escrita.

- **Refatoramento:** à medida que novas funcionalidades vão sendo acrescentadas, é preciso revisar o código para remover redundâncias e complexidade. O funcionamento do sistema não é afetado nessa atividade, apenas a estrutura do código é reformulada, para que ele permaneça claro o suficiente para servir de documentação para o projeto.

- **Programação em pares:** os desenvolvedores trabalham em duplas, dividindo a mesma máquina para a produção de código. Enquanto um dos programadores escreve o código, o outro faz a revisão e discute modificações, garantindo assim uma maior qualidade para o software e uma maior satisfação para a equipe como um todo.

- **Propriedade coletiva:** todos os desenvolvedores têm responsabilidade pelo código do sistema, e possuem permissão para alterar qualquer trecho dele sempre que julgarem necessário. O uso de testes automatizados dá aos desenvolvedores mais segurança na hora de modificar o código, pois qualquer alteração equivocada pode ser facilmente percebida.

- **Integração contínua:** à medida que as funcionalidades vão sendo desenvolvidas, elas são integradas ao código já pronto do sistema e são realizados testes para verificar a existência de erros. A integração pode ocorrer várias vezes ao dia, caracterizando a evolução incremental do software, e garantindo que sempre se tenha uma versão executável e atualizada do programa.

- **40 horas semanais:** jornadas extras de trabalho devem ser evitadas para não comprometer a produtividade e a motivação dos desenvolvedores.

- **Presença do cliente:** o cliente trabalha junto com os desenvolvedores ao longo de todo o projeto, e é encarregado de determinar as características do sistema, esclarecer dúvidas e definir testes funcionais. Sendo assim, a comunicação entre cliente e desenvolvedores é direta, sem a presença de intermediários.

- **Padrões de codificação:** devem ser adotados por toda a equipe para facilitar a compreensão do código e a comunicação entre os desenvolvedores. O código deve ser o mais claro possível para minimizar a necessidade de documentação adicional.

De um modo geral, as práticas XP enfatizam três aspectos: a satisfação do cliente, a qualidade do software produzido, e a organização do processo de desenvolvimento [MAURER, 2002].

No começo do projeto, o cliente se reúne com os desenvolvedores para definir o que será feito e em quanto tempo. São escritas conjuntamente *user stories* (descrições textuais de use-cases) que definem os requisitos funcionais do sistema. A implementação de cada *user story* é estimada pelos desenvolvedores, e o cliente define, de acordo com as prioridades do negócio, o que será feito em cada iteração.

O desenvolvimento é iniciado e as equipes fazem um detalhamento das *user stories*, na forma de uma lista de atividades necessárias para seu funcionamento. Cada par de desenvolvedores escolhe, então, as atividades que vai desempenhar e faz estimativas de tempo, de modo que todas as duplas tenham a mesma carga de trabalho.

Cada dupla escreve testes de unidade para as atividades, antes mesmo de começar a codificação. Essa é também uma forma de planejamento, para que o código só comece a ser escrito quando não houver dúvidas sobre o que deve ser feito. A produção e revisão de código são feitas pela dupla.

Uma vez finalizado, o código é integrado ao restante da aplicação e são feitos os testes. Caso algum teste falhe, os desenvolvedores desfazem a alteração e voltam a trabalhar em seu código para remover os erros. Caso contrário, uma nova versão executável do software, já com as novas funcionalidades, é disponibilizada para toda a equipe.

Durante todo o desenvolvimento, um membro da equipe acompanha o progresso das atividades, liderando reuniões rápidas e diárias, onde cada participante relata os avanços feitos.

Ao final da iteração, o software é submetido aos testes funcionais definidos pelo cliente e implementados pelos desenvolvedores para comprovar a eficiência da *user story*. Caso sejam encontrados defeitos, ou sejam necessárias melhorias, esses aspectos são considerados como requisitos nas iterações seguintes.

No início de cada iteração, o cliente revisa o que foi definido anteriormente e pode decidir acrescentar, excluir ou redefinir prioridades para as *user stories*. De acordo com as mudanças, são feitas novas estimativas e o processo se repete ciclicamente.

A relação entre a simplicidade no projeto defendida por XP e o custo das mudanças ao longo do ciclo de desenvolvimento é uma questão discutida por alguns especialistas [HIGHSMITH, 2002][NOYES, 2002]. Afinal, os processos tradicionais assumem que o custo das mudanças cresce exponencialmente ao longo do desenvolvimento, e por isso destacam a necessidade de uma fase de planejamento mais detalhada, onde todos os aspectos do software possam ser analisados, antes que a codificação seja iniciada.

No caso de XP, uma abordagem de processo para lidar com requisitos instáveis, a recomendação é justamente o contrário: não perder tempo tentando antecipar mudanças e apostar na simplicidade como forma de acomodar as alterações futuras.

Kent Beck [BECK, 2000] alega que essa é uma estratégia válida, pois os custos da mudança podem não crescer exponencialmente como se pensava (Figura 2.3 – Crescimento exponencial do custo das mudanças). O uso efetivo da comunicação, a qualidade assegurada pelos testes e a flexibilidade do processo podem ser usados para "achatar" a curva de mudanças e diminuir seu impacto sobre os custos do projeto (Figura 2.4 – Crescimento gradativo do custo das mudanças).

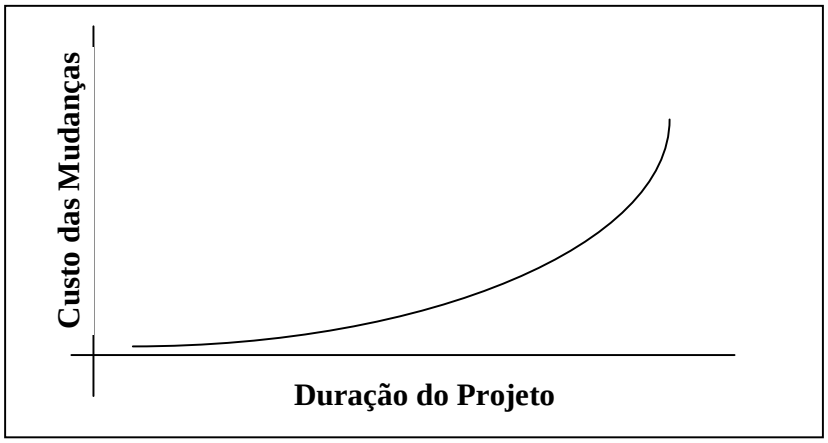

Figura 2.3 – Crescimento exponencial do custo das mudanças

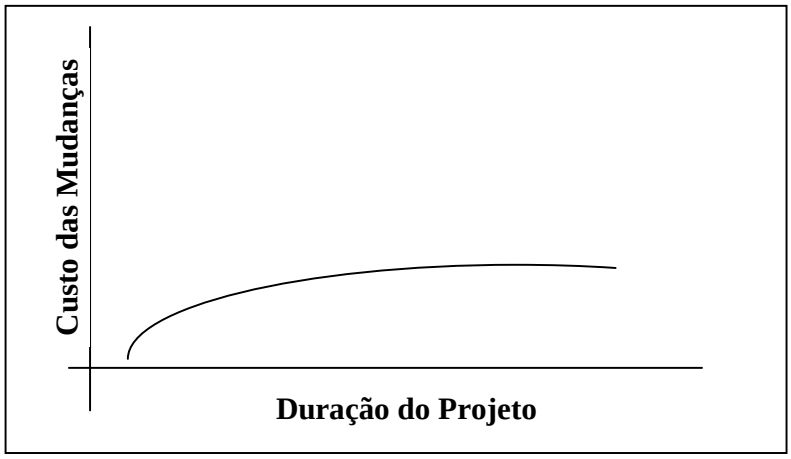

Figura 2.4 – Crescimento gradativo do custo das mudanças

Outro questionamento está relacionado à minimização da documentação do projeto. De acordo com XP, a melhor e mais completa fonte de informações sobre o sistema é o código. Por isso, ele deve ser simplificado e comentado de modo a ser compreendido por toda a equipe.

A justificativa para evitar a produção de documentação adicional é a necessidade de atualizações durante o desenvolvimento, à medida que os requisitos vão sendo alterados. Além do tempo necessário para escrever a documentação inicial, seria preciso mais tempo para torná-la consistente a cada mudança.

Entretanto, segundo Maurer [MAURER, 2002], essa estratégia só funciona bem para times pequenos (5 a 15 pessoas) com comunicação intensa, e fixos, em que não há mudança constante dos membros. Do contrário, o tempo necessário para transmitir conhecimentos sobre o projeto e treinar os novos integrantes passaria a ser crítico para o projeto. Essa é uma opinião compartilhada por Gary Pollice [POLLICE, 2001].

No caso do desenvolvimento Web há ainda outra questão: como as equipes envolvem profissionais multidisciplinares, nem sempre todos os integrantes conseguirão entender o sistema apenas observando seu código. Designers e gerentes de redes, por exemplo, podem não conhecer uma determinada linguagem de programação, ou não conseguir ter uma visão geral do sistema apenas examinando sua coleção de classes e métodos. Nesse caso, uma documentação em mais alto nível do sistema, seja na forma de texto ou diagramas, poderia ser útil para comunicar os objetivos do projeto mais claramente. O uso de ferramentas CASE pode reduzir o esforço necessário para gerar essa documentação, uma vez que elas utilizam o código da aplicação para gerar os diagramas automaticamente.

A questão da necessidade de documentação também está diretamente ligada à manutenção da aplicação. Como sugere Scott Ambler [AMBLER, 2002b], XP assume que o sistema sempre será mantido pela equipe que o desenvolver, ou parte dela, minimizando assim a necessidade de documentação para transmitir as informações. Entretanto, essa pode não ser a realidade de muitas empresas, principalmente as que atuam no mercado global, com terceirização de suas atividades (Outsourcing) e adaptação de software para diferentes mercados (Localização).

De qualquer forma, as práticas de XP vêm sendo bastante recomendadas por diversos autores [NOYES, 2002] [AMBLER, 2002a] [AMBLER, 2002b] [FOWLER, 2001] [HIGHSMITH, 2002], e alguns já discutem a adequação da metodologia ao desenvolvimento Web [MAURER, 2002] [MURRU, 2003], e até mesmo ao desenvolvimento distribuído [KIRCHER, 2001], ignorando a restrição de times fisicamente

próximos. Os usos de soluções híbridas, como RUP e XP [POLLICE, 2001] e XP e Scrum [MAR, 2001] [SLIWA, 2002], também são discutidos.

2.3.1.2 SCRUM

A comparação entre desenvolvedores de software e jogadores de *Rugby* originou o nome dessa metodologia. Scrum é a denominação de cada time de 8 jogadores nesse esporte, cujo objetivo é levar uma bola através do campo. Cada time age em conjunto, como uma unidade integrada, em que cada membro desempenha um papel específico e todos se ajudam em busca de um objetivo comum. Assim também são os times de desenvolvimento que adotam o processo Scrum.

Criado por Jeff Sutherland, Ken Schwaber e Mike Beedle, Scrum baseia-se em 6 características: flexibilidade dos resultados, flexibilidade dos prazos, times pequenos, revisões freqüentes, colaboração e orientação a objetos [SCHWABER, 1995].

- **Flexibilidade dos resultados:** a decisão sobre o que será entregue e quando é tomada pelo cliente, de modo que os interesses do negócio sejam respeitados. As estimativas dos desenvolvedores auxiliam os clientes nessa definição.

- **Flexibilidade dos prazos:** os cronogramas podem ser alterados de modo a antecipar ou adiar a produção das funcionalidades, conforme as necessidades do cliente.

- **Times pequenos:** segundo Schwaber, um time não deve ter mais de 6 pessoas, mas é possível ter mais de um time de desenvolvimento. Já Linda Rising e Norman Janoff [RISING, 2000] propõem que o time possa ser formado por até 10 integrantes.

- **Revisões freqüentes:** o progresso, a complexidade e os riscos do projeto são constantemente acompanhados e medidos. Ao final de cada ciclo de desenvolvimento, uma versão executável do software é produzida para avaliação pelo cliente.

- **Colaboração:** a troca de informações e de resultados ocorre com freqüência, dentro de cada time e entre os times.

- **Orientação a Objetos:** cada time assume a responsabilidade por um conjunto de objetos relacionados.

Uma visão geral da dinâmica de funcionamento do processo Scrum pode ser observada na Figura 2.5 (Visão geral da dinâmica de processo Scrum)[MAR, 2001]. No início do projeto, cliente e desenvolvedores definem o *Backlog*, ou lista de requisitos, para a aplicação. Também são definidas as datas para entrega de resultados, a partir da priorização mais favorável ao cliente, e são estimados os custos do projeto. Uma análise

inicial de riscos é preparada. As ferramentas de trabalho e os integrantes das equipes são escolhidos. Um dos desenvolvedores é eleito gerente do projeto (*Scrum Master*).

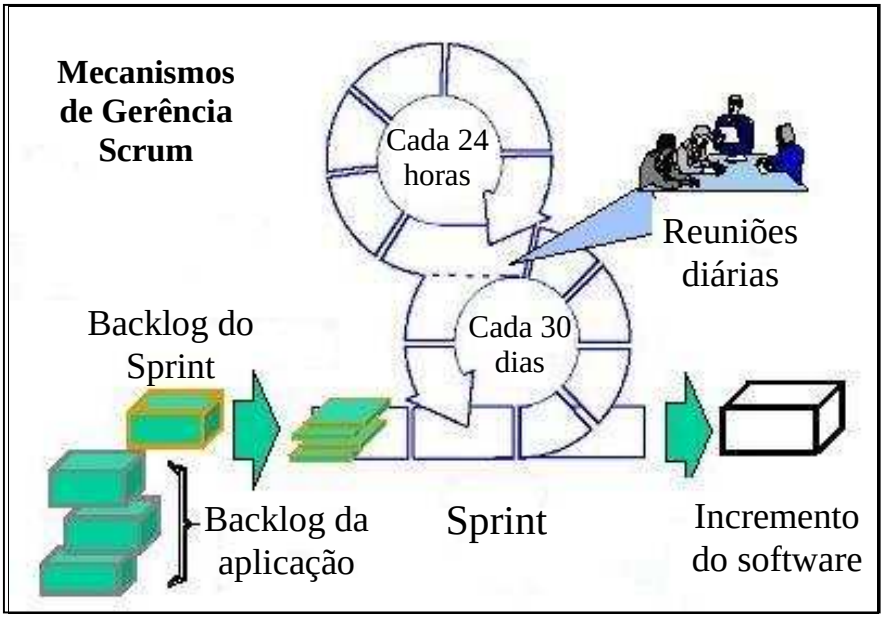

Figura 2.5 – Visão geral da dinâmica de processo Scrum

O próximo passo é analisar a arquitetura da aplicação, através de um projeto em alto nível. À medida que os requisitos do projeto vão sendo alterados, a arquitetura é refinada para refletir as mudanças.

Os ciclos de desenvolvimento duram no máximo 30 dias e são chamados de *Sprints*. No início de cada *Sprint*, as equipes fazem uma lista das atividades que precisam ser realizadas (*Backlog do Sprint*), e as responsabilidades são distribuídas. Os desenvolvedores discutem os padrões que serão adotados, e as atividades de análise, codificação e testes se iniciam.

Durante todo o *Sprint*, são realizadas reuniões diárias de 15 minutos, conduzidas pelo gerente do projeto, com o objetivo de acompanhar o progresso e fortalecer o espírito de equipe e o comprometimento. Três perguntas devem ser respondidas por cada membro sobre suas responsabilidades [RISING, 2000]:

- O que foi feito ontem?
- O que será feito hoje?
- Há algum obstáculo à realização de suas atividades?

Remover os obstáculos apontados na reunião é dever do gerente do projeto, de modo que os desenvolvedores se concentrem apenas nas questões técnicas.

Ao final de cada Sprint, uma versão executável do software é apresentada ao cliente para obter feedback. Os defeitos encontrados são adicionados ao *Backlog* do Projeto. Esse

também é o momento do cliente decidir se deseja que o projeto continue ou não, de acordo com a qualidade e eficiência percebidas.

Ao longo de todo o projeto, são aplicados mecanismos de gerência Scrum, como o acompanhamento de alguns controles. A quantidade de funcionalidades não entregues, a necessidade de mudanças para corrigir defeitos ou para atualização tecnológica, os problemas técnicos encontrados, e os riscos e as estratégias para evitá-los são exemplos de controles observados durante o desenvolvimento.

No final do projeto, são feitos os testes finais de integração, e a documentação do usuário é escrita. Também são preparados materiais para treinamento e para campanhas de marketing, uma vez que o processo acompanha o ciclo completo de desenvolvimento da aplicação [SCHWABER, 1995].

Por serem Processos *Agile*, Scrum e XP apresentam muitas semelhanças. Entretanto, a organização das metodologias é diferente. Enquanto a ênfase de XP está em técnicas de desenvolvimento, a ênfase de Scrum está nos mecanismos de gerência do processo, permitindo que quaisquer técnicas para a produção de software sejam adotadas. Como conseqüência, alguns autores sugerem o uso conjunto das duas abordagens [MAR, 2001][SLIWA, 2002].

2.3.2 METODOLOGIA PARA DESENVOLVIMENTO DE APLICAÇÕES WEB ESCALÁVEIS E MANUTENÍVEIS

Como já foi observado anteriormente, apesar de atenderem os requisitos dos processos de desenvolvimento Web, as metodologias *Agile* são muito genéricas e não descrevem atividades específicas do desenvolvimento de Aplicações Web. Essa nova proposta de metodologia foi apresentada por Athula Ginige e San Murugesan [GINIGE, 2001c], e destaca, com detalhes, alguns passos importantes para construir uma Aplicação Web de sucesso.

O artigo apresenta duas idéias centrais muito importantes. A primeira é que algumas características não-funcionais das Aplicações Web, como escalabilidade e manutenibilidade, devem ser previstas pela equipe de desenvolvimento desde o início do projeto, pois não podem ser acrescentadas depois que o software fica pronto. Essa é uma questão que merece a atenção dos desenvolvedores.

Os autores destacam ainda que uma Aplicação Web é formada basicamente por dois aspectos: o conteúdo das páginas, e o software utilizado para apresentar o conteúdo ao usuário e prover serviços. A separação desses aspectos é muito importante durante o desenvolvimento, pois os resultados são gerados por profissionais com características distintas (times heterogêneos). Cada equipe de profissionais trabalha de maneira diferente, embora seja necessário coordenar o trabalho conjunto de todos eles.

As fases do processo proposto por Ginige e Murugesan [GINIGE, 2001c] podem ser vistas na Figura 2.6 (Processo de desenvolvimento para Sistemas Web). Foram

acrescentadas indicações dos resultados gerados em cada etapa para tornar o gráfico mais significativo.

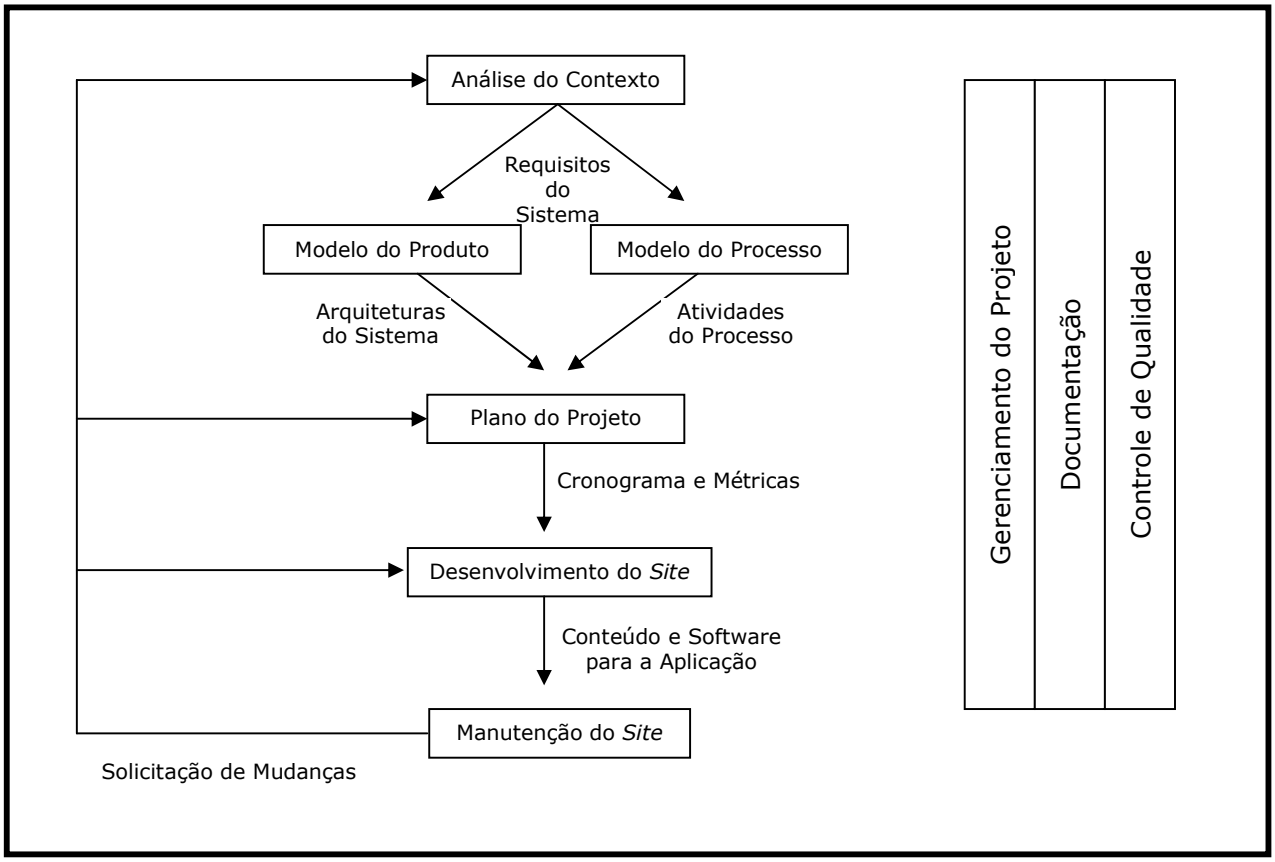

Figura 2.6 – Processo de desenvolvimento para Sistemas Web

O primeiro passo é fazer uma análise sobre o contexto em que será inserida a aplicação (regras do negócio, ambiente operacional, objetivos, etc). Além dos requisitos funcionais, são discutidos nessa etapa os requisitos não-funcionais do software e sua relevância de acordo com o tipo de aplicação a ser desenvolvida.

Por exemplo, numa aplicação para comércio eletrônico, em que conteúdo e funcionalidade evoluem constantemente, a escalabilidade é crítica. Já num sistema de catálogo de produtos on-line, é fundamental que haja facilidade de manutenção para que os produtos disponíveis e seus preços possam ser atualizados constantemente.

Também nessa fase, são identificados aspectos não-técnicos que podem afetar o projeto, como políticas organizacionais, e questões culturais, legais e sociais.

A partir dos requisitos levantados, dois modelos importantes são concebidos: o Modelo do Produto e o Modelo do Processo.

No Modelo do Produto, são descritos os componentes do sistema e seus relacionamentos. O objetivo é fazer um detalhamento sobre o software que será construído. Para isso, são definidos três tipos de arquitetura, como mostra a Tabela 2.2 (Definição de arquiteturas para Aplicações Web).

Arquitetura do Sistema	Arquitetura da Aplicação	Arquitetura do Software
Descrição das interações da rede com os diversos servidores (servidor Web, servidor de aplicação, servidor de Banco de Dados)	Descrição dos módulos do sistema e das funcionalidades disponíveis	Descrição dos módulos de software e de Banco de Dados necessários para implementar as funcionalidades pretendidas

Tabela 2.2 – Definição de arquiteturas para Aplicações Web

Já no Modelo do Processo, o objetivo é identificar as atividades que precisam ser desempenhadas para implementar o sistema (análise de requisitos, projeto, testes, etc) e definir uma seqüência para as ações, que serão realizadas pelas três categorias de desenvolvedores. O Modelo do Processo deverá ser adequado à natureza da aplicação sendo construída.

O próximo passo antes de se iniciar o desenvolvimento propriamente dito é a definição de um Plano para o Projeto, com métricas e cronogramas, que devem ser baseados no escopo da aplicação e no processo adotado.

O desenvolvimento da Aplicação Web se dá em duas etapas:

- Desenvolvimento do conteúdo: os profissionais responsáveis por esse aspecto da aplicação constroem protótipos para as páginas Web, a partir dos requisitos do cliente e do perfil dos usuários do sistema. O *layout* e a navegação são avaliados e segue-se um processo iterativo de implementação de melhorias e novas avaliações. Ao final dessa etapa, os desenvolvedores apresentam modelos de *layout* para as páginas, diretrizes para o desenvolvimento de conteúdo e definições sobre a forma de estruturação das informações (tipo de mídia, mecanismo de acesso, etc).

- Desenvolvimento do software para a aplicação: a partir dos resultados da fase anterior, os profissionais de software dão início à codificação, sempre considerando os requisitos não-funcionais do sistema. Nessa fase, estratégias como o Desenvolvimento Baseado em Componentes e a criação de menus e *links* dinamicamente são bastante recomendáveis.

Finalizada a aplicação, inicia-se a fase de manutenção, que no caso das Aplicações Web se caracteriza como um processo contínuo. São três os tipos de manutenção realizados: manutenção do conteúdo (atualizações), manutenção do software (mudanças tecnológicas e necessidade de novas funcionalidades), e manutenção de hardware e de rede (atualização de versões, correção de falhas).

Dependendo das necessidades observadas na fase de manutenção, o desenvolvimento pode recomeçar a partir de qualquer etapa. O Gerenciamento do Projeto, a Documentação e o Controle de Qualidade são atividades que acompanham todo o ciclo de vida do Desenvolvimento Web.

Apesar da metodologia proposta refletir em muitos aspectos as necessidades das Aplicações Web, não há sugestão concreta de atividades que possam integrar o Modelo do Processo, tampouco de ferramentas e métricas que devam ser adotadas durante o desenvolvimento.

Além dos Processos *Agile* e da metodologia proposta por Ginige e Murugesan [GINIGE, 2001c] já descritos, o Processo Unificado [KRUCHTEN, 2000], bastante conhecido da comunidade de software, também pode ser adaptado para projetos de desenvolvimento de Aplicações Web.

Para isso, seria necessário substituir algumas de suas atividades genéricas por outras mais apropriadas às necessidades dos desenvolvedores Web, e diminuir a ênfase na produção de artefatos para acelerar o ritmo de desenvolvimento. As limitações em relação ao uso de ferramentas e métricas adequadas são as mesmas dos outros processos vistos.

2.4 Considerações Finais

Neste capítulo foram descritas as principais características que diferenciam as Aplicações Web das aplicações tradicionais, como a importância dos requisitos não-funcionais (confiabilidade, usabilidade, segurança, acessibilidade, escalabilidade, manutenibilidade) e a pressão do *time-to-market*. Também foram apresentados os elementos que compõem as arquiteturas das Aplicações Web.

Em seguida, foi justificada a necessidade de uma abordagem de Engenharia de Software diferente das tradicionais para a produção de Aplicações Web. A nova abordagem deve considerar a realização de ciclos de desenvolvimento menores, por equipes pequenas e multidisciplinares, num cenário com mudança constante de requisitos e necessidade de manutenção constante do software.

Finalmente, foram apresentadas algumas propostas de metodologias, como os Processos *Agile* (*eXtreme Programming* e *Scrum*) e o processo sugerido por Ginige e Murugesan [GINIGE, 2001c], que podem ser adaptadas para o Desenvolvimento de Aplicações Web, embora não atendam plenamente suas necessidades.

Outro aspecto interessante sobre o Desenvolvimento de Aplicações Web não citado neste capítulo é que, em muitos casos, ele vem sendo realizado de maneira distribuída. Uma vez que profissionais com diferentes competências são necessários, e sendo pouco provável que eles possam estar localizados todos numa mesma região, é natural que as empresas resolvam usar a Internet para obter a colaboração remota de especialistas dispersos geograficamente.

Também é bastante comum que as Aplicações Web sejam desenvolvidas simultaneamente para diferentes mercados, uma vez que usuários de todos os países acessam a Internet. Adaptando suas aplicações para as características de cada cultura, as empresas de software podem maximizar seus lucros e ganhar destaque no cenário global.

No próximo capítulo, serão analisados os principais desafios para a realização de projetos de desenvolvimento de software com equipes distribuídas, e também as implicações da produção de software para usuários de diferentes países. Serão avaliados ainda os impactos desse novo modelo de desenvolvimento no ciclo de vida das aplicações.

Capítulo 3

3. Desenvolvimento Global de Software

Com a expansão da Internet e das tecnologias de comunicação como um todo, as empresas de software passaram a considerar duas importantes oportunidades de negócios: o desenvolvimento distribuído e a globalização do mercado.

O desenvolvimento distribuído surgiu como uma alternativa para solucionar o problema de mão-de-obra especializada das empresas. Como afirma Karolak [KAROLAK, 1998], a demanda por aplicações cresce exponencialmente e a oferta de profissionais não tem sido suficiente para dar conta de tantos projetos. Também é comum que as diferentes competências necessárias para a realização de um projeto não estejam presentes numa mesma região.

Usando a Internet, as empresas perceberam que poderiam ter desenvolvedores de diferentes países trabalhando num mesmo projeto, sem que eles precisassem se deslocar para os centros de desenvolvimento. O diálogo e a troca de dados seriam possíveis através de tecnologias de comunicação como e-mail e vídeo-conferência.

A interação remota entre os times também possibilitaria a formação de parcerias com outras empresas no desenvolvimento de software para mercados maiores. A união dos esforços poderia reduzir o *time-to-market* e melhorar a qualidade das aplicações produzidas.

Por outro lado, o avanço dos meios de comunicação contribuiu para uma significativa redução da distância entre os povos. As fronteiras geográficas deixaram de ser um obstáculo para os negócios e os mercados locais se transformaram em mercados globais.

As empresas de software viram a oportunidade de vender seus produtos para usuários de todo o mundo através da Internet. Passaram a estar presentes virtualmente em todos os países, atingindo mercados antes inexplorados, com custos mínimos.

Unindo as duas tendências, o Desenvolvimento Global de Software é caracterizado pela colaboração de times dispersos geograficamente no desenvolvimento de aplicações para mercados globais. A idéia é que um mesmo projeto seja desenvolvido de maneira concorrente por organizações parceiras com o objetivo de atender mercados com características próprias.

As vantagens para a adoção dessa prática são muitas, como aponta Herbsleb [HERBSLEB, 2001]:

- **Oportunidade de explorar recursos escassos em qualquer lugar onde estejam disponíveis, e com custos competitivos**: uma vez que não existe nos grandes centros uma quantidade suficiente de desenvolvedores de software para atender à demanda atual, muitas empresas estão usando o Desenvolvimento Global de Software como alternativa para explorar a mão-de-obra disponível em outros países. A contratação de programadores em países como Índia, China e Filipinas implica também numa redução de custos, visto que os salários pagos nessas regiões são bem menores que os dos grandes centros.

- **Vantagens da proximidade entre os desenvolvedores e o mercado-alvo**: para que um software seja adequado a um mercado específico, é muito importante que os desenvolvedores possam interagir diretamente com os clientes potenciais. Com o desenvolvimento distribuído por vários países, é possível analisar melhor as adaptações que o produto terá que sofrer para cada mercado, pois as necessidades do cliente e o contexto cultural onde ele está inserido são conhecidos.

- **Rápida formação de corporações virtuais para explorar diferentes mercados e necessidade de aproveitar oportunidades de fusões e aquisições de empresas, onde quer que elas apareçam**: a grande competitividade do mercado está levando as empresas a buscarem parcerias para se manterem fortes e atuantes em todo o mundo. As fusões e aquisições acontecem entre empresas que nem sempre estão numa mesma região geográfica, e o Desenvolvimento Global de Software pode ser a alternativa para permitir que os esforços individuais de cada parceiro sejam melhor aproveitados pela corporação como um todo.

- **Pressão para diminuir o *time-to-market* tirando vantagem dos diferentes fusos-horários entre países**: com o Desenvolvimento Global de Software, quanto maior a diferença entre os fusos-horários dos países onde estão espalhados os desenvolvedores, maior a possibilidade de reduzir o tempo de produção do software. Numa situação extrema, com países com fuso de 12 horas, o desenvolvimento pode ocorrer 24 horas por dia e 7 dias por semana (desenvolvimento *follow-the-sun*), com a jornada de uma equipe começando simultaneamente com o fim da jornada de outra, embora isso possa implicar em problemas de comunicação entre as equipes.

Alguns dados indicam que, atualmente, existem mais de 50 nações envolvidas em projetos de desenvolvimento colaborativo de software internacionalmente, o que demonstra uma forte tendência de mercado [CARMEL, 2001].

Muitas multinacionais, como Motorola [BATTIN, 2001], Alcatel [EBERT, 2001], Fujitsu [AOYAMA, 1998] [GAO, 1999], e Bell Labs [MOCKUS, 2001], relatam suas experiências com o Desenvolvimento Global de Software em suas diversas filiais espalhadas pelo mundo.

Mas, antes de adotarem esse modelo de desenvolvimento, as empresas precisam considerar suas particularidades. As parcerias precisam ser bem definidas, os riscos do projeto são maiores, a gerência do desenvolvimento se torna mais complexa, e a diversidade cultural pode dificultar a comunicação entre os times e a aceitação dos produtos em diferentes mercados.

A seguir, serão avaliadas as mudanças organizacionais decorrentes do Desenvolvimento Global de Software, com a participação de diferentes empresas num mesmo projeto, e o trabalho colaborativo de times virtuais dispersos geograficamente. Serão apontados os principais desafios e riscos para a realização desses projetos.

Também serão descritos os impactos dessa nova abordagem de desenvolvimento sobre o ciclo de vida das aplicações produzidas. Considerando Aspectos de Negócios, Aspectos de Desenvolvimento e Aspectos Gerenciais dos projetos, serão apontadas as principais adaptações necessárias nas atividades para atender às particularidades dos times dispersos, e da produção de software para diferentes mercados.

3.1 Aspectos Organizacionais

As primeiras implicações do Desenvolvimento Global de Software podem ser percebidas na transformação das empresas e dos times de desenvolvimento.

A realização de projetos com a participação de várias empresas exige acertos de negócios bem definidos, para que não haja dúvidas sobre prazos e responsabilidades. Na formação das Organizações Virtuais, as empresas podem optar por diferentes tipos de parceria, de acordo com as necessidades do projeto.

Já os Times Virtuais, formados por desenvolvedores dispersos geograficamente, apresentam características que os diferenciam dos times tradicionais, como a troca de informações em formato eletrônico e o compartilhamento de código. Seus principais problemas estão relacionados à comunicação e às diferenças culturais.

3.1.1 ORGANIZAÇÕES VIRTUAIS

As Organizações Virtuais foram criadas com o intuito de distribuir e terceirizar o trabalho antes feito dentro de uma única empresa. Uma Organização Virtual de software é caracterizada por uma rede de empresas diferentes e independentes, trabalhando juntas como se fossem uma única organização no desenvolvimento de projetos de software [CARMEL, 1999]. No caso do Desenvolvimento Global de Software, as partes integrantes de uma Organização Virtual costumam estar dispersas geograficamente, como mostra a Figura 3.1 (Estrutura das Organizações Virtuais).

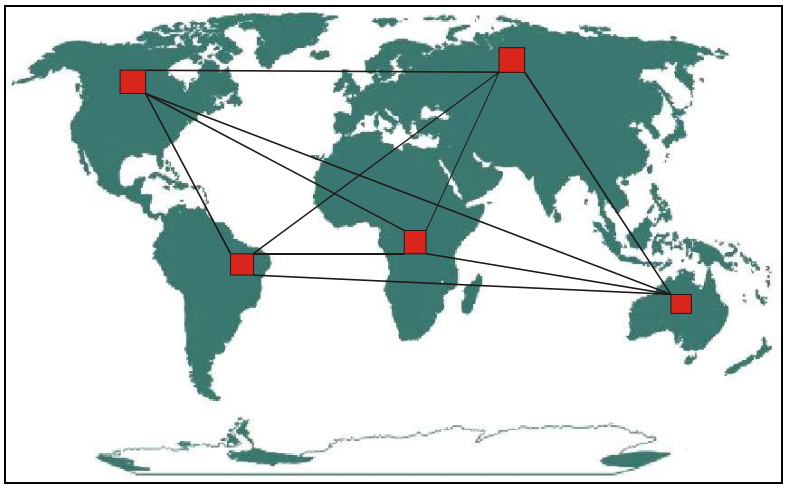

Figura 3.1 – Estrutura das Organizações Virtuais

As responsabilidades das empresas numa Organização Virtual podem variar de acordo com o tipo de parceria firmada por elas. Karolak [KAROLAK, 1998] caracteriza duas formas de parceria entre empresas de software:

- **Parceria Estratégica**: ocorre quando uma empresa desenvolve um produto de software e busca empresas parceiras para atuar no mercado global. Algumas etapas do ciclo de vida do software, como manutenção, suporte, treinamento e comercialização passam a ser de responsabilidade dessas empresas parceiras, que podem também fazer ajustes ao software para adequá-lo ao perfil de seu público-alvo. Esse tipo de parceria pode ser de grande valia para empresas que desejem penetrar em mercados fechados ou desconhecidos, já que as empresas da região podem fornecer uma visão melhor do perfil dos clientes.

- **Joint Venture**: ocorre quando a união das empresas parceiras origina uma nova empresa, com características próprias, mas que mantém responsabilidades para com todas as partes envolvidas. Os projetos são desenvolvidos em conjunto, com investimento de recursos financeiros, tecnológicos e humanos distribuídos de acordo com a competência e a participação de cada empresa no negócio. Essa prática é indicada para corporações que desejem diversificar seu ramo de atuação sem perder o foco inicial.

O ***Outsourcing*** é outro tipo de parceria comum em Organizações Virtuais, e ocorre quando uma organização contrata outras empresas para desempenharem etapas do desenvolvimento de software, especialmente a codificação dos produtos. Nesse caso, as empresas contratadas não atuam como representantes da empresa principal, como ocorre nas Parcerias Estratégicas, mas apenas participam como prestadoras de serviços. No Desenvolvimento Global de Software, as empresas contratadas costumam estar localizadas

em países com muita oferta de mão-de-obra qualificada, como é o caso da Índia, grande líder nesse segmento [HEEKS, 2001].

Contratos de *Outsourcing* também podem ser feitos para a manutenção de software e a adaptação de produtos para diferentes mercados (Localização). As empresas que investem em contratos de software, de um modo geral, conseguem ver os custos de seus projetos reduzidos de 9 a 15% [THOMSETT, 1998].

Em todos os casos de parcerias em Organizações Virtuais, é importante formalizar as responsabilidades das empresas envolvidas. Afinal, assuntos como propriedade intelectual das soluções desenvolvidas, cumprimento de cronogramas e orçamentos, atendimento a padrões de qualidade, e questões relacionadas ao acompanhamento dos projetos precisam ser bem discutidas para evitar problemas futuros. É recomendável estabelecer um *Statement Of Work (*SOW), um tipo de acordo de trabalho reconhecido legalmente, que descreva de forma não ambígua como esses e outros aspectos serão tratados pela Organização Virtual [KAROLAK, 1998].

3.1.2 TIMES VIRTUAIS

A formação das equipes nos projetos desenvolvidos nas Organizações Virtuais também não é convencional. Pessoas com diferentes culturas, e que nem sempre falam a mesma língua, precisam trocar idéias e experiências como se estivessem todas numa mesma sala.

Os chamados Times Virtuais possuem características bem diferentes em relação às equipes de trabalho tradicionais, como pode ser observado na Tabela 3.1 (Comparação entre times tradicionais e times virtuais) [CARMEL, 1999].

A gestão de Times Virtuais pode ser vista como um fator crucial para o sucesso do Desenvolvimento Global de Software, uma vez que todo o processo é concebido sobre essa base. Técnicas diferenciadas de gerenciamento precisam ser adotadas para atender essas características particulares.

Times Tradicionais	Times Virtuais
Membros geograficamente juntos	Membros geograficamente dispersos
Interação face-a-face	Comunicação eletrônica
Integrantes ligados a uma mesma organização	Integrantes ligados a diferentes organizações
Estrutura hierárquica	Estrutura em rede (descentralizada)
Comunicação informal predominante	Comunicação contínua e estruturada predominante
Autoridade baseada na posição ocupada na organização	Autoridade baseada no conhecimento dos processos e/ou tecnologias
Informação distribuída de cima para baixo	Informação acessível e compartilhada em todos os níveis
Informação em papel	Informação em formato eletrônico
Compartilhamento de trabalhos acabados	Compartilhamento contínuo de trabalhos inacabados
Concentração de conhecimentos	Compartilhamento de conhecimentos
Processos transparentes (pouco documentados)	Processos conhecidos apenas através da documentação visível pelo computador
Cultura da organização e do projeto aprendida por osmose	Cultura da organização e do projeto aprendida através de comunicações eletrônicas e artefatos

Tabela 3.1 – Comparação entre times tradicionais e times virtuais

Alguns problemas típicos dos Times Virtuais são apontados por Carmel [CARMEL, 1999], e classificados como forças centrífugas, que tendem a separar a equipe e prejudicar os projetos virtuais. São elas:

- **Dispersão Geográfica**: gerenciar Times Virtuais envolve uma complexidade muito maior do que a observada em projetos tradicionais. O acompanhamento das atividades precisa ser mais contínuo e intenso, os canais de comunicação tornam o contato entre a equipe menos freqüente e passível de desentendimentos, e os artefatos e ferramentas de desenvolvimento precisam ser disponibilizados em várias localidades simultaneamente. Esses problemas podem resultar em custos extras com tecnologias colaborativas e de comunicação, além de atrasos no cronograma.

- **Falência dos Mecanismos de Supervisão e Coordenação Tradicionais**: as práticas para acompanhamento dos projetos, que costumam ser baseadas na comunicação, passam a ser inadequadas, pois as formas de interação entre as equipes não são as mesmas. A Coordenação (ato de integrar as entidades e atividades da empresa para que contribuam para um objetivo comum) deixa de ser feita através de visitas às salas dos desenvolvedores, e as dúvidas

passam a depender de fusos-horários para serem esclarecidas. A Supervisão (processo de adesão a metas, práticas, padrões e níveis de qualidade) só se torna possível quando todos os membros entendem as diretrizes a serem seguidas, cada um em seu idioma e no contexto cultural em que vivem, sem interpretações dúbias.

- **Perda Semântica na Comunicação**: os Times Virtuais costumam se comunicar através de telefonemas ou pela troca de mensagens eletrônicas, tecnologias que possibilitam a interação à distância. Mesmo sendo eficazes para a propagação de informações, nenhuma tecnologia parece ser capaz de transmitir toda a carga semântica contida na linguagem corporal perceptível no contato face-a-face (gestos, postura, expressões faciais). Como conseqüência, muitas atitudes e opiniões podem ser mal interpretadas, gerando conflitos entre os membros do time.

- **Perda do Espírito de Equipe**: gerenciar um Projeto Virtual implica em fazer com que pessoas que não se conhecem, e que estão separadas por fronteiras geográficas e culturais, confiem umas nas outras e se unam para trabalhar por um único objetivo. Certamente esta não é uma tarefa simples, e além dos problemas de relacionamento que podem surgir, algumas pessoas podem se sentir desmotivadas quando não conseguem perceber a importância de seu trabalho para o projeto como um todo.

- **Diferenças Culturais**: pessoas de diferentes países possuem características próprias que vão muito além de um idioma e uma posição geográfica. Os valores pessoais e organizacionais costumam ser conflitantes, principalmente em se tratando de culturas ocidentais em relação a culturas orientais. A visão de hierarquia na empresa, os riscos aceitáveis nos negócios, a pontualidade e a objetividade nas reuniões, a formalidade esperada na comunicação: todos esses aspectos precisam ser compreendidos, e os ajustes necessários devem ser feitos para que o projeto dê certo.

Além dos problemas, Carmel [CARMEL, 1999] cita também os fatores que podem contribuir para que os Times Virtuais permaneçam coesos e integrados, as chamadas forças centrípetas:

- Uso da infra-estrutura de comunicação disponível para intensificar o contato entre a equipe.
- Uso de tecnologias colaborativas[1] para o compartilhamento de artefatos e ferramentas de projeto.
- Uso de técnicas especiais de gerenciamento.

[1] Tecnologias Colaborativas são aquelas usadas para dar suporte à comunicação remota dos desenvolvedores, como e-mail, fax, áudio e vídeo-conferência, etc, ou para auxiliar na adoção de práticas da Engenharia de Software, como programas para gerência de projeto e controle de configuração de software [CARMEL, 1999].

- Estabelecimento de uma metodologia de desenvolvimento para funcionar como linguagem comum para o time.
- Definição de uma arquitetura do produto, fator determinante para a alocação de tarefas.
- Construção do time através de uma comunicação efetiva e baseada na criação de relacionamentos de confiança entre os membros.

3.2 Impactos no Ciclo de Vida das Aplicações

A opção pelo Desenvolvimento Global de Software pressupõe alterações no ciclo de vida das aplicações, considerando em suas atividades alguns aspectos importantes, como a ampliação do mercado-alvo, a dispersão geográfica e organizacional dos times, e o uso da Web e de tecnologias colaborativas durante o desenvolvimento.

Embora grande parte das empresas veja a ampliação do mercado-alvo como uma oportunidade de realizar mais negócios e aumentar os lucros, algumas organizações ignoram a complexidade envolvida na produção de software para o mercado global. Juntamente com o idioma, as necessidades e preferências dos usuários variam de país para país, e agradar um público tão heterogêneo torna-se um desafio. Pesquisas de mercado e adaptações no produto final são essenciais.

A dispersão geográfica e organizacional dos times torna ainda mais difícil a divisão de tarefas e a alocação de recursos, como ferramentas de desenvolvimento e bibliotecas de código. Além disso, são necessários esforços adicionais para a integração das soluções produzidas pelos times.

A comunicação entre os participantes e o acompanhamento do projeto dependem diretamente do uso da Web e de tecnologias colaborativas, como e-mail e vídeo-conferência. A Internet também é responsável pelo surgimento de novos modelos de compra e distribuição de software, além de criar novos canais de comunicação entre as empresas e seus clientes globais.

A seguir, serão analisadas as principais implicações do Desenvolvimento Global de Software sobre os Aspectos de Negócio, os Aspectos de Desenvolvimento, e os Aspectos Gerenciais, como mostra a Figura 3.2 (Implicações do desenvolvimento global de software).

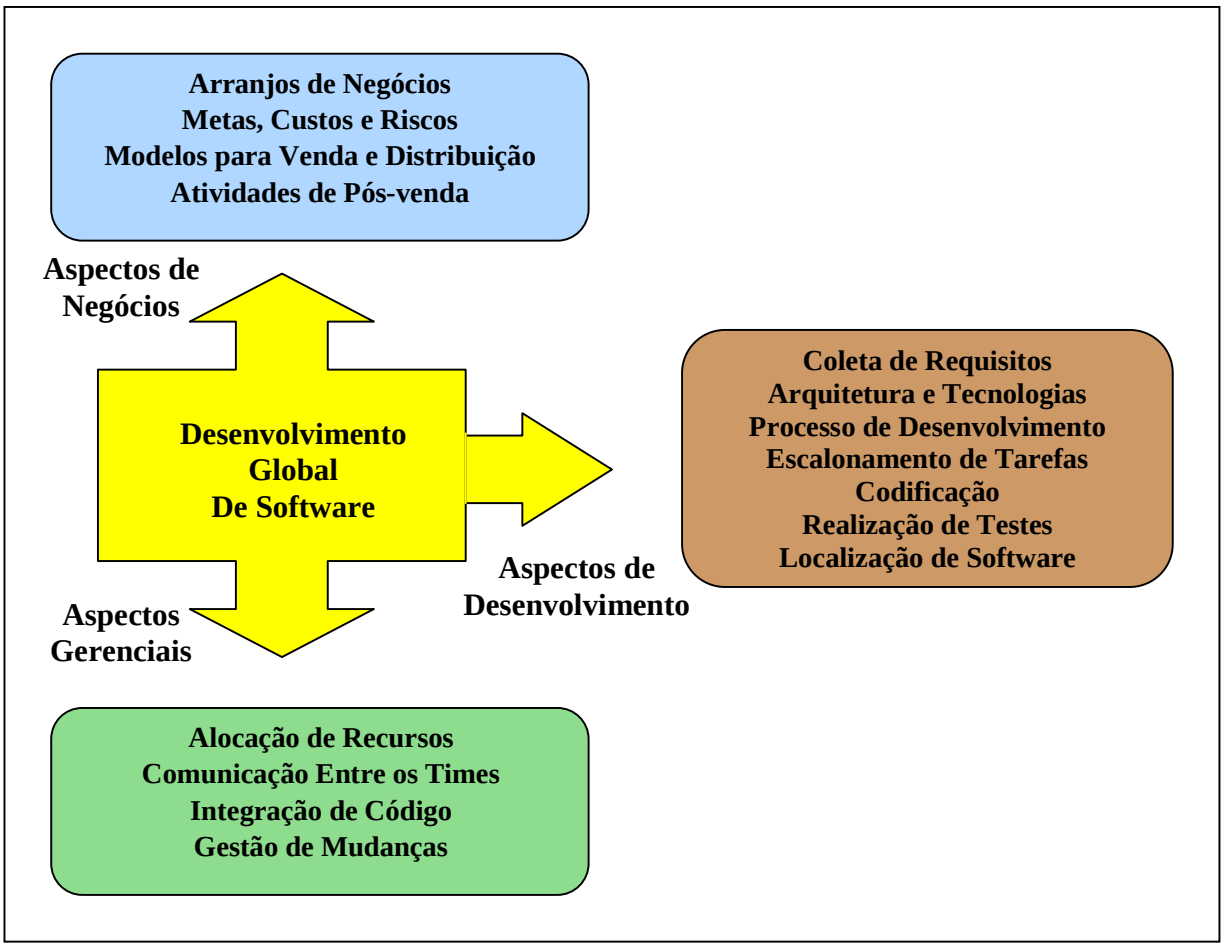

Figura 3.2 – Implicações do Desenvolvimento Global de Software

3.2.1 ASPECTOS DE NEGÓCIOS

Em um projeto de Desenvolvimento Global de Software, as atividades de negócio estão relacionadas principalmente à formação das Organizações Virtuais, e à comercialização do software para diferentes mercados.

As principais fases do ciclo de vida de desenvolvimento afetadas são o planejamento inicial e as atividades de venda e pós-venda.

3.2.1.1 ESTABELECIMENTO DOS ARRANJOS DE NEGÓCIOS

O primeiro passo num projeto de Desenvolvimento Global de Software é a formação das Organizações Virtuais, que ocorre quando é percebida uma oportunidade de negócio. Se uma empresa não tem a competência exigida para o projeto, ou os recursos necessários para realizá-lo, ela vai, então, em busca de empresas parceiras.

É aconselhável que a participação no negócio e as atribuições de cada uma das empresas envolvidas sejam definidas formalmente logo no início do projeto para evitar desentendimentos futuros. Se a parceria envolver empresas diferentes, o contrato, também conhecido como *Statement of Work* (SOW), deve incluir questões como propriedade intelectual das soluções desenvolvidas, padrões de qualidade, e comprometimento em relação a cronogramas e custos. Como cada empresa possui normas e valores próprios, é importante estabelecer regras comuns para o projeto, que devem ser respeitadas por todos os times.

3.2.1.2 DEFINIÇÃO DE METAS, CUSTOS E RISCOS PARA O PROJETO

Logo na fase de planejamento, as metas para o projeto são traçadas, e critérios de aceitação, como padrões de qualidade e retorno esperado do investimento, são determinados pela Organização Virtual.

As experiências de alguns projetos virtuais [EBERT, 2001] [FAVELA, 2001] destacam a importância de garantir que todos os participantes tenham uma mesma visão do projeto, com metas claras e bem definidas. Além de contribuir para o comprometimento dos participantes, evidenciando o papel de cada um no projeto, as metas ajudam a estabelecer o espírito de time, levando pessoas que na maioria das vezes não se conhecem a se unirem por objetivos comuns.

A análise de riscos tem início no planejamento e segue por todo o desenvolvimento. Projetos virtuais são essencialmente mais arriscados que projetos convencionais, pois a distância entre os desenvolvedores impõe dificuldades não só operacionais, relativas à colaboração dos times, mas principalmente gerenciais, que envolvem a integração do trabalho das equipes e o acompanhamento do projeto. O gerenciamento desses riscos é fundamental para antecipar problemas e evitar decisões equivocadas durante o desenvolvimento.

Uma estimativa de custos para o projeto também é importante logo no planejamento. Despesas com viagens, ferramentas de desenvolvimento e infra-estrutura de comunicação (Internet e tecnologias colaborativas) devem ser previstas.

3.2.1.3 ESCOLHA DE MODELOS PARA VENDA E DISTRIBUIÇÃO DE SOFTWARE

A popularização da Internet proporcionou o surgimento de novos modelos de distribuição e venda de software. A Distribuição Eletrônica de Software [MURUGESAN, 1999] é uma ótima opção para reduzir os custos das empresas e atingir o mercado global mais facilmente.

A idéia de distribuir pacotes de software pela Internet já vem sendo usada há algum tempo por empresas que precisam fornecer atualizações de seus produtos a usuários de todo o mundo, como é o caso das ferramentas antivírus. A solução parece ideal para as Organizações Virtuais, pois além de eliminar os custos com embalagens, transporte e armazenamento, o acesso aos produtos através de páginas Web é mais fácil e rápido, e não requer a logística que a entrega de software para diferentes países envolveria.

Modelos de venda de software mais flexíveis também começam a ganhar espaço. A idéia é que os usuários possam pagar apenas pelo que utilizam (*pay-per-function*), e só quando utilizam (*pay-per-use*). Esse conceito de venda customizada de produtos já pode ser observado na comercialização de assinaturas e licenças temporárias para acesso a algumas aplicações e páginas Web.

3.2.1.4 REALIZAÇÃO DE ATIVIDADES DE PÓS-VENDA DO SOFTWARE

Até mesmo as atividades de pós-venda do software vêm sofrendo influência da globalização evidenciada pela Internet. Sem a possibilidade de realizar treinamentos e dar suporte aos usuários remotamente, dificilmente as Organizações Virtuais conseguiriam atender a demanda por esses serviços em diferentes regiões do mundo.

Felizmente, o uso de vídeo-conferência e de ferramentas de transmissão de dados em tempo real ajuda os times virtuais nessas atividades.

A manutenção de software também pode ser feita remotamente, usando a Internet para receber solicitações de mudanças dos clientes e enviar as correções necessárias.

3.2.2 ASPECTOS DE DESENVOLVIMENTO

O impacto do Desenvolvimento Global de Software nas atividades de desenvolvimento do software pode ser percebido desde sua concepção até a fase de transição do produto para o mercado.

A heterogeneidade do público-alvo, espalhado por vários mercados, torna mais difícil a coleta de requisitos, e a realização de testes. Além disso, a arquitetura do software deve ser projetada para sofrer adaptações às características de cada mercado com esforços mínimos, para não aumentar o tempo de desenvolvimento e os custos do projeto. Essas adaptações podem ser funcionais, ou até mesmo culturais, caracterizando o processo de Localização.

Já a dispersão geográfica dos times impõe mudanças em decisões importantes, como a escolha do processo de desenvolvimento e a estratégia para o escalonamento de tarefas. Em ambos os casos, deve-se buscar a produtividade máxima dos times, seja evitando a adoção de metodologias novas que requerem adaptação, seja tornando seu trabalho mais independente.

A codificação do software também é afetada, e o auxílio de tecnologias colaborativas é fundamental. Até mesmo os tipos de testes realizados na aplicação sofrem mudanças, e os times precisam se adequar às novas exigências.

3.2.2.1 COLETA DE REQUISITOS

Quando todos os acertos para o projeto já foram feitos no planejamento inicial, o próximo passo é identificar o público-alvo do produto. Se o objetivo da Organização

Virtual for o lançamento simultâneo do software em vários mercados, a coleta de requisitos deverá envolver clientes potenciais de cada região.

Essa é uma tarefa complexa, pois consultar indivíduos de diferentes regiões geográficas para conhecer suas preferências e necessidades pode demandar tempo e dinheiro. Entretanto, é um procedimento indispensável, pois a satisfação dos usuários vem se impondo como uma condição para o sucesso das aplicações no mercado [CORONADO, 2001].

A Internet é a grande aliada dos times virtuais para realizar a coleta distribuída de requisitos, pois exige menos esforços e consegue atingir um número muito maior de pessoas simultaneamente. São utilizados e-mails e formulários Web para traçar o perfil dos usuários.

A definição do escopo do software ocorre a partir da análise dos requisitos iniciais.

3.2.2.2 DEFINIÇÃO DA ARQUITETURA E DAS TECNOLOGIAS

Além dos requisitos funcionais apontados pelos usuários, existem importantes requisitos não-funcionais que devem ser considerados na escolha da arquitetura das aplicações destinadas ao mercado global. A portabilidade, a adaptabilidade e o reuso são alguns exemplos.

- **Portabilidade**: como os usuários globais utilizam diferentes tipos de processador e sistema operacional, o ideal é que o código do software possa ser executado independentemente da plataforma destino, ou do tipo de *browser*, no caso de Aplicações Web. Esse é um requisito importante para que o software esteja acessível aos diferentes mercados, e pode ser atendido com a produção de diferentes versões do software, uma para cada contexto, ou com o uso de linguagens e tecnologias independentes de plataforma, como Java e CORBA, respectivamente.

- **Adaptabilidade**: na busca da satisfação dos usuários globais, são necessárias adaptações nas aplicações para adequá-las às necessidades e preferências do público-alvo. A modificação de características do software, e a inclusão de novas funcionalidades de acordo com a demanda do mercado são constantes em projetos deste tipo. Por isso, a arquitetura do software deve tentar minimizar o impacto dessas alterações no tempo e no orçamento do projeto.

- **Reuso**: as adaptações necessárias no software não estão relacionadas apenas a suas funcionalidades, mas também à sua apresentação aos usuários. O idioma e alguns aspectos da interface, como o uso de cores e ícones, variam de acordo com a cultura do mercado-alvo. Neste caso, além de ser adaptável, a arquitetura deve possibilitar o reuso de código, para que toda a lógica da aplicação seja reaproveitada, e cada versão do software não seja construída do zero.

Uma boa estratégia para facilitar a adaptabilidade e o reuso do software é a modularização. A concepção do software como módulos coesos – que encapsulam funcionalidades – e fracamente acoplados – que apresentam um certo grau de independência em suas interações – ajuda a diminuir a complexidade do desenvolvimento, além de propiciar o trabalho distribuído dos times [CARMEL, 1999] [BENNATAN, 2002].

Outra prática indicada para promover o reuso é isolar a lógica da aplicação das estruturas de apresentação dos dados. Assim, um mesmo código poderá ser usado com diferentes interfaces, e poderá ser até reaproveitado para aplicações futuras na mesma área de domínio.

A escolha das tecnologias utilizadas no projeto deve ser baseada no atendimento a esses requisitos, e também na familiaridade das equipes com as ferramentas, para que o projeto não sofra atrasos devido a períodos de aprendizagem.

3.2.2.3 ESCOLHA DO PROCESSO DE DESENVOLVIMENTO

Uma Organização Virtual pode envolver diversas empresas diferentes, cada uma com seu próprio processo de desenvolvimento e sua metodologia particular. Essa heterogeneidade dificulta a comunicação entre os times durante o desenvolvimento, e pode causar desentendimentos, pois uma mesma nomenclatura pode referenciar artefatos distintos, e a definição das fases também pode não ser a mesma.

Existem duas possíveis soluções para esse problema: impor um processo único a ser seguido por todos os times, ou permitir que cada time utilize seu próprio processo e buscar meios de facilitar a integração.

Os times da Alcatel [EBERT, 2001] escolheram a primeira opção. Um rigoroso processo de CMM nível 3 passou a ser seguido pelas equipes, que se encontravam dispersas por mais de 15 centros.

Já os times da Motorola [BATTIN, 2001] evitaram o uso de um processo único. Além da dificuldade de impor um processo comum para participantes com culturas e idiomas diferentes, outra razão para respeitar a diversidade foi a diminuição no tempo de desenvolvimento. Como cada time já estava familiarizado com sua metodologia, não foi preciso considerar uma curva de aprendizagem, e os resultados puderam ser produzidos desde o começo. Para facilitar a integração dos processos, foi definido um vocabulário comum, incluindo a nomenclatura dos artefatos, permitindo que cada time fizesse o mapeamento de sua terminologia para a convenção padrão.

Os dois casos foram aplicados em Organizações Virtuais especiais, onde as parceiras são as unidades de uma mesma empresa. No caso de Organizações Virtuais formadas por empresas diferentes, adotar um processo comum seria um desafio ainda maior, pois a cultura organizacional não é a mesma, e é possível que nem todas as parceiras utilizem um processo bem definido. A opção da Motorola parece ser a mais indicada.

Independentemente da solução escolhida, é fundamental garantir que todos os aspectos do projeto sejam devidamente documentados, de forma clara e coerente, numa linguagem acessível a todos os times. Como aponta Karolak [KAROLAK, 1998], a documentação é a cola que mantém unidos os elementos do projeto. É a linguagem comum entre os times, onde são representados os acordos e as convenções a serem seguidas por eles. Por isso, é fundamental que todas as fases do projeto sejam bem documentadas.

3.2.2.4 ESCALONAMENTO DE TAREFAS

A alocação das atividades referentes à concepção de um software entre seus desenvolvedores é uma prática comum em qualquer projeto. Entretanto, no Desenvolvimento Global de Software, esse procedimento exige um planejamento maior, pois os times estão dispersos.

Vários autores [CARMEL, 1999][EBERT, 2001][BATTIN, 2001][MOCKUS, 2001][REPENNING, 2001] descrevem os critérios usados para a distribuição de responsabilidades entre os participantes de projetos virtuais segundo suas experiências. Bennatan [BENNATAN, 2002] apresenta uma compilação dessas propostas, e indica cinco possíveis estratégias: divisão por funcionalidades, divisão por subsistema, divisão por fase de desenvolvimento, divisão por especialidade e divisão por versão do produto.

- **Divisão por funcionalidades ou características do software**: cada time é encarregado de desenvolver um conjunto isolado de funções do software. A vantagem dessa abordagem é que a definição de módulos fracamente acoplados permite uma maior independência entre os times, diminuindo a necessidade de colaboração entre eles. Como os times são definidos pela proximidade de seus participantes, as despesas com comunicação e viagens também são reduzidas. Uma desvantagem é que funcionalidades mais complexas exigem diferentes competências, nem sempre reunidas num mesmo time. Outro problema que pode ocorrer é que o trabalho independente dos times gere resultados incompatíveis, que dificultarão a integração das funcionalidades para formar o software final. Entretanto, uma boa definição da arquitetura pode evitar isso. Essa é a estratégia comumente adotada em projetos de Desenvolvimento Global de Software.

- **Divisão por subsistemas**: a arquitetura do software é dividida em subsistemas e cada time assume a responsabilidade por um deles. Essa estratégia é semelhante à divisão por funcionalidades, mas as características do software são consideradas num nível mais alto, na forma de grandes componentes, sem especificar sua divisão interna. As vantagens e desvantagens dessa estratégia são as mesmas da divisão por funcionalidades.

- **Divisão por fase de desenvolvimento**: cada time passa a ser responsável por uma fase da construção do software. Essa abordagem é muito utilizada em projetos envolvendo *Outsourcing*, quando uma empresa é contratada para codificar o sistema a partir de sua especificação, ou para realizar os testes necessários. A vantagem dessa estratégia é contar com profissionais

especializados em cada fase, aproveitando as experiências deles para o projeto. A desvantagem está na passagem de uma fase para a outra, quando todas as informações precisam ser transferidas de uma equipe para a outra, criando curvas de aprendizado que afetam o tempo do projeto, e gerando problemas de comunicação decorrentes das diferenças culturais. Assim, nenhuma equipe tem conhecimento total do projeto e a necessidade de comunicação entre elas é intensificada, com impacto também sobre os custos do desenvolvimento. Essa é uma estratégia pouco adequada para projetos de Desenvolvimento Global de Software.

- **Divisão por especialidade dos desenvolvedores**: cada time desenvolve as características do software de acordo com sua área de especialidade (banco de dados, programação OO, desenvolvimento Web, etc). Assim como na divisão por fase de desenvolvimento, a vantagem está em aproveitar a competência e experiência dos participantes da melhor forma possível, com uma grande contribuição para o projeto. O problema é que os especialistas de cada área podem estar espalhados pelas diversas regiões geográficas envolvidas no projeto, e a ligação entre eles é exclusivamente tecnológica. Além de elevar os custos com comunicação, esse cenário exige um esforço maior de coordenação e integração do trabalho.

- **Divisão por versão do produto**: se o produto precisar ser adaptado para diferentes mercados ou diferentes plataformas, cada time pode ficar responsável por uma das versões do software. Como as adaptações necessárias geralmente estão associadas a questões culturais, cada time pode produzir a versão correspondente para seu mercado local, pois conhece o perfil dos usuários da região. No caso de versões para diferentes plataformas, cada time pode se concentrar nas particularidades de um ambiente específico. Essa estratégia pode ser adotada depois que a versão principal do software já estiver pronta.

Ainda é possível utilizar uma combinação dessas estratégias, aproveitando as vantagens de cada uma. Por exemplo, a divisão por funcionalidades pode levar em consideração as especialidades dos desenvolvedores de cada localidade.

Carmel [CARMEL, 1999] aponta ainda outra estratégia para a divisão de trabalho em projetos distribuídos, denominada *follow-the-sun*. A idéia é que os times trabalhem colaborativamente na construção de todo o sistema, em todas as suas fases, tirando proveito dos fusos-horários existentes entre as diversas localidades de desenvolvimento.

É como se o trabalho fosse ininterrupto, pois a jornada de um time começa quando a de outro está terminando, e assim por diante. Esse desenvolvimento 24 horas por dia, 7 dias por semana e 365 dias por ano seria muito conveniente para reduzir o tempo dos projetos, não fosse a complexidade envolvida na interação constante de pessoas que falam diferentes idiomas e que estão separadas geograficamente. Até a comunicação entre as equipes é prejudicada, pois conciliar horários para reuniões, mesmo que virtuais, utilizando vídeo-conferência, fica difícil.

Na escolha do critério de escalonamento de tarefas, vários autores [CARMEL, 2001] [HERBSLEB, 2001] [MOCKUS, 2001] indicam a redução na colaboração entre os times como uma boa estratégia para reduzir os esforços gerenciais. A idéia é simples: quanto mais dependente e integrado for o trabalho entre os times virtuais, mais difícil será administrar a comunicação entre eles e a integração de código, que precisarão ocorrer com uma freqüência maior. Por isso, as divisões por funcionalidades e por subsistemas são as mais recomendáveis para projetos de Desenvolvimento Global de Software.

3.2.2.5 CODIFICAÇÃO

Tomando como base o projeto arquitetural, os times começam a utilizar as ferramentas e tecnologias indicadas para codificar o software.

Durante a codificação, os times fazem uso de tecnologias colaborativas para tirarem dúvidas e trocarem resultados. A necessidade de comunicação entre os times dependerá principalmente do modelo de escalonamento de tarefas escolhido.

Carmel [CARMEL, 1999] classifica as tecnologias colaborativas em dois tipos: as tecnologias genéricas e as tecnologias de suporte à Engenharia de Software. As tecnologias genéricas se dividem em síncronas e assíncronas [CARMEL, 2001].

Tecnologias genéricas são aquelas que servem para a comunicação e a troca de informações entre os times. Podem ser síncronas, como vídeo-conferência, áudio-conferência, *e-chat* e *e-whiteboard*, ou assíncronas, como e-mail, *voice-mail*, v*ideo-mail*, listas de discussão, etc. Devido às diferenças de fusos-horários existentes em projetos globais, as tecnologias síncronas são menos utilizadas, apesar de bastante eficientes para permitir que os participantes do projeto se conheçam, aproximando assim os times.

As tecnologias de suporte à Engenharia de Software são utilizadas especificamente para ajudar os desenvolvedores no desempenho de suas atividades. Alguns exemplos são os gerenciadores de configuração de software, as ferramentas CASE e de programação, e as ferramentas para registro de defeitos e acompanhamento de mudanças no software.

Todas essas tecnologias são muito importantes para o sucesso dos times virtuais, e é fundamental que estejam disponíveis em todos os centros de desenvolvimento.

3.2.2.6 REALIZAÇÃO DE TESTES

O planejamento dos testes deve ocorrer paralelamente à codificação da aplicação. No caso do Desenvolvimento Global de Software, além dos testes funcionais obrigatórios, os desenvolvedores têm que se preocupar ainda com os testes de integração, de portabilidade e de usabilidade.

O código produzido pelos diversos times envolvidos no projeto precisa ser integrado para evitar que o software final apresente inconsistências ou falhas. Essa

integração costuma ser bastante complexa, principalmente quando as equipes utilizam plataformas e ferramentas diferentes.

O principal objetivo do teste de integração é verificar se a comunicação entre os módulos ou subsistemas da aplicação ocorre corretamente. No caso do teste de *browser*, o que se espera é garantir que o software poderá ser executado em diferentes ambientes computacionais (plataformas, sistemas operacionais, *browsers*, etc).

Como coletar os requisitos dos vários mercados não é suficiente para garantir a aceitação do software final, os testes de usabilidade também são muito importantes para observar as reações dos usuários, identificar falhas e obter sugestões de melhorias. Os testes de usabilidade podem começar com um protótipo da interface, e continuar, após a codificação, com a distribuição de versões beta para os usuários.

Em ambos os casos, a Internet pode ser usada para atingir um grande número de usuários: o protótipo pode ser disponibilizado para testes *on-line*, com o navegador como único requisito, e as versões beta podem ser distribuídas virtualmente em listas de discussão ou *sites*. Os usuários também podem usar e-mails ou *chats* para transmitir seus comentários.

É importante destacar que todos os testes são realizados e analisados de acordo com os critérios de aceitação definidos previamente para o projeto.

3.2.2.7 LOCALIZAÇÃO DE SOFTWARE

A Localização é uma estratégia bastante utilizada pelas empresas para conquistar os clientes globais. Localizar um produto para um mercado específico significa adaptá-lo para o perfil de seus usuários. Num mercado global, o idioma e as convenções culturais são os aspectos que merecem especial atenção, pois os valores dos usuários são diferentes.

Assim, na fase de transição do produto para o mercado, os desenvolvedores fazem adaptações não só em relação à interface do software (escolha de ícones, imagens, cores, fluxo de leitura e escrita, etc), mas também em relação a aspectos funcionais (escolha de idioma, formato de representação dos dados, plataforma, etc).

A Internacionalização é um processo complementar à Localização que, se aplicado durante o desenvolvimento, pode reduzir a complexidade envolvida na realização de mudanças no código finalizado. O objetivo é construir o software antecipando as possíveis alterações necessárias para permitir que diferentes versões sejam geradas em menor tempo e com custos reduzidos.

Essas duas estratégias podem ser aplicadas à interface e à documentação do software (manuais e tutoriais), e encontram-se detalhadas no Anexo A (Internacionalização e Localização de Software).

3.2.3 ASPECTOS GERENCIAIS

Sem dúvidas, a gerência de projeto é o grande desafio dos projetos de Desenvolvimento Global de Software. Se a alocação de recursos, a comunicação entre os desenvolvedores, a integração de código, e a gestão de mudanças no código já são difíceis em projetos com equipes presenciais, em projetos com times virtuais essa complexidade aumenta bastante.

O gerente de projeto precisa fazer o acompanhamento contínuo das equipes, mesmo estando elas dispersas, para evitar problemas na entrega da aplicação final.

3.2.3.1 ALOCAÇÃO DE RECURSOS

Uma grande dificuldade do desenvolvimento distribuído é prover os recursos necessários para as equipes em todos os centros de desenvolvimento.

Para que a integração do código produzido seja facilitada, uma medida fundamental é assegurar que sejam usados o mesmo hardware, os mesmos programas, e até as mesmas versões de software em todos os centros de desenvolvimento.

O problema é que essa unificação envolve custos para o projeto. Em desenvolvimentos presenciais, a licença de um software pode ser usada por vários membros de um projeto, mas em projetos distribuídos, são necessárias várias cópias do software, uma para cada centro de desenvolvimento.

Além disso, depois de prover a infra-estrutura inicial, a Organização Virtual teria ainda que realizar atualizações simultâneas em todos os centros sempre que uma nova ferramenta fosse incluída ou substituída no projeto. Além dos altos custos, a dificuldade gerencial para realizar essas mudanças seria outra limitação.

Entretanto, essa questão pode ser amenizada graças à Internet, que permite o uso compartilhado de ferramentas on-line. Quando uma cópia da ferramenta é instalada em um dos centros de desenvolvimento, todos os participantes do projeto podem ter acesso a ela remotamente, utilizando navegadores Web, sem que o software precise ser instalado nas máquinas locais [MURUGESAN, 1999].

Essa solução garante uma grande flexibilidade para os times, pois o uso de navegadores Web permite que eles tenham acesso aos recursos através de uma interface comum e independente de plataforma.

O mesmo princípio é aplicável ainda para a criação de repositórios de dados e bibliotecas de código disponíveis para todos. Inclusive, a criação de um repositório central, acessível pela *Web*, contendo todos os artefatos do projeto, é fortemente recomendada para evitar inconsistências e mudanças indevidas nas versões do software geradas por cada time [FAVELA, 2001].

Além das ferramentas de desenvolvimento, os recursos para a comunicação entre os times também precisam estar disponíveis em todos os centros de desenvolvimento.

Tecnologias de comunicação síncronas e assíncronas devem ser providenciados para que os times possam interagir mais de perto.

Assim como os recursos materiais, os recursos humanos são alocados de acordo com as necessidades do projeto. Os diferentes critérios utilizados para o escalonamento de tarefas discutidos no item 3.2.2.4 (Escalonamento de Tarefas) ajudam os gerentes de projeto a definir que profissionais serão necessários e em que momento do desenvolvimento.

É importante destacar mais uma vez que, quanto mais independente for o trabalho dos times, menores serão os custos com tecnologias colaborativas de comunicação, e menor será o esforço gerencial do projeto.

3.2.3.2 COMUNICAÇÃO ENTRE OS TIMES

O maior desafio para os gerentes de projeto é garantir que a comunicação entre os times ocorra, e que a distância não seja um obstáculo para a existência de um espírito de equipe compartilhado pelos membros do projeto. Os desenvolvedores devem se sentir parte do processo para que a motivação e o compromisso contribuam para o sucesso do trabalho.

Além do uso contínuo de tecnologias de comunicação, como vídeo-conferência, a realização de reuniões presenciais com representantes dos diferentes centros de desenvolvimento ajuda a aproximar as pessoas, que, apesar de não se conhecerem e terem culturas diferentes, precisam se unir por um objetivo comum. Mas, essa aproximação deve ser bem planejada para que o projeto não seja onerado desnecessariamente.

No relato das experiências de desenvolvimento global na Motorola [BATTIN, 2001], é destacada a importância de mediadores na fase de elaboração do projeto. O papel do mediador é atuar como representante do time que integra na tomada de decisões. Um bom mediador funciona como líder de sua equipe, comunicando as metas da Organização Virtual e observando seu cumprimento.

Como as equipes envolvidas em projetos globais costumam ser formadas por pessoas que não se conhecem, os mediadores servem também como elos de ligação entre os times, ajudando na construção do espírito de grupo e garantindo o comprometimento com o projeto.

Carmel e Agarwal [CARMEL, 2001] recomendam ainda a participação de mediadores culturais nos projetos. Esses mediadores seriam pessoas que conhecessem bem as culturas e idiomas dos times envolvidos no projeto, cuja função seria viajar pelas diversas localidades de desenvolvimento fazendo contatos com os líderes de cada unidade.

Os mediadores culturais facilitam a comunicação entre as equipes, resolvendo conflitos e desentendimentos causados pelo uso de idiomas distintos ou expressões mal-interpretadas. Num estudo envolvendo 20 projetos globais em 17 países diferentes, Carmel observou que 47 % dos times possuíam integrantes que atuavam como mediadores culturais [CARMEL, 1999].

Mais uma vez, a redução no trabalho colaborativo das equipes evita que problemas relacionados a falhas e desentendimentos na comunicação ocorram com freqüência.

3.2.3.3 INTEGRAÇÃO DE CÓDIGO

Apesar de desejável, nem sempre é possível ter times virtuais utilizando a mesma metodologia de desenvolvimento e as mesmas ferramentas.

Por isso, uma especificação de software bem elaborada facilita a integração do código, pois evita inconsistências e redundâncias no trabalho dos times.

Um modelo que vem ganhando adeptos em projetos de software é o Desenvolvimento Baseado em Componentes, que tem como base a modularização e o reuso, aspectos importantes para o Desenvolvimento Global de Software.

Um componente funciona como uma caixa-preta, que possui interfaces definindo os serviços que oferece e os parâmetros necessários para que o código seja executado [BRERETON, 2000].

Ou seja, o código gerado na construção de um componente não precisa ser conhecido para que ele possa ser usado ou integrado a outros componentes. Seu funcionamento é encapsulado, e o integrador se preocupa apenas com as interfaces de comunicação.

Dessa forma, a concepção do software pode ser vista como a montagem de blocos de construir que se encaixam [REPENNING, 2001]. Essa facilidade de reuso é um fator determinante para a redução do *time-to-market* das soluções desenvolvidas.

Paul Harmon [HARMON, 1998] destaca a existência de duas abordagens: o Desenvolvimento de Componentes e o Desenvolvimento Baseado em Componentes. A primeira corresponde à criação de módulos de software independentes, e o foco está em identificar características da lógica do negócio que possam ser reusadas em diversas aplicações. Esses componentes encontram-se disponíveis em repositórios virtuais na Internet, onde são vendidos ou distribuídos gratuitamente [MURUGESAN, 1999].

A segunda abordagem consiste na utilização de componentes já prontos na construção de novas aplicações, e é o modelo adotado no Desenvolvimento Global de Software. Harmon [HARMON, 1998] descreve um modelo de ciclo de vida específico para essa abordagem. A idéia é que as atividades de especificação, projeto e codificação do software sejam substituídas pela escolha, avaliação e integração de componentes [BRERETON, 2000].

O uso de componentes é muito indicado para o desenvolvimento distribuído por dois motivos principais [REPENNING, 2001]:

- As abordagens tradicionais costumam falhar na decomposição do software, causando duplicação de trabalho e desentendimentos entre os times virtuais. A definição do software como componentes integrados, cada um desempenhando um conjunto específico de atividades, facilita a divisão de trabalho entre as equipes.

- A natureza dos componentes força os projetistas e desenvolvedores a melhor encapsular as funcionalidades em módulos de software coesos e bem documentados. Além de facilitar a integração entre o trabalho dos times, essa prática contribui para o reuso e a adaptabilidade do software.

A realização de testes remotos de integração é outra preocupação dos gerentes de projetos virtuais. Mesmo utilizando o Desenvolvimento Baseado em Componentes, a integração dos módulos não deve ser deixada apenas para o final do projeto. Mesmo com a definição de módulos coesos e fracamente acoplados, é possível que haja falhas na implementação das equipes, causadas principalmente por problemas culturais e de comunicação.

O ideal é que os componentes sejam integrados continuamente, e os testes de integração sejam realizados em cada nova versão do software possibilitando a detecção de problemas tão logo eles apareçam.

3.2.3.4 GESTÃO DE MUDANÇAS

Esse certamente é o maior desafio nos desenvolvimentos virtuais. À medida que os times vão gerando novas versões para o software, um rígido controle deve existir para que, em qualquer dado instante, todos os times estejam trabalhando com uma mesma versão.

Caso contrário, muito tempo pode ser desperdiçado tentando corrigir novamente erros já removidos, ou ainda trabalhando com trechos de código que já não existem mais.

Além da atualização do código nos diferentes centros de desenvolvimento, a documentação das modificações e de suas justificativas é muito importante para que toda a equipe esteja ciente do estado atual do projeto.

Uma possível solução para esse problema seria a criação de um repositório central, acessível pela *Web*, contendo todos os artefatos do projeto [FAVELA, 2001]. Dessa forma, todos os times trabalhariam remotamente no mesmo código, não havendo redundâncias em cada centro de desenvolvimento. No Capítulo 6 (Testes e ferramentas para o desenvolvimento Web) serão sugeridas ferramentas já existentes no mercado para a sincronização dessas modificações.

Algumas lições para a gestão de mudanças no código podem ser aprendidas das práticas do Desenvolvimento de Código Aberto.

Utilizado principalmente na produção de aplicações gratuitas (*freewares*), esse tipo de desenvolvimento é caracterizado pelo livre compartilhamento do código fonte do

software através da Internet, permitindo que curiosos e profissionais de todo o mundo façam modificações e correções voluntariamente.

O sistema operacional Linux e o servidor Web Apache são exemplos de aplicações desenvolvidas com sucesso em projetos de código aberto.

No caso do servidor Apache, Jim Jagielski [JAGIELSKI, 1999] aponta as condutas que ajudaram, e continuam ajudando, na coordenação dos esforços de desenvolvimento:

- Cada desenvolvedor pode ter acesso à versão mais recente do software, disponível no *site* do projeto, e fazer uma cópia para sua máquina. As modificações feitas pelo desenvolvedor a partir desse momento alteram apenas a cópia local, não sendo refletidas para os outros desenvolvedores.

- As sugestões de mudança ou acréscimo de funcionalidades na versão oficial do software, propostas pelos desenvolvedores voluntários, devem ser submetidas à aprovação de um conselho de membros escolhidos para gerenciar o projeto (core members). Quando são aceitas, as modificações são incorporadas ao código, gerando uma nova versão oficial, que é disponibilizada no *site* do projeto com informações sobre as mudanças.

- Para manter-se atualizado, o desenvolvedor pode fazer visitas freqüentes ao *site* do projeto e obter as versões mais novas. Quando uma nova versão é copiada para a máquina local, uma ferramenta especial de controle de versões chamada CVS (*Concurrent Versions System*) evita que as alterações feitas anteriormente na versão local sejam perdidas. É feita uma comparação entre as versões, e os trechos de código atualizados ficam em destaque.

Nesse exemplo de modelo de desenvolvimento, o controle de mudanças e a integração de código são atividades restritas a um grupo de desenvolvedores, diminuindo assim os problemas de inconsistência e de comunicação.

3.3 Considerações Finais

Neste capítulo, foi introduzido o conceito do Desenvolvimento Global de Software, com a característica da produção de aplicações para diferentes mercados com a colaboração de times dispersos geograficamente.

Apontadas as vantagens desse tipo de desenvolvimento, o próximo passo foi avaliar as mudanças organizacionais decorrentes de sua adoção, com a formação de Organizações Virtuais e o envolvimento de Times Virtuais.

Logo em seguida, foram analisados os impactos da adoção do Desenvolvimento Global de Software em todo o ciclo de vida da aplicação, sob o ponto de vista das atividades de negócios, do desenvolvimento propriamente dito, e da gerência de projeto.

O Desenvolvimento Global de Software já vem sendo realizado há alguns anos, e sua forma mais conhecida é o Desenvolvimento de Código Aberto, que já produziu aplicações de sucesso como o servidor Apache e o sistema operacional Linux.

Entretanto, apesar de ser um tipo de desenvolvimento global, o Desenvolvimento de Código Aberto acontece num cenário bem mais flexível, sem pressões de mercado nem interesses de negócios. Não há despesas com pessoal nem infra-estrutura, já que os desenvolvedores são voluntários e utilizam seus próprios equipamentos, e não há definição de prazos para entrega de resultados. Também não há problemas com propriedade intelectual ou participação nos lucros, visto que aplicações de código aberto costumam ser distribuídas gratuitamente.

Por isso, as condutas adotadas nesses projetos não atendem com precisão as necessidades do Desenvolvimento Global de Software, que apresenta uma complexidade maior, mas podem servir de inspiração para que os desenvolvedores encontrem outras soluções.

Assim como ocorre com o Desenvolvimento de Aplicações Web, o Desenvolvimento Global de Software apresenta particularidades que precisam ser consideradas na definição do processo de desenvolvimento a ser seguido pelas equipes. Afinal, as atividades devem refletir os riscos e limitações do cenário em que são realizados os projetos.

O próximo capítulo apresenta algumas razões para o uso conjunto das duas abordagens, e uma análise resumida das melhores práticas para o Desenvolvimento de Aplicações Web e o Desenvolvimento Global de Software, que servirão como base para a proposta da metodologia *WideWorkWeb* no Capítulo 5.

Capítulo 4

4. As Aplicações Web e o Cenário Global

Conforme visto nos Capítulos 2 e 3, há diversos aspectos a serem analisados tanto no Desenvolvimento de Aplicações Web como no Desenvolvimento Global de Software. Essas duas abordagens vêm ganhando cada vez mais destaque nas empresas, muitas vezes de maneira conjunta.

A produção de Aplicações Web complexas, como as da categoria transacional (descrita em 2.1), requer o envolvimento de profissionais com diferentes competências para lidar com questões como processamento distribuído em diversos servidores, integração com sistemas legados, e confiabilidade e segurança no processamento dos dados [OFFUTT, 2002]. Essas competências nem sempre estão reunidas numa única região, ou mesmo numa única empresa, sendo uma oportunidade para a participação remota de outros profissionais e para a formação de parcerias entre organizações.

Além disso, tomando-se o caso específico das aplicações de comércio eletrônico, é bastante desejável que o maior número possível de usuários possa ter acesso ao sistema [OFFUTT, 2002]. Para isso, faz-se necessário oferecer versões da aplicação adaptadas às características de um público-alvo bastante heterogêneo, com aspectos culturais (idioma, formato de representação de datas e horários, convenções sociais, etc) que variam de uma região geográfica para outra. A proximidade dos desenvolvedores com os usuários permite a percepção dessas particularidades e facilita a realização de adaptações na aplicação durante e após o desenvolvimento (atividades de manutenção e suporte), constituindo, assim, outra boa razão para o desenvolvimento distribuído.

Além da busca de competências e do interesse em estar presente nos mercados-alvo da aplicação, há ainda outras boas razões para o Desenvolvimento Global de Software, como o interesse em reduzir custos com mão-de-obra incluindo participantes de países com maior oferta de pessoal qualificado, e a necessidade de integrar, num mesmo projeto, centros de desenvolvimento de empresas que passaram por processos de fusão ou aquisição de novas empresas, sem que o deslocamento dos funcionários seja necessário [HERBSLEB, 2001]. O envolvimento atual de mais de 50 nações em projetos de desenvolvimento colaborativo de software internacionalmente [CARMEL, 2001] não deixa dúvidas sobre a adesão das empresas a essa abordagem.

O objetivo do presente trabalho é apresentar uma proposta de modelo de desenvolvimento para projetos em que a combinação de fatores como complexidade das aplicações desenvolvidas, importância da proximidade com os mercados-alvo, e pressões de mercado para a formação de parcerias levam as empresas a adotarem simultaneamente

condutas do Desenvolvimento de Aplicações Web e do Desenvolvimento Global de Software.

Neste capítulo, será definido um cenário específico para a metodologia de desenvolvimento proposta, e serão evidenciados os principais problemas presentes em projetos dessa natureza. A partir da análise, sob um ponto de vista crítico, das soluções adotadas por empresas como Motorola, Alcatel e Bell-Labs, dentre outras, para essas questões, será possível obter um conjunto de boas práticas que poderão ser combinadas para compor a metodologia.

4.1 Cenário para a Metodologia de Desenvolvimento

Para caracterizar bem o cenário para o qual a metodologia será proposta, é importante analisar três fatores fundamentais: a natureza das aplicações produzidas, os objetivos de mercado do projeto, e a organização dos participantes.

- **Natureza da Aplicação:** o foco da metodologia será o desenvolvimento de Aplicações Web transacionais, especialmente sistemas de comércio eletrônico, visto que essas aplicações apresentam uma complexidade maior que outros tipos de aplicação Web, e requerem competências diversas, justificando a necessidade de participantes distribuídos. A complexidade das aplicações será refletida na escolha dos artefatos e na definição de papéis para os participantes, bem como na escolha de métricas e testes para o projeto.

- **Objetivos de mercado:** a metodologia será voltada para a produção de aplicações que tenham como público-alvo usuários culturalmente heterogêneos. Ou seja, deverão ser previstas atividades de adaptação das aplicações ao longo do desenvolvimento, além do uso de técnicas de Internacionalização e Localização de Software (Anexo A).

- **Organização dos participantes**: será considerada a participação de times de uma ou mais organizações, localizados em um ou mais países. Os times serão pequenos, com no mínimo três e no máximo oito participantes. A metodologia não contempla a participação remota de indivíduos isolados, apenas a colaboração de times distribuídos. Ou seja, cada centro de desenvolvimento deve ter um time como representante no projeto, em vez de um ou dois indivíduos apenas.

É importante destacar que a definição de um cenário específico para a metodologia não significa que ela não possa ser adotada nas demais circunstâncias, como a produção de outros tipos de aplicações Web, ou a escolha de um único mercado-alvo, ou ainda a participação de equipes fisicamente próximas. O objetivo é reproduzir um domínio de problema semelhante ao vivenciado por grandes empresas mundiais em projetos recentes, para que suas experiências possam servir como referência para a metodologia proposta.

4.2 Desenvolvimento Web: Melhores Práticas

Uma vez que as características das Aplicações Web e de seu desenvolvimento já foram bem detalhadas no Capítulo 2 (Desenvolvimento de Aplicações Web), e a adequação de processos mais flexíveis (como os Processos Agile) ao cenário Web já foi discutida neste mesmo capítulo, o próximo passo é analisar como as condutas propostas por esses processos podem ser úteis na elaboração de uma metodologia para o desenvolvimento de Aplicações Web transacionais por times geograficamente distribuídos.

Como destaca Ginige [GINIGE, 2001c], os requisitos não-funcionais de uma Aplicação Web precisam ser considerados desde o início do projeto, pois a qualidade da aplicação é determinante para seu sucesso. Uma boa estratégia para atingir esse objetivo é adotar algumas recomendações dos Processos Agile [AMBLER, 2002a], como a participação direta do cliente junto à equipe de desenvolvimento, opinando e discutindo características do software, e a contínua realização de testes na aplicação.

Entretanto, em projetos distribuídos, pode não ser possível ter uma colaboração tão intensa do cliente com os desenvolvedores, que não estarão mais reunidos numa única sala, mas espalhados em diversos centros ao redor do globo. Por isso, faz-se necessário definir desde o início do projeto, não apenas as características funcionais da aplicação, mas também alguns critérios de aceitação, indicando as aspirações do cliente em relação aos aspectos não-funcionais do software.

Exemplos de critérios de aceitação em relação à performance da aplicação seriam a definição do tempo máximo para a exibição de uma página, ou a quantidade máxima de usuários simultâneos que a aplicação deve suportar. De forma semelhante, poderiam ser definidos critérios para cada um dos aspectos relacionados à qualidade das Aplicações Web descritos no Capítulo 2 (Desenvolvimento de Aplicações Web). Esses critérios de aceitação passariam a ser tratados como metas de qualidade durante o desenvolvimento, e poderiam ser utilizados pelos times distribuídos na realização de testes de qualidade, sem a necessidade da participação direta do cliente durante o processo.

Já no caso da realização contínua de testes, é importante ter em mente que as Aplicações Web requerem testes diferenciados que possam ajudar na verificação de requisitos como usabilidade e segurança. O Capítulo 6 (Testes e Ferramentas para o Desenvolvimento Web) apresenta em detalhes as principais categorias de testes Web.

Considerando agora as características do desenvolvimento de Aplicações Web, é preciso analisar de que forma a mudança constante de requisitos, a realização de ciclos menores de desenvolvimento, a participação de times multidisciplinares, e a necessidade contínua de manutenção podem ser tratadas em projetos de Desenvolvimento Global de Aplicações Web.

De um modo geral, a maioria das condutas dos Processos Agile e da metodologia proposta por Ginige e Murugesan [GINIGE, 2001c] para o desenvolvimento de Aplicações Web (vide 2.3) se aplica também a projetos distribuídos. Algumas dessas práticas merecem

destaque, uma vez que servirão como diretrizes na definição de uma metodologia para o desenvolvimento de Aplicações Web num cenário global. São elas: ciclos rápidos de desenvolvimento, participação do cliente, técnicas gerenciais, técnicas de desenvolvimento, modularização da aplicação, escalonamento de atividades por competências e ênfase na manutenção do software.

4.2.1 CICLOS RÁPIDOS DE DESENVOLVIMENTO

A definição de ciclos curtos com escopo reduzido e prioridades definidas de acordo com os interesses de negócio do cliente é uma excelente alternativa para lidar com a constante mudança de requisitos dos projetos Web. O desenvolvimento da aplicação como um todo poderá prever a realização de vários ciclos, cada um com duração estimada em três a quatro semanas, e as user stories propostas por eXtreme Programming (XP) poderão ser usadas como base para a definição do escopo de cada ciclo.

4.2.2 PARTICIPAÇÃO DO CLIENTE

Como pregam os Processos Agile [AMBLER, 2002a], o acompanhamento do desenvolvimento pelo cliente é de suma importância para que a aplicação atenda os requisitos funcionais e não-funcionais pretendidos. Equipes com acesso direto ao cliente podem solucionar dúvidas com rapidez e evitar desentendimentos. Além disso, o feedback do cliente mantém os desenvolvedores motivados por saberem que estão obtendo sucesso.

Nos projetos distribuídos, é provável que a participação do cliente não possa se dar de forma tão freqüente quanto em projetos presenciais, visto que diferentes clientes poderão estar envolvidos (representantes das diferentes organizações) e dificilmente poderão estar presentes em todos os centros de desenvolvimento. Para amenizar esse problema, uma boa estratégia seria tentar obter o máximo de detalhes sobre as metas do projeto na fase inicial do desenvolvimento, e utilizar tecnologias de comunicação como e-mail, fax, áudio e vídeo-conferência para agilizar os contatos remotos com o cliente sempre que necessário.

4.2.3 TÉCNICAS GERENCIAIS

Tanto Scrum [SCHWABER, 1995] como XP [BECK, 2000] propõem a realização de reuniões rápidas e diárias com todos os membros da equipe para avaliar o progresso do projeto e detectar possíveis riscos ou problemas que precisem ser resolvidos. Além de serem uma boa oportunidade para a coleta de métricas, essas reuniões permitem que todos os desenvolvedores tenham uma visão geral do projeto e percebam a importância de sua contribuição para o objetivo final, contribuindo para sua motivação. Essas reuniões costumam ser dirigidas por um dos membros da equipe designado como líder ou gerente de projeto (Scrum Master) para coordenar os demais participantes e acompanhar o desenvolvimento.

É importante destacar que a ênfase na comunicação entre os participantes como alternativa para reduzir a necessidade de documentação e agilizar o desenvolvimento proposta pelos Processos Agile pode não funcionar em projetos distribuídos, uma vez que a

distância e as diferenças culturais podem ser empecilhos para a integração total das equipes. Assim, o processo de desenvolvimento adotado na produção de Aplicações Web por times distribuídos deve prever uma documentação consistente para evitar falhas na comunicação e mal-entendidos.

4.2.4 TÉCNICAS DE DESENVOLVIMENTO

Como mostrou Frank Maurer [MAURER, 2002], as técnicas de desenvolvimento propostas por eXtreme Programming (planejamento, metáfora, projeto simples, testes, refatoramento, programação em pares, propriedade coletiva, integração contínua, quarenta horas semanais, presença do cliente e padrões de codificação) se ajustam muito bem às necessidades dos projetos Web. Entretanto, algumas técnicas precisam ser adaptadas para a realização de projetos distribuídos com foco em diferentes mercados.

É o caso, por exemplo, da técnica Projeto Simples, cuja proposta é incluir apenas o essencialmente necessário na hora em que a aplicação está sendo desenvolvida, sem adicionar código que antecipe futuras necessidades do cliente. Embora o objetivo dessa medida seja simplificar ao máximo a aplicação, essa simplicidade pode sair caro, caso a aplicação precise ser adaptada para mercados culturalmente heterogêneos posteriormente. Isto porque, caso não sejam adotadas desde o começo técnicas de Internacionalização como o isolamento entre a interface e o código da aplicação e o uso de padrões como o Unicode, os custos para adaptar a aplicação e o tempo necessário para essas modificações podem significar a perda de importantes oportunidades de negócios. Por isso, mesmo que a ampliação de mercados não seja a idéia inicial, é preciso considerar essa possibilidade e preparar o código de antemão para que as mudanças sejam feitas prontamente caso necessário.

Outra técnica que pode não ser viável em projetos distribuídos é a programação em pares, uma vez que os times estarão distribuídos e os poucos participantes presentes em cada centro de desenvolvimento poderão ter competências diferentes, participando assim de diferentes atividades dentro do ciclo. Embora haja relatos de projetos que adotaram a programação em pares distribuída, com a colaboração remota e simultânea dos participantes [KIRCHER, 2001], as dificuldades parecem sobrepor os benefícios, principalmente considerando-se que os participantes de projetos distribuídos podem apresentar diferenças culturais, a começar pelo idioma, consistindo em uma dificuldade adicional para a colaboração entre eles.

4.2.5 MODULARIZAÇÃO DA APLICAÇÃO

Ginige e Murugesan [GINIGE, 2001c] propõem que as Aplicações Web sejam vistas como uma composição de dois elementos distintos: o conteúdo das páginas (os dados, as mídias, etc) e o software necessário para exibir o conteúdo e oferecer serviços aos usuários. Essa é uma abordagem interessante, pois além de isolar os dados da interface na aplicação, facilitando o reuso e a adaptação do produto para diferentes mercados, permite a concepção do software como um conjunto de módulos que podem ser desenvolvidos

paralelamente por diferentes profissionais para depois comporem a aplicação final. É o uso do Desenvolvimento Baseado em Componentes na produção de Aplicações Web.

É possível ainda fazer uma distinção em relação ao software das Aplicações Web, considerando o código da interface propriamente dita (design das páginas, scripts para geração dinâmica de páginas, menus de navegação, etc) e o código para manipulação dos dados (scripts de acesso ao banco de dados para realização de consultas e atualizações, código para validação de dados, etc) como elementos independentes. Assim, seria possível aumentar as chances de paralelismo no desenvolvimento, com os componentes de cada elemento (conteúdo, interface e código) sendo produzidos por diferentes profissionais.

4.2.6 ESCALONAMENTO DE ATIVIDADES POR COMPETÊNCIAS

Assim como propõem a modularização das Aplicações Web, Ginige e Murugesan [GINIGE, 2001c] sugerem uma divisão nas equipes de desenvolvimento de acordo com as competências necessárias para a produção de cada elemento do software. Enquanto o conteúdo costuma ser gerado por jornalistas e profissionais com experiência em marketing e relações públicas, e o design das páginas é produzido por especialistas em computação gráfica e artes visuais, o código com as funcionalidades da aplicação e os bancos de dados são de responsabilidade de profissionais de Tecnologia da Informação e áreas afins. Assim, é possível aproveitar melhor as habilidades de cada profissional, contribuindo para uma melhor qualidade da aplicação como um todo.

Entretanto, apesar de bastante desejável, a adoção dessa estratégia pode não ser viável em projetos envolvendo equipes muito pequenas em que não seja possível alocar profissionais para cada perfil necessário. Nesses casos, os profissionais acabam acumulando papéis para atender as restrições do projeto.

É preciso ressaltar ainda que o envolvimento de participantes com diferentes perfis no projeto deve ser considerado na hora de escolher o processo de desenvolvimento que será adotado. Isto porque os profissionais de outras áreas costumam ter dificuldade em se adaptar aos formalismos da Engenharia de Software, como a produção excessiva de documentação. Assim, é desejável que o processo adotado seja simples e objetivo, com a simplificação de fases, nomenclaturas e artefatos, para que todos possam se adaptar mais facilmente.

4.2.7 ÊNFASE NA MANUTENÇÃO DO SOFTWARE

Ainda na metodologia proposta por Ginige e Murugesan [GINIGE, 2001c], há um destaque para a fase de manutenção da aplicação. Embora muitos processos de desenvolvimento considerem a manutenção apenas como uma atividade de pós-venda do software, no caso das Aplicações Web é preciso levar em conta que a manutenção ocorre constantemente, promovendo a evolução da aplicação de acordo com as necessidades do mercado. Assim, cada ciclo de manutenção corresponde ao início de um novo ciclo de desenvolvimento, onde as atividades serão definidas de acordo com as alterações necessárias na aplicação.

É preciso considerar ainda que existem diferentes tipos de manutenção. Ginige e Murugesan caracterizam três categorias principais que ocorrem em Aplicações Web: a manutenção de conteúdo (atualizações de dados, criação de novas mídias, etc), a manutenção de software (acréscimo de funcionalidades, mudança de tecnologias, etc) e a manutenção de hardware e de rede (correção de falhas, atualização de versões das ferramentas, etc). Técnicas como a modularização do software e o Desenvolvimento Baseado em Componentes ajudam a tornar os ciclos de manutenção mais ágeis, facilitando o reuso e a adaptação da aplicação aos novos requisitos.

4.2.8 ADAPTAÇÃO DAS SOLUÇÕES

Conforme visto, as principais soluções propostas pela comunidade de software para atender as necessidades do desenvolvimento de Aplicações Web podem também ser utilizadas em projetos distribuídos e com foco no mercado global, requerendo apenas alguns ajustes, que podem ser resumidos da seguinte maneira:

- → A adoção de técnicas de Internacionalização deve ocorrer desde o começo do projeto, mesmo que a adaptação da aplicação para diferentes mercados não esteja inicialmente prevista, em oposição à recomendação de Projeto Simples dos Processos Agile. Isto porque os custos para realizar as mudanças depois seriam muito altos, e o tempo necessário para adaptar a aplicação e gerar versões para outros mercados poderia limitar as oportunidades de negócios.

- → O uso de um código claro e bem comentado pode não ser suficiente para garantir o entendimento de todos os participantes do projeto, visto que eles se encontram separados por barreiras geográficas e culturais e não podem ter uma comunicação tão efetiva quanto os times ditos presenciais. Assim, é recomendado o uso de uma documentação mais consistente do projeto, na forma de relatórios ou comunicados, para estabelecer mudanças nos requisitos ou prazos previamente definidos, e permitir o acompanhamento remoto das atividades das equipes.

4.3 Desenvolvimento Global de Software: Melhores Práticas

No Capítulo 3 foram discutidos os principais impactos do Desenvolvimento Global de Software sobre a organização dos times e das empresas, e sobre o ciclo de vida das aplicações produzidas em relação aos negócios, às atividades do desenvolvimento e à gerência dos projetos. Também foram discutidas as soluções adotadas por algumas empresas multinacionais para lidar com os principais desafios dos projetos globais.

Como o objetivo deste trabalho é propor uma metodologia para o desenvolvimento de Aplicações Web transacionais por times geograficamente distribuídos, e as melhores práticas para a produção de Aplicações Web já foram apontadas no item 4.2, é preciso agora destacar as condutas de grandes empresas como Motorola e Alcatel, dentre outras,

que relataram suas experiências em projetos semelhantes [AOYAMA, 1998] [BATTIN, 2001] [EBERT, 2001] [FAVELA, 2001] [GAO, 1999] [HERBSLEB, 2001] [MOCKUS, 2001] [REPENNING, 2001]. Suas experiências certamente serão de grande valia na definição da nova metodologia.

A seguir, serão discutidas as melhores práticas para um processo de Desenvolvimento Global de Software considerando seis aspectos importantes: definição do processo de desenvolvimento, escalonamento de atividades, modularização do software, escolha de ferramentas, presença de mediadores, acesso aos artefatos.

4.3.1 DEFINIÇÃO DO PROCESSO DE DESENVOLVIMENTO

Quando times distribuídos estão trabalhando num mesmo projeto, o grande desafio é garantir que todos tenham a mesma visão do projeto e cumpram suas metas nos prazos esperados, principalmente se estes participantes são membros de diferentes empresas, cada uma com uma metodologia de desenvolvimento própria.

Alcatel e Motorola passaram por esse problema e adotaram soluções bastante distintas. Enquanto a Alcatel optou pelo uso de um rigoroso processo CMM nível 3 em todos os centros de desenvolvimento para facilitar a comunicação entre os participantes e unificar a produção de artefatos e a seqüência de realização das atividades, a Motorola preferiu permitir que cada um de seus centros adotasse a metodologia que lhe fosse mais familiar, sob o argumento de que a imposição de um processo único criaria curvas de aprendizado correspondentes ao tempo que as equipes levariam para se adaptar à metodologia nova. Para lidar com as diferenças, foram definidos um vocabulário e um conjunto de artefatos padrão que poderiam ser facilmente adotados pelos times, cada um seguindo sua metodologia.

A solução adotada pela Motorola certamente é mais flexível e seria mais indicada para projetos em que diferentes empresas estão envolvidas, uma vez que a padronização total do processo seria inviável. Entretanto, é preciso considerar que, mesmo utilizando um vocabulário e um conjunto de artefatos padrão, será bastante difícil manter o sincronismo da equipe e acompanhar a evolução do projeto caso cada time siga uma metodologia própria. No caso da Motorola, as empresas participantes eram todas filiais da instituição e, portanto, a cultura organizacional era comum a todas as equipes.

Já no caso de projetos com empresas diferentes, principalmente no caso da aquisição de pequenas empresas por grandes corporações, pode ocorrer de equipes acostumadas a adotar prática CMM nível 3 terem que colaborar com equipes que não adotam nenhum processo de desenvolvimento específico, mas realizam suas atividades ainda de modo ad hoc. Nesse caso, permitir que cada time utilize seu próprio processo poderia trazer mais conflitos do que soluções, além de por em risco o sucesso do projeto.

Assim, a melhor conduta na hora de escolher o processo de desenvolvimento a ser seguido seria uma solução híbrida das propostas da Alcatel e da Motorola:definir uma metodologia padrão para ser utilizada por todas as equipes que fosse simples e intuitivo o

suficiente para não exigir curvas de aprendizado. Ou seja, um conjunto mínimo de atividades a serem seguidas e artefatos a serem gerados que atendesse as necessidades do projeto e não exigissem muitas adaptações para os times. Dessa forma, a comunicação e o sincronismo dos participantes poderiam ocorrer mais facilmente e os prazos do projeto não seriam afetados.

4.3.2 ESCALONAMENTO DE ATIVIDADES

Como foi discutido no Capítulo 3, há várias propostas de estratégias para a alocação de atividades entre os participantes de projetos virtuais: a divisão por funcionalidades do software, a divisão por subsistema, a divisão por fase de desenvolvimento, a divisão por especialidade dos desenvolvedores e a divisão por versão do produto.

Entretanto, todos os autores [CARMEL, 1999][EBERT, 2001][BATTIN, 2001][MOCKUS, 2001][REPENNING, 2001][BENNATAN, 2002] são unânimes em sugerir que a melhor forma de executar e gerenciar um projeto de desenvolvimento distribuído é permitir que os times trabalhem de forma independente, reduzindo a necessidade de comunicação entre os participantes e favorecendo o paralelismo na execução das atividades do projeto. Segundo os autores, essa é a tática ideal para evitar problemas como dificuldades na comunicação entre participantes com culturas e fusos-horários diferentes, falta de sincronismo das equipes, atrasos e necessidade de re-trabalho devido ao mau desempenho de uma das equipes ou a equívocos na interpretação dos requisitos. É também uma forma de intensificar a comunicação, o entrosamento e a motivação dentro de cada time, uma vez que as pessoas já se conhecem e possuem afinidades, para que todos possam contribuir da melhor maneira possível para o sucesso do projeto.

Para que ocorra essa independência entre os times, a estratégia de escalonamento recomendada é a divisão por funcionalidades do software. Assim, cada time se responsabiliza pelo atendimento a um conjunto de requisitos relacionados, e as equipes podem trabalhar em paralelo, desde que tenham uma visão do projeto como um todo e da importância de sua contribuição para a aplicação final. É importante destacar que deve ocorrer uma replicação de papéis em cada centro de desenvolvimento [EBERT, 2001]. Ou seja, para que a independência das equipes seja completa, cada time deve ter gerente de projeto, desenvolvedores, especialistas em testes, etc.

No caso específico de projetos de desenvolvimento distribuído de Aplicações Web, a divisão de tarefas por funcionalidades pode ser útil entre os times, mas dentro de cada time, é preciso adotar uma outra estratégia: a divisão por especialidade dos desenvolvedores. Afinal, como já discutido anteriormente neste capítulo, a produção dos diferentes elementos de uma Aplicação Web (conteúdo, interface e código) exige competências de profissionais de diversas áreas e com perfis bastante distintos. Assim, a metodologia de desenvolvimento que será proposta no próximo capítulo deve prever a combinação dessas duas estratégias: a divisão por funcionalidades entre os times e divisão por competências ou especialidades dentro de cada time.

4.3.3 MODULARIZAÇÃO DO SOFTWARE

Para que a independência entre o trabalho dos times possa ocorrer, é importante que a aplicação seja concebida desde o começo na forma de módulos coesos – que encapsulam funcionalidades – e fracamente acoplados – que apresentam um certo grau de independência em suas interações [CARMEL, 1999]. Além de garantir o paralelismo, os módulos contribuem para o reuso das aplicações, pois podem ser armazenados em bibliotecas para compor futuros projetos.

Outra boa estratégia de modularização é o Desenvolvimento Baseado em Componentes. Um componente funciona como uma caixa-preta, que possui interfaces definindo os serviços que oferece e os parâmetros necessários para que o código seja executado [BRERETON, 2000]. Ou seja, o código gerado na construção de um componente não precisa ser conhecido para que ele possa ser usado ou integrado a outros componentes. Seu funcionamento é encapsulado e são definidas interfaces de comunicação, que os desenvolvedores consideram na elaboração de cada componente do sistema.

Dessa forma, a concepção do software pode ser vista como a montagem de blocos de construir que se encaixam [REPENNING, 2001]. No caso do desenvolvimento de Aplicações Web, esses componentes são definidos não apenas para o código que provê as funcionalidades, mas também para o conteúdo e para a interface da aplicação na forma de mídias, elementos gráficos e formulários eletrônicos.

4.3.4 ESCOLHA DE FERRAMENTAS

Em um projeto de desenvolvimento distribuído de Aplicações Web, o uso de ferramentas adequadas na realização das atividades de construção da aplicação, e, principalmente, na gerência do projeto é de fundamental importância. De um modo geral, a partir da análise da experiência de várias empresas autores [CARMEL, 1999][EBERT, 2001][BATTIN, 2001][MOCKUS, 2001][REPENNING, 2001][FAVELA, 2001][HERBSLEB, 2001], é possível destacar três categorias principais de ferramentas necessárias em projetos dessa natureza: ferramentas de comunicação, ferramentas de desenvolvimento e ferramentas de gestão.

As ferramentas de comunicação, como e-mail, chat, áudio e vídeo-conferência, são muito importantes para reduzir a distância entre os times, garantindo um bom entendimento entre os participantes e propiciando a discussão de aspectos críticos do projeto. O objetivo das ferramentas de desenvolvimento é auxiliar as equipes na produção dos três elementos que compõem as Aplicações Web: o código, o conteúdo e a interface. Assim, editores de código, bancos de dados, compiladores, editores gráficos, ferramentas para a criação e manipulação de áudio e vídeo, e, principalmente, ferramentas para automatização de testes são fundamentais para que as equipes possam desempenhar com sucesso e rapidez suas atividades. Já as ferramentas de gestão, como as utilizadas para o planejamento de projetos, o escalonamento de atividades, o controle de versões e o acompanhamento de métricas, são necessárias para que se tenha uma visão uniforme do trabalho das equipes e para que não haja redundância na produção de artefatos.

De um modo geral, é desejável que as equipes utilizem as mesmas ferramentas em cada centro de desenvolvimento para facilitar a integração dos resultados e a comunicação entre os participantes [CARMEL, 1999]. Como isto implicaria num alto custo com a aquisição, instalação e suporte de ferramentas para cada equipe, a melhor alternativa é optar por ferramentas gratuitas ou de código aberto disponíveis no mercado.

4.3.5 PRESENÇA DE MEDIADORES

Assim como o gerente de projeto nos processos de desenvolvimento tradicionais, o mediador atua coordenando as atividades, traçando as metas e medindo a qualidade dos resultados que vão sendo produzidos em projetos distribuídos. É desejável que cada time tenha seu próprio mediador local, para que ele possa acompanhar de perto os participantes e não tenha problemas para se adaptar culturalmente à empresa ou aos costumes da região.

Alguns autores [CARMEL, 1999] [BATTIN, 2001] destacam a importância do uso de mediadores como intermediários na comunicação com o cliente e com os demais times, e como representantes das equipes na tomada de decisões sobre o projeto, principalmente na fase inicial. Assim, além de serem líderes de suas equipes, motivando e auxiliando os participantes na solução de problemas, os mediadores atuam como pontos de integração entre as equipes, trocando informações sobre o progresso das atividades e interagindo diretamente com o cliente na hora de esclarecer dúvidas ou medir a qualidade da aplicação em relação aos requisitos previamente definidos.

Carmel [CARMEL, 1999] destaca ainda a necessidade da participação de uma outra categoria de mediador em alguns projetos: o mediador cultural. Seu papel seria fazer a ponte entre as diferentes culturas existentes no projeto, não só em relação ao idioma, mas também em relação a convenções sociais. De acordo com Carmel, o mediador cultural atuaria na integração dos diferentes centros de desenvolvimento, viajando sempre que necessário para fazer contatos com os mediadores de cada equipe.

4.3.6 ACESSO AOS ARTEFATOS

Para que os times distribuídos possam ter uma visão completa e atualizada sobre o progresso do desenvolvimento, é importante que eles tenham acesso a informações do projeto, como cronogramas e métricas de acompanhamento, e também aos artefatos produzidos. Como destacam vários autores [CARMEL, 1999][EBERT, 2001][BATTIN, 2001][FAVELA, 2001], a existência de um repositório comum de artefatos, acessível a todas as equipes através da Web, facilita o controle de versões do software e evita inconsistências, pois as mudanças realizadas são disponibilizadas para todas as equipes.

Além disso, os times se sentem motivados por perceberem a importância de sua participação no projeto e por observarem o avanço das atividades. Disponibilizar todos os artefatos e a documentação gerada na Web permite que todos tenham uma interface comum de acesso, independente da plataforma e das ferramentas utilizadas por cada time.

4.4 Considerações Finais

Enquanto o objetivo dos Capítulos 2 e 3 foi apresentar uma visão geral sobre o Desenvolvimento de Aplicações Web e o Desenvolvimento Global de Software, respectivamente, apontando suas características, implicações e problemas, a finalidade do presente capítulo era analisar de forma clara e objetiva como as soluções adotadas por grandes empresas em projetos com essas abordagens poderiam ser combinadas de forma a compor uma metodologia única para o Desenvolvimento de Aplicações Web num Cenário Global. Assim, foram apontadas as treze melhores práticas de desenvolvimento para projetos dessa natureza, e foram discutidas as adaptações necessárias para um projeto envolvendo as duas abordagens.

Além disso, no início do capítulo, foram apontadas algumas razões para o uso conjunto dessas duas abordagens, e foi caracterizado o cenário de atuação da metodologia que será proposta neste trabalho. A metodologia será destinada à produção de Aplicações Web Transacionais para mercados culturalmente heterogêneos por times pequenos, de regiões geográficas e empresas diferentes.

O próximo capítulo apresenta a metodologia *WideWorkWeb*, definindo seu fluxo de atividades, seu conjunto de artefatos e sua coleção de métricas de acompanhamento, dentre outras características.

Capítulo 5

5. A Metodologia *WideWorkWeb*

No capítulo anterior, foram apontadas algumas das melhores práticas do Desenvolvimento de Aplicações Web e do Desenvolvimento Global de Software, bem como razões para o uso conjunto dessas abordagens.

Esses dois tipos de desenvolvimento apresentam muitas semelhanças, a começar pela heterogeneidade do público-alvo que se propõem a atingir. O objetivo é produzir aplicações que possam atender as necessidades do maior número possível de usuários, mesmo que eles estejam localizados em diferentes países. Para isso a Internet e as tecnologias de comunicação são as grandes aliadas.

A ênfase dada aos requisitos não-funcionais das aplicações é outra característica comum. Nos dois casos, os desenvolvedores buscam a portabilidade e a adaptabilidade do código como forma de promover o reuso, minimizando os esforços e os custos da produção de diferentes versões, com ajustes na interface e nos aspectos técnicos, para cada mercado.

A principal semelhança entre esses dois tipos de desenvolvimento é a inadequação dos processos tradicionais da Engenharia de Software às suas particularidades. Por não disporem de uma definição de fases e procedimentos coerente com suas realidades, os desenvolvedores acabam optando por práticas *ad hoc* de desenvolvimento, no caso do Desenvolvimento de Aplicações Web, ou iniciam os projetos sem uma noção clara dos riscos envolvidos e dos investimentos necessários, no caso do Desenvolvimento Global de Software.

Como conseqüência, os projetos acabam estourando prazos e orçamentos, e a qualidade das aplicações produzidas é prejudicada, reduzindo assim as possibilidades de sucesso do produto no mercado.

O objetivo deste capítulo é apresentar uma proposta de metodologia para o Desenvolvimento de Aplicações Web num cenário Global que atenda as necessidades dos desenvolvedores, uma vez que, conforme visto no Capítulo 4 (As Aplicações Web e o Cenário Global), são muitas as razões para o uso combinado dessas abordagens. Essa metodologia será baseada nas práticas descritas no Capítulo 4.

Inicialmente, serão descritos o cenário e os valores que servirão como base para o processo. Em seguida, será apresentada uma seqüência de atividades a serem seguidas pelas equipes de desenvolvimento.

Serão definidos ainda os papéis que os envolvidos devem desempenhar durante o projeto e os artefatos que precisam ser gerados, bem como técnicas gerenciais e métricas de acompanhamento para dar suporte ao processo.

Finalmente, será apresentada uma recomendação para a definição do ciclo de vida das Aplicações Web, tendo como base os diferentes tipos de manutenção que podem ser aplicados a esse tipo de software.

5.1 Metodologia para o Desenvolvimento de Aplicações Web num Cenário Global

O principal objetivo em se definir uma metodologia para a produção de software é auxiliar os desenvolvedores na avaliação dos riscos dos projetos, maximizando as chances de sucesso, e na realização efetiva das atividades no menor espaço de tempo e, se possível, com custos reduzidos.

Para isso, a metodologia proposta deve definir claramente um conjunto de passos a serem seguidos, deve ser mensurável e passível de acompanhamento, e deve facilitar o refinamento e a evolução da aplicação de acordo com o feedback dos usuários [GINIGE, 2001c].

Também é fundamental que exista um ambiente computacional adequado para dar suporte à realização das atividades descritas pela metodologia. No Capítulo 6 (Testes e ferramentas para o desenvolvimento Web) são indicadas algumas ferramentas disponíveis atualmente no mercado para auxiliar os desenvolvedores. A construção de um ambiente integrado para apoiar a metodologia aqui sugerida será fruto de trabalhos futuros.

Tendo como base essas diretrizes e as particularidades do Desenvolvimento de Aplicações Web e do Desenvolvimento Global de Software, será apresentada a seguir a Metodologia *WideWorkWeb* para o desenvolvimento de Aplicações Web num cenário global.

5.1.1 CENÁRIO

É importante destacar que o objetivo deste trabalho não é propor uma metodologia genérica para projetos de Desenvolvimento de Aplicações Web, nem tampouco uma metodologia eficaz para projetos de Desenvolvimento Global de Software. O que se deseja é propor um conjunto de práticas que possam ser utilizadas em projetos com características específicas que precisem combinar essas duas abordagens devido a pressões de mercado ou circunstâncias de negócios.

A justificativa para a definição de um cenário específico para a metodologia é reproduzir um domínio de problema semelhante ao vivenciado por grandes empresas mundiais em projetos recentes, para que suas experiências possam servir como referência para a metodologia proposta, uma vez que esta ainda não foi validada formalmente.

Conforme apresentado no Capítulo 4 (As Aplicações Web e o Cenário Global), o cenário para a metodologia *WideWorkWeb* deve ser analisado em relação a três aspectos: natureza da aplicação, objetivos de mercado e organização dos participantes.

A metodologia de desenvolvimento proposta tem como foco o desenvolvimento de Aplicações Web transacionais, especialmente sistemas de *e-commerce*. As justificativas para essa escolha são a evidente complexidade desse tipo de software em relação às demais categorias, demandando o esforço conjunto de diferentes parceiros, e a conseqüente necessidade de uma definição mais completa e detalhada das atividades, de modo a minimizar riscos.

Acredita-se que, uma vez apresentada uma metodologia eficiente para o desenvolvimento de Aplicações Web transacionais, esta poderá ser facilmente ajustada para as demais categorias, com a simplificação, ou até supressão, de algumas atividades, refletindo a diminuição da complexidade.

Em relação aos objetivos de mercado, a metodologia considera a produção de aplicações para diferentes mercados-alvo, que apresentem usuários culturalmente heterogêneos. A principal implicação dessa escolha é a necessidade de adaptações do software ao longo do desenvolvimento, com técnicas de Internacionalização, e após o desenvolvimento, com ciclos de Localização para adaptar a aplicação a cada mercado. Outra conseqüência é a escolha de modelos de desenvolvimento que favoreçam o reuso e a adaptabilidade, como o Desenvolvimento Baseado em Componentes.

Quanto à organização dos desenvolvedores, a metodologia considera a colaboração entre times pequenos, de no mínimo três e no máximo oito participantes, que podem estar distribuídos, localizados em um ou mais países, e podem ainda pertencer a uma ou mais organizações. Essa flexibilidade permite que o processo seja adaptado tanto para projetos pequenos, envolvendo diferentes unidades de uma única empresa, distribuídas em uma mesma cidade, como para projetos maiores, com parcerias entre empresas intercontinentais.

A opção por um número pequeno de participantes em cada time tem o objetivo de promover a comunicação entre eles, facilitando, assim, os esforços gerenciais.

Para que o trabalho dos times possa ser o mais homogêneo possível e a integração de resultados entre eles ocorra sem grandes dificuldades, a metodologia proposta é bastante simples, com um conjunto resumido de atividades, de modo a poder ser adotada desde o início por todos os times, sem a necessidade de curvas de aprendizado que atrasem o projeto.

5.1.2 VALORES

Assim como ocorre com os Processos Agile [AMBLER, 2002a], a metodologia proposta também tem como base alguns princípios importantes que norteiam as decisões de desenvolvimento. São eles: simplicidade, interatividade, agilidade, e flexibilidade.

❖ **Simplicidade**

Como o Desenvolvimento de Aplicações Web envolve profissionais de diferentes áreas, é bastante desejável que o processo adotado durante o projeto seja simples e conciso, para que não haja resistência por parte dos especialistas pouco familiarizados com as práticas de Engenharia de Software.

Para que sejam seguidas intuitivamente, as práticas sugeridas devem refletir a realidade dos desenvolvedores e fugir da formalidade excessiva. A documentação excessiva presente nos processos tradicionais, com a produção de uma infinidade de artefatos durante o desenvolvimento, deve ser evitada, pois além de atrasar e encarecer os projetos, é rejeitada pela maioria dos desenvolvedores.

Por isso, a presente metodologia define uma lista mínima de documentos realmente indispensáveis para as equipes envolvidas no Desenvolvimento Global de Aplicações Web, descritos no item 5.1.7 (Artefatos).

❖ **Interatividade**

Durante todo o desenvolvimento, é muito importante que o cliente participe do projeto ativamente, dando sugestões, tirando dúvidas e avaliando resultados. Essa comunicação direta com os desenvolvedores garante uma maior motivação para as equipes, que sentem que estão atingindo os objetivos planejados, e uma maior satisfação para o cliente, que obterá ao final do projeto uma aplicação adequada a suas necessidades.

Mas, além dos contatos com o cliente, é fundamental que haja também uma boa interação dentro de cada time e entre os times. A comunicação deve ser estimulada, mesmo entre participantes dispersos geograficamente e com diferenças culturais, pois ajuda a criar um espírito de equipe e fortalece a confiança nos demais membros do projeto.

❖ **Agilidade**

Uma vez que a pressão do time-to-market é maior no Desenvolvimento de Aplicações Web, e as mudanças nos requisitos do cliente e nas tecnologias são constantes, é importante que os times se organizem para entregar resultados que agreguem valor ao negócio dos clientes em espaços de tempo cada vez menores.

Para conseguir isso, os times precisam de um processo de desenvolvimento que contemple um conjunto reduzido de atividades, para que o ciclo de produção seja rapidamente vencido. Também é fundamental que haja uma redução no escopo de cada etapa, para que os times consigam concentrar seus esforços e garantir a qualidade do software.

❖ **Flexibilidade**

A escolha de um processo para o Desenvolvimento Global de Aplicações Web é bastante influenciada pela complexidade da aplicação a ser produzida e também pela estrutura de organização dos times, seja em diferentes países, seja em diferentes empresas.

Pensando nisso, a metodologia aqui apresentada foi formulada de modo a permitir sua adaptação para diferentes cenários de desenvolvimento. O fluxo de atividades proposto não precisa ser seguido completamente pelos times, que podem optar apenas por um subconjunto delas. Mais que isso, a ênfase dada a cada etapa pode variar de acordo com o tipo de projeto, cabendo aos desenvolvedores decidir que atividades merecem mais tempo.

A metodologia também garante flexibilidade no tratamento das mudanças de requisitos típicas das Aplicações Web, permitindo a inclusão ou alteração de funcionalidades durante o desenvolvimento, de acordo com as necessidades do negócio ou as preferências do cliente.

5.1.3 FLUXO DE ATIVIDADES

A Figura 5.1 (*Workflow* para o desenvolvimento global de Aplicações Web) apresenta a seqüência de atividades a serem realizadas durante o Desenvolvimento Global de Aplicações Web. Para a definição das fases, foram consideradas as práticas tradicionais da Engenharia de Software, e as particularidades dos projetos globais e dos projetos Web.

O processo se inicia com a definição do contexto de desenvolvimento, onde são decididas as questões de negócios, e prossegue com a análise do problema a ser resolvido.

Definidos os requisitos iniciais da aplicação, o próximo passo é a tomada de decisões de projeto, como a alocação de recursos, o escalonamento de tarefas e o planejamento das entregas de resultados ao Cliente.

Para a produção do software, as equipes são divididas de acordo com suas especialidades e passam a trabalhar em paralelo na concepção dos três grandes elementos que compõem uma Aplicação Web: código, conteúdo e interface.

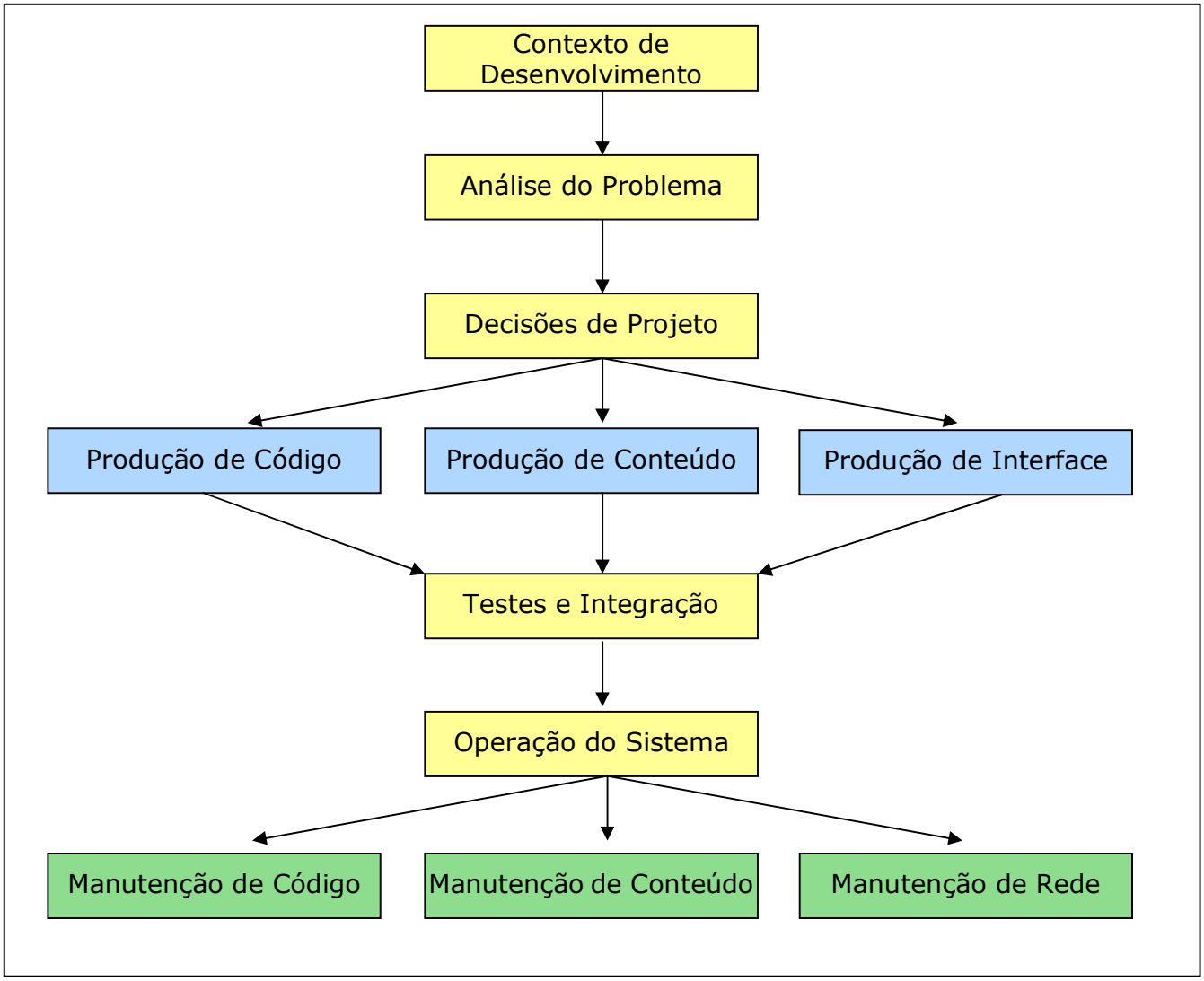

Figura 5.1 – *Workflow* para o Desenvolvimento Global de Aplicações Web

Depois de prontos, esses elementos são testados e integrados, constituindo um protótipo que será avaliado pelo Cliente. Finalmente, a primeira versão do sistema é posta em operação, e, segue-se para a fase de manutenção. A definição detalhada de cada etapa encontra-se no item 5.1.5 (Descrição das atividades).

É importante destacar que todas essas atividades são desempenhadas num curto espaço de tempo, estimado em 3 ou 4 meses para o ciclo inicial. Devido à flexibilidade do processo, os desenvolvedores podem preferir seguir apenas um subconjunto dessas atividades, ou dar mais ênfase a algumas fases em vez de outras.

Também é válido ressaltar que, de acordo com as necessidades do projeto, o processo pode ser repetido ciclicamente a partir de qualquer fase. É o que ocorre

principalmente na manutenção do sistema, etapa crítica do ciclo de vida das Aplicações Web, que pode ser vista como o início de um novo ciclo de desenvolvimento.

Uma vez em operação, o sistema permanece na fase de manutenção indefinidamente, até que saia de atividade. Václav Rajlich e Keith Bennett [RAJLICH, 2000] descrevem quatro categorias de manutenção:

- Aperfeiçoamento: ocorre quando são necessárias mudanças nas funcionalidades do sistema.

- Adaptativa: necessária quando ocorrem mudanças no ambiente, como atualizações de hardware e software, ou adoção de novas tecnologias.

- Corretiva: ocorre quando é preciso corrigir erros do sistema.

- Preventiva: realizada com o objetivo de melhorar a estrutura do sistema para evitar problemas futuros.

Assim, sempre que há necessidade de mudanças, os desenvolvedores analisam que tipo de manutenção será aplicada e definem que atividades serão realizadas. Por exemplo, se o objetivo for mudar as cores da interface da aplicação, não será necessário produzir nenhum código, tampouco modificar o conteúdo. Já no caso de uma manutenção de aperfeiçoamento, que pode ter impacto sobre a arquitetura do sistema, é provável que a interface e o conteúdo também precisem ser modificados para acomodar o novo código.

Esse caráter cíclico do processo garante uma maior flexibilidade para os desenvolvedores e uma maior agilidade para a evolução do software.

Outro aspecto relevante do processo é a concepção da interface como um elemento independente da lógica da aplicação, conforme propõe o Princípio da Independência do Diálogo [DRAPER, 1985].

Esse isolamento entre interface e software é de particular importância na produção de aplicações para mercados globais, uma vez que várias interfaces adaptadas culturalmente ao perfil dos usuários podem ser criadas por diferentes times trabalhando em paralelo sem que o código seja afetado, contribuindo, assim, para o reuso e para a redução do *time-to-market* dos projetos.

Já no caso das Aplicações Web, em que a interface é, sem dúvidas, o aspecto mais importante, e, por isso, requer aptidões especializadas dos desenvolvedores, a principal vantagem dessa prática é permitir que, além dos programadores, especialistas em design e artes gráficas possam integrar a equipe de desenvolvimento, contribuindo para a melhoria da qualidade das aplicações.

5.1.4 PAPÉIS

Para realizar as atividades do Desenvolvimento Global de Aplicações Web, os desenvolvedores devem se organizar de modo a desempenharem oito diferentes papéis, apresentados na Figura 5.2 (Papéis no Desenvolvimento Global de Aplicações Web).

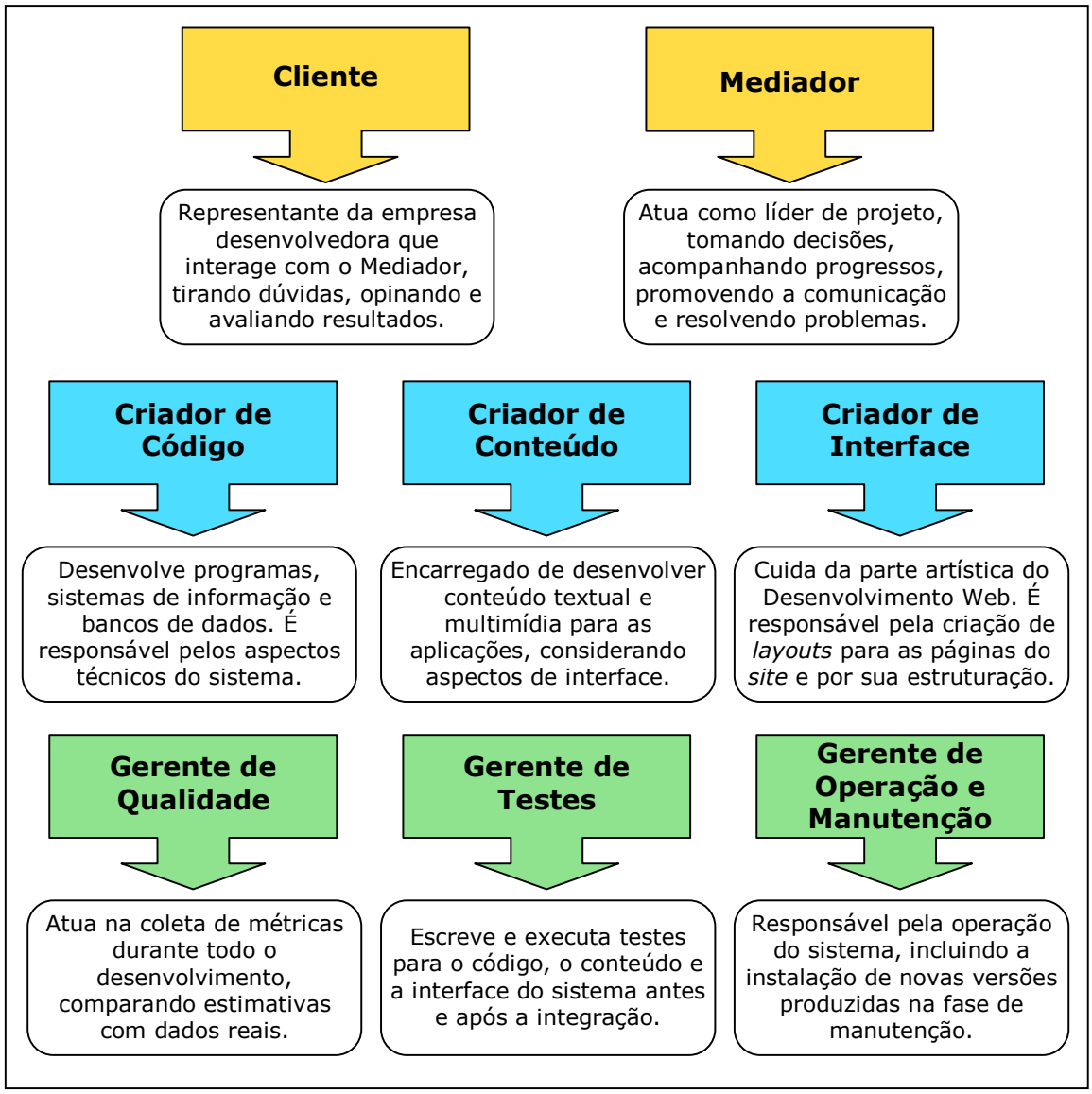

Figura 5.2 – Papéis no Desenvolvimento Global de Aplicações Web

A presença do Cliente[2] durante a produção do software é muito importante, pois, além de permitir que os desenvolvedores esclareçam suas dúvidas à medida que forem surgindo, evitando especulações e desentendimentos, garante um contínuo feedback sobre os resultados gerados, motivando a equipe na busca da qualidade.

Por isso, as empresas envolvidas no projeto devem escolher um ou mais representantes para atuarem como Clientes para as equipes, seja através do contato face-a-face, ou à distância, usando tecnologias de comunicação, como e-mail e vídeo-conferência.

Conforme descrito no Capítulo 3 (Desenvolvimento Global de Software), o Mediador é uma pessoa que atua como líder da equipe de desenvolvedores, motivando os profissionais, promovendo a comunicação entre eles, resolvendo problemas, acompanhando o cumprimento de metas e prazos, e avaliando os riscos do projeto.

Mais que isso, ele representa sua equipe nas decisões de projeto e interage diretamente com o Cliente, obtendo esclarecimentos e feedback. Assim, o Mediador funciona como uma ponte entre Clientes e desenvolvedores.

A produção da aplicação é feita por profissionais com perfis distintos. A criação de código, ou programação, é feita por profissionais de Tecnologia da Informação e de Computação, que cuidam de todos os aspectos técnicos do sistema, desde a criação de scripts e bancos de dados, até a integração da aplicação com sistemas legados.

Já a criação de conteúdo pode exigir profissionais especializados, como jornalistas, pessoal de marketing ou relações públicas, dependendo do tipo de aplicação. Esses profissionais devem produzir conteúdo textual e multimídia para o *site*, levando em consideração aspectos de interface.

A criação de *layouts* e estruturas de navegação entre as páginas do *site* é feita por designers e especialistas em computação. Conhecimentos sobre usabilidade ajudam esses profissionais na escolha de cores e ícones adequados ao público-alvo da aplicação.

Durante a produção, o Gerente de Qualidade atua na coleta de métricas do projeto e, juntamente com o Mediador, analisa o progresso da equipe, avaliando riscos de atrasos nos cronogramas e o cumprimento de metas previamente estabelecidas.

O papel do Gerente de Testes durante o desenvolvimento é escrever casos de teste que possam ser automatizados, e cenários para os testes manuais. Na integração do código com o conteúdo e a interface, é ele que realiza os testes para garantir o funcionamento adequado do sistema.

[2] É válido destacar a diferença entre os termos *cliente* e *Cliente* usados ao longo do texto. A palavra *cliente* é usada de modo genérico para representar as empresas e os usuários que demandam a aplicação. Já a expressão *Cliente* caracteriza os indivíduos que desempenham o papel de tomadores de decisão no projeto, e que atuam diretamente junto aos mediadores na elaboração dos requisitos e na avaliação dos resultados.

Uma vez que o sistema já foi desenvolvido e testado, o Gerente de Operação e Manutenção é responsável pela instalação da aplicação nos servidores e pelo monitoramento de seu funcionamento, para detectar e corrigir falhas técnicas. Na fase de manutenção, quando são geradas novas versões para o software, ele é o responsável pela atualização nas máquinas.

Sendo os times de desenvolvimento Web formados por um número pequeno de profissionais, é certo que um mesmo membro da equipe poderá acumular mais de um papel. Por exemplo, o conteúdo e a interface podem ser produzidos pela mesma pessoa, os próprios desenvolvedores podem escrever os testes para o sistema, ou ainda, o Mediador pode cuidar também dos testes e das métricas do projeto.

Outra alternativa é a terceirização de atividades que não exijam especialistas em desenvolvimento, como a operação do sistema ou sua manutenção. Essas decisões serão tomadas de acordo com as necessidades e o contexto de negócios do projeto.

No caso de projetos envolvendo a colaboração de vários times de desenvolvimento, esses papéis devem ser desempenhados em cada equipe, pois o objetivo é tornar o trabalho deles o mais independente possível.

Entretanto, é recomendável que os Clientes sejam sempre os mesmos para todas as equipes, para evitar confrontos de opiniões e desentendimentos em relação aos requisitos a serem seguidos. Apenas no caso da produção de várias versões de uma mesma aplicação para diferentes mercados, as opiniões dos Clientes podem diferir para representar as características culturais de cada público-alvo.

A comunicação entre os times concentra-se nos Mediadores, que passam a ser não só o elo com o Cliente, mas também com as outras equipes. Se os times usarem idiomas diferentes e possuírem valores distintos, outra atribuição dos Mediadores escolhidos será resolver problemas culturais de comunicação.

5.1.5 DESCRIÇÃO DAS ATIVIDADES

Agora que o fluxo de atividades e os papéis da metodologia já são conhecidos, serão detalhadas as atividades descritas na Figura 5.1 (*Workflow* para o desenvolvimento global de Aplicações Web). Para cada atividade, serão apontados os objetivos e a melhor forma de atingí-los.

5.1.5.1 CONTEXTO DE DESENVOLVIMENTO

Logo no início do projeto, é preciso caracterizar o cenário onde ocorrerá o desenvolvimento. A formação de parcerias, a organização dos times, e a intenção de atingir diferentes mercados são questões que precisam ser consideradas pelas empresas antes de começar o desenvolvimento, para que os riscos sejam antecipados e as decisões corretas sejam tomadas.

A realização de reuniões presenciais com representantes das empresas envolvidas é fundamental para que a negociação seja efetiva e todas as dúvidas sejam esclarecidas.

Algumas perguntas importantes que devem ser respondidas nessa fase estão listadas a seguir.

- **Sobre a formação de parcerias**

  - Uma parceria com outras empresas é necessária para o projeto?
  - Que vantagens essa parceria trará?
  - Que tipo de arranjo de negócios será feito?
  - Como serão respeitados os interesses de cada empresa em relação à propriedade intelectual dos resultados, ao cumprimento de prazos, e à participação nos lucros do projeto?

- **Sobre o modelo de desenvolvimento**

  - Onde estão localizadas as competências necessárias?
  - O desenvolvimento precisará ser distribuído? Em que etapas? Alguma fase será terceirizada?
  - Quantos centros de desenvolvimento estarão envolvidos?
  - O desenvolvimento ocorrerá em países diferentes?
  - Os participantes possuem afinidades culturais?

- **Sobre os mercados-alvo**

  - Que mercados o software deverá atender?
  - Qual a importância de cada mercado para o negócio?
  - O lançamento do produto nesses mercados ocorrerá simultaneamente?
  - Esses mercados apresentam semelhanças culturais?

- **Sobre as metas do projeto**

  - Quais os objetivos do negócio?
  - Qual será o escopo da aplicação?
  - Qual o prazo estimado?
  - Qual o retorno esperado?

As respostas para essas e outras perguntas devem ser dadas de comum acordo pelas empresas envolvidas, e devem ser documentadas em um contrato formal, denominado *Statement Of Work (SOW)*, para que sejam evitados contratempos durante o projeto. O Anexo B (Modelos de artefatos da metodologia *WideWorkWeb*) apresenta um modelo para a elaboração do *SOW*.

O contrato deve instituir políticas para a resolução dos problemas mais comuns em projetos desse tipo, e deve respeitar aspectos legais.

5.1.5.2 ANÁLISE DO PROBLEMA

Essa é uma das fases decisivas do projeto, pois é quando são definidos os requisitos do sistema. Além da área de domínio da aplicação, é importante conhecer o perfil do público-alvo que se deseja atingir.

É natural que os Clientes tenham dificuldades para definir o que esperam de suas aplicações, por isso uma avaliação de *sites* semelhantes já existentes no mercado pode ajudá-los a fazer escolhas e determinar quais serão seus diferenciais. Uma ferramenta específica para este propósito foi criada pelo Gartner Group, e pode ser obtida gratuitamente na Internet (http://www.techrepublic.com).

Nessa fase, são escolhidos os Mediadores que representarão cada time do projeto, e são realizadas reuniões presenciais com os Clientes para acertar detalhes sobre as características que deverão estar presentes na aplicação.

Num primeiro momento, os Clientes são auxiliados pelos Mediadores na produção de *user stories* [BECK, 2000] que representem as funcionalidades requeridas para o sistema. A prioridade de cada *user story* para o negócio é determinada pelos próprios Clientes.

Além dos requisitos funcionais, os Clientes também são estimulados a analisar que aspectos não-funcionais são críticos para o sistema. Isso pode ser feito através da definição de critérios de aceitação para aspectos como tempo de resposta para a solicitação de uma página, ou configuração mínima de hardware para uso da aplicação.

É claro que a importância dos requisitos não-funcionais será variável, de acordo com o tipo de aplicação a ser desenvolvido. No caso específico das aplicações de *e-commerce*, a segurança, a usabilidade e a acessibilidade merecem destaque.

Ainda nessa fase, os Clientes ajudam a definir a aparência das páginas e a estrutura do *site*. Em vez de usar protótipos, os desenvolvedores podem simplificar essa atividade apresentando esboços em papel com as cores e formas sugeridas. Além de reduzir os esforços, essa abordagem dá aos Clientes mais liberdade para opinar e propor mudanças, pois nada foi produzido ainda.

Colhidas as opiniões dos Clientes, a tarefa dos Mediadores é produzir dois importantes artefatos: o Modelo Arquitetural e o Modelo de Interface.

Tomando como base as *user stories* definidas, serão escolhidas as tecnologias necessárias na implementação da solução, e será planejada a arquitetura da aplicação. Essas decisões devem considerar questões importantes como reuso, manutenibilidade e portabilidade, para garantir a redução de esforços e custos durante o projeto.

Como já destacado nos Capítulos 2 (Desenvolvimento de Aplicações Web) e 3 (Desenvolvimento Global de Software), o Desenvolvimento Baseado em Componentes é a abordagem mais indicada, tanto para facilitar a produção de várias versões de uma mesma

aplicação, como para tornar o trabalho dos times mais independente. Por isso, a presente metodologia recomenda a adoção dessa prática durante o desenvolvimento.

Sendo assim, o Modelo Arquitetural deverá conter uma breve descrição sobre os componentes necessários, e informações sobre a comunicação entre eles, como parâmetros de acesso. A decisão sobre quais componentes serão produzidos e quais serão reusados também ocorre nessa fase.

O Modelo de Interface é composto pelos esboços para o *layout* das páginas aprovados pelo Cliente, e por um mapa de navegação que determina a hierarquia das páginas no *site*. Contém ainda decisões sobre o método de geração de conteúdo das páginas do *site*. A escolha entre páginas estáticas ou dinâmicas dependerá da freqüência de atualização dos dados e da performance exigida para o sistema.

Por exemplo, no caso dos sistemas de *e-commerce*, as páginas com informações institucionais da empresa dificilmente precisarão ser modificadas, podendo estar armazenadas em documentos estáticos, que têm um tempo de acesso menor. Já nas páginas que apresentam o catálogo de produtos da empresa, as atualizações são constantes, com inclusão de novos itens e mudanças nos preços dos produtos. Caso essas informações estivessem armazenadas em documentos estáticos, a exclusão dos dados de um único produto implicaria na modificação de todos os arquivos que fizessem referência àquele item.

Esse procedimento seria simplificado se as informações sobre os produtos estivessem armazenadas em uma fonte de dados única, como um banco de dados, e fossem recuperadas sob demanda. Esse é o princípio das páginas dinâmicas, que facilitam a manutenção dos sistemas, mas demoram mais que as estáticas para serem geradas, pois precisam ser processadas pelo servidor.

Finalmente, os Mediadores devem definir as métricas que serão utilizadas para acompanhamento do projeto, e os testes necessários para garantir a qualidade da aplicação. Essas escolhas serão feitas a partir de listas de métricas e testes possíveis, como as sugeridas no item 5.1.8 (Métricas para Suporte Executivo) e no Capítulo 6 (Testes e Ferramentas para o Desenvolvimento Web), respectivamente.

5.1.5.3 DECISÕES DE PROJETO

Uma vez que a arquitetura inicial da aplicação e sua interface já estão estabelecidas, o próximo passo é decidir como o trabalho será alocado entre os times. A escolha da estratégia de escalonamento dependerá, principalmente, do arranjo de negócios feito entre as empresas e da localização das competências necessárias.

Entretanto, como já foi dito no Capítulo 3 (Desenvolvimento Global de Software), o ideal é dividir as atividades de modo que o trabalho dos times seja o mais independente possível, e o critério mais comumente usado é a distribuição por funcionalidades do software.

Por isso, a proposta aqui apresentada é alocar um conjunto de *user stories* para cada equipe de desenvolvimento. Assim, cada time poderá se concentrar na produção de uma determinada coleção de componentes e da interface necessária para a realização das funcionalidades.

Durante a escolha das *user stories* que serão implementadas por cada equipe é importante considerar a dependência entre elas e a seqüência em que os componentes serão produzidos para evitar redundâncias e atrasos ao longo do projeto.

Já dentro de cada time, a alocação ocorre de acordo com as competências dos desenvolvedores. O código, o conteúdo e a interface são produzidos por profissionais com perfis diferentes, conforme descrito no item 5.1.4(Papéis).

Escolhida a estratégia de alocação das equipes, é possível determinar quantos ciclos de produção-testes-integração serão necessários para a entrega da aplicação ao Cliente. A definição do escopo de cada ciclo é feita de acordo com as prioridades estabelecidas pelos Clientes para as *user stories*, uma vez que os interesses de negócios devem ser respeitados. A duração de cada ciclo é, então, estimada, e são analisados os riscos envolvidos.

É importante esclarecer que o planejamento dos ciclos não é fixo, mas adaptável à realidade do projeto. Se durante o desenvolvimento houver uma mudança nos requisitos do cliente, e o acréscimo de novas *user stories* for necessário, a definição dos ciclos terá que ser refeita.

Da mesma forma, se durante os testes de aceitação com o cliente, realizados após cada ciclo, a aplicação apresentar falhas ou não estiver de acordo com o esperado, *user stories* corretivas deverão ser adicionadas ao planejamento.

O último passo é definir e disponibilizar os recursos necessários em cada centro de desenvolvimento. A escolha das ferramentas de desenvolvimento e das tecnologias de comunicação deve favorecer a integração dos resultados produzidos pelos times e a comunicação entre os desenvolvedores e com os Clientes. O Capítulo 6 (Testes e ferramentas para o desenvolvimento Web) apresenta sugestões para as ferramentas que costumam ser necessárias em projetos desse tipo.

5.1.5.4 PRODUÇÃO DE CÓDIGO

Quando um time recebe a *user story* que deverá implementar, começa o detalhamento das atividades que a compõem. São listados os passos necessários para que as funcionalidades corretas sejam criadas para o sistema, e são estimados prazos para cada atividade.

É desejável que a alocação das atividades dentro do time ocorra espontaneamente, com os programadores escolhendo suas próprias atribuições, para que a motivação para o trabalho seja maior.

Antes do início da codificação, é importante que os testes de unidade sejam escritos para ajudar a esclarecer dúvidas dos desenvolvedores e promover a automatização da fase de testes, uma vez que esse tipo de teste será realizado ao longo de todo o desenvolvimento.

Já durante a codificação, o uso de técnicas de desenvolvimento como as propostas por *eXtreme Programming* [BECK, 2000] (integração contínua, uso de padrões de codificação, refatoramento, projeto simples, e propriedade coletiva) é bastante recomendável.

Outras práticas sugeridas são a boa documentação do código, e o uso de um repositório único de artefatos, que permita o registro de *bugs* e a notificação de mudanças, em cada centro de desenvolvimento. O objetivo é facilitar as modificações necessárias e evitar a inconsistência das versões do software.

Também é importante que os desenvolvedores usem técnicas de Internacionalização desde o começo da codificação, facilitando, assim, a produção de versões para diferentes mercados. Mesmo que esta não seja uma meta inicial do projeto, os times devem tentar isolar os componentes que apresentam mensagens de erro aos usuários, ou utilizam padrões de dados pré-definidos, para que esses módulos possam ser adaptados prontamente se necessário.

O objetivo é construir o código de modo que o idioma e o formato dos dados possam ser definidos dinamicamente, de acordo com a versão do software. Algumas sugestões de práticas e bibliotecas de código que facilitam a Internacionalização são citadas num guia produzido pelo consórcio X/Open [X/OPEN, 1994].

O controle da qualidade do código pode ser feito através de práticas simples, como revisões e inspeções. Apesar de eficiente em alguns casos, a programação em pares pode não ser uma alternativa viável, dado o reduzido tamanho das equipes de desenvolvimento.

Paralelamente à codificação, os casos de teste devem ser produzidos, seja pelo Gerente de Testes da equipe ou pelos próprios desenvolvedores. As funcionalidades, a portabilidade, a performance e a segurança do sistema são os principais aspectos a serem avaliados. Se a arquitetura da aplicação incluir a comunicação com sistemas legados, essa integração também precisará ser testada.

5.1.5.5 PRODUÇÃO DE CONTEÚDO

O primeiro passo dos Criadores de Conteúdo é definir, de acordo com os *layouts* e o mapa de navegação do Modelo de Interface, que tipo de conteúdo será necessário em cada página do *site*. Ou seja, é feita uma lista de todas as mídias (textos, imagens, sons, animações, etc) que precisam ser criadas.

A partir daí, começa o trabalho de criação dos profissionais, que devem considerar, além da estrutura das páginas, a heterogeneidade dos mercados-alvo. Enquanto os aspectos culturais influenciam a escolha de ícones e temas para as imagens, a existência de vários

browsers e de conexões de rede variáveis determinam o tamanho e o formato das mídias, que não devem comprometer a performance ou a portabilidade do sistema.

Além de criar mídias para a aplicação, os Criadores de Conteúdo também são responsáveis pela produção da documentação do sistema para o usuário. No caso das Aplicações Web, poucos artefatos são necessários, visto que não há procedimentos de instalação e o uso do software é facilitado por uma interface com boa usabilidade. Um bom mapa do *site*, com indicações sobre a estrutura hierárquica das páginas, e uma lista de perguntas freqüentes (*Frequently Asked Questions - FAQ*), com suas respectivas respostas, podem ser suficientes para auxiliar o usuário.

As revisões de texto são realizadas para evitar erros de sintaxe e gramática. Sempre que possível, os usuários dos diferentes mercados devem ser consultados para assegurar que as traduções e adaptações dos termos usados na aplicação estão corretas.

5.1.5.6 PRODUÇÃO DE INTERFACE

A tarefa dos desenvolvedores nessa etapa é criar as páginas Web necessárias na montagem do *site*. Para isso, são utilizadas as definições do Modelo de Interface, como a escolha das cores e a disposição dos elementos na tela.

Uma vez que a criação das páginas envolve a integração com o conteúdo, num primeiro momento espera-se que os desenvolvedores criem apenas modelos de *layout* (*templates*) para os documentos, deixando lacunas para as mídias que serão inseridas posteriormente.

Além de facilitar a futura manutenção do *site*, permitindo a rápida criação de novas páginas, o uso de *templates* ajuda a garantir a consistência das cores e da localização de menus entre as páginas, tornando a navegação mais simples para o usuário.

Assim como ocorre com o conteúdo, é desejável que a interface da aplicação seja culturalmente adaptada para os diferentes mercados-alvo, pois, como apontam Jakob Nielsen e Donald Norman [NIELSEN, 2000], para os sistemas Web, a usabilidade não é um luxo, mas uma questão de sobrevivência.

Além disso, a escolha dos elementos da interface deve considerar a existência de vários *browsers*, cada um com várias versões disponíveis no mercado. O uso de recursos como *plug-ins* e *frames* pode comprometer o acesso de alguns usuários à aplicação, o que não é desejável, principalmente em sistemas de *e-commerce*. Uma alternativa é apresentar diferentes opções de interface e permitir que o usuário escolha, na hora do acesso, a mais adequada à configuração de sua máquina.

Prontos os *templates* das páginas Web, o próximo passo é montar a estrutura do *site*. Para isso, os desenvolvedores tomam como base o mapa de navegação definido no Modelo de Interface, que deve conter detalhes sobre a hierarquia entre as páginas, e indicar a localização dos *links*.

Após a montagem do *site*, o conteúdo é integrado à interface e um protótipo da aplicação, sem funcionalidades, é gerado para a realização de testes. Os principais aspectos que precisam ser testados nessa etapa são a sintaxe dos documentos HTML, a aparência do *site* em diferentes *browsers*, o funcionamento correto de todos os *links*, a consistência de *layout* entre as páginas, o tempo necessário para o carregamento da página com diferentes taxas de conexão, e a disposição do conteúdo na interface (dimensões dos textos e imagens).

Os procedimentos necessários para verificar a adequação de todas essas características aos critérios de aceitação definidos para o software devem estar descritos em casos de teste automatizados e cenários para os testes manuais. Exemplos de casos de teste e cenários podem ser encontrados no Capítulo 6 (Testes e ferramentas para o desenvolvimento Web).

É importante destacar que durante a produção da interface, ou mesmo do conteúdo, os resultados parciais podem ser submetidos à avaliação do Cliente. Essa apresentação é feita pelo Mediador da equipe, que colhe opiniões e sugestões de mudanças e repassa as informações para a equipe.

5.1.5.7 TESTES E INTEGRAÇÃO

É nesta etapa que o código, o conteúdo e a interface, já testados isoladamente, serão integrados para compor o protótipo da *user story*, que será submetido à aceitação do Cliente.

Integrar esses elementos significa concentrar, num único servidor, todos os arquivos gerados, e realizar alguns testes para garantir que o funcionamento do sistema ocorre da maneira esperada. Esses testes, que podem ser automáticos, feitos com a ajuda de ferramentas, ou manuais, de acordo com cenários de testes definidos previamente, encontram-se descritos detalhadamente no Capítulo 6 (Testes e ferramentas para o desenvolvimento Web).

Em projetos com vários times distribuídos, após a integração e os testes de cada *user story* separadamente, os Gerentes de Teste e os Mediadores precisam ainda integrar as *user stories* produzidas e testar o funcionamento do sistema completo.

Para isso, o procedimento é idêntico ao realizado em cada time. Entretanto, como os arquivos estão localizados em máquinas distribuídas, os desenvolvedores podem escolher uma máquina única, onde todos deverão integrar seus resultados, e onde serão realizados os testes finais. É importante que a integração ocorra gradativamente, para que as falhas sejam detectadas mais facilmente.

Durante a integração e os testes das *user stories*, é possível que sejam detectados alguns *bugs*. Se as falhas forem poucas e puderem ser rapidamente reparadas, a correção poderá ser feita imediatamente pelos desenvolvedores. Caso contrário, *user stories* corretivas deverão ser adicionadas ao projeto, e o planejamento dos ciclos terá que ser refeito.

A versão parcial da aplicação, composta pela integração das *user stories*, é então submetida à avaliação do Cliente, tendo como base os critérios previamente definidos. As opiniões e sugestões do Cliente são colhidas pelos Mediadores, e também podem originar *user stories* corretivas.

Se ainda houver *user stories* a serem desenvolvidas, os times iniciam um novo ciclo de produção-testes-integração. Caso contrário, a aplicação inicial é dada como finalizada, e têm início os procedimentos para a operação do sistema.

Entretanto, antes de ser disponibilizada, é desejável que a aplicação seja submetida à avaliação de amostras de usuários dos mercados-alvo. Os testes beta podem ser feitos sem custos adicionais, com os usuários acessando o sistema remotamente através da Web e enviando suas sugestões e comentários.

Um documento denominado *Post-Mortem*, contendo as opiniões dos usuários e do Cliente e o relato dos problemas encontrados na integração dos sistemas, deve ser produzido pelos desenvolvedores no final de cada ciclo para servir de referência para melhoramentos do sistema, e de experiência para projetos posteriores. O Anexo B (Modelos de artefatos da metodologia *WideWorkWeb*) apresenta um modelo para a produção do Post-Mortem.

5.1.5.8 OPERAÇÃO DO SISTEMA

O papel do Gerente de Operação e Manutenção é resolver as questões técnicas e preparar o ambiente onde será disponibilizada a aplicação. Instalar e configurar os programas necessários nos servidores, verificar o funcionamento dos equipamentos e da rede, e testar o acesso à aplicação são algumas de suas atribuições. Quando necessária, a integração com o legado também ocorre nessa etapa.

É recomendável que esse papel seja inicialmente desempenhado por um dos membros da equipe de Produção de Código, visto que a instalação e configuração da aplicação pressupõem um bom conhecimento sobre o funcionamento do sistema. De qualquer forma, todos os procedimentos devem ser documentados, na forma de scripts de instalação e *checklists* de atividades, para que a manutenção do sistema seja facilitada.

No caso de projetos envolvendo equipes distribuídas e versões para diferentes mercados, é preciso considerar ainda se a operação do sistema ocorrerá também de maneira distribuída. Como essa decisão depende principalmente dos interesses de negócios, o *Statement Of Work* deverá indicar se um único centro será escolhido para concentrar todas as versões da aplicação, ou se cada centro será responsável pela operação da versão do software correspondente a seu mercado.

A partir do momento em que o sistema é disponibilizado para os usuários, começam as campanhas de divulgação do *site*, seja através do envio de e-mails para os usuários potenciais, ou com o cadastro nas principais ferramentas de busca.

Em aplicações de *e-commerce*, mais que um suporte técnico para garantir o bom funcionamento do sistema, é necessário também que uma equipe de profissionais bem treinados esteja disponível para tirar as dúvidas dos usuários, e coletar suas opiniões e sugestões através de formulários on-line e e-mails.

5.1.5.9 MANUTENÇÃO DE CÓDIGO

Conforme discutido no Capítulo 2 (Desenvolvimento de Aplicações Web), a manutenção é a fase de maior destaque no Desenvolvimento de Aplicações Web, devido à rápida e constante mudança de requisitos do mercado.

Durante a operação do sistema, tão importante quanto acompanhar a evolução das aplicações concorrentes e da tecnologia, é observar os hábitos de acesso dos usuários. C*ookies* e ferramentas de análise de *logs* podem ajudar as empresas a identificar que aspectos da aplicação merecem maior ênfase ou precisam de melhorias.

Sempre que uma mudança no código da aplicação é necessária, seja para inclusão de funcionalidades (manutenção de aperfeiçoamento), seja para a migração para uma nova tecnologia ou plataforma (manutenção adaptativa), inicia-se um novo ciclo de desenvolvimento, que pode durar de duas a quatro semanas.

A manutenção para correção de erros também pode ocorrer no código, mas a realização de testes durante a fase de produção e o uso de *user stories* corretivas minimiza essa necessidade.

A decisão sobre que atividades do desenvolvimento precisarão ser realizadas nesse novo ciclo dependerá do tipo de mudança necessária e do impacto dessa mudança na estrutura da aplicação.

A Manutenção de Código costuma ser mais complexa que a Manutenção de Conteúdo, pois além de provocar alterações na arquitetura do sistema, a inclusão de funcionalidades implica na criação de novas páginas Web. Ou seja, além de código, os desenvolvedores precisam produzir conteúdo e interface para o sistema.

Embora não seja responsável pelos melhoramentos na aplicação, é o Gerente de Operação e Manutenção que disponibiliza as novas versões do sistema para os usuários. É importante observar que fazer a migração de uma versão da aplicação para outra quando o sistema já está em operação é uma tarefa arriscada, e que exige bastante planejamento.

É preciso antecipar todas as ações e escolher bem o momento de realizar as mudanças para minimizar o tempo em que o sistema ficará indisponível, e a quantidade de usuários atingidos. O uso de *checklists* de atividades indicando os procedimentos necessários para a configuração e operação do sistema (ex: que arquivos devem ser copiados, que scripts precisam ser executados e em que ordem, quais são os parâmetros de configuração da máquina e do *browser*, etc.) é bastante recomendável para guiar os desenvolvedores e agilizar as alterações.

Além disso, deve existir um plano de contingência, caso a migração não seja realizada com sucesso no tempo esperado, para que as mudanças sejam desfeitas e a versão antiga do sistema volte a operar. Glenn Stout [STOUT, 2001b] apresenta várias outras dicas para a instalação de sistemas Web, e o Anexo B (Modelos de artefatos da metodologia *WideWorkWeb*) apresenta um modelo para a elaboração de um Plano de *Deployment*.

Algumas ferramentas para controle de versões disponíveis no mercado podem ajudar os desenvolvedores na hora de fazer a migração do sistema. O mais importante, entretanto, é que todos os testes sejam realizados previamente para evitar surpresas de última hora. É certo que qualquer modificação no código ou nos demais elementos da aplicação deverá ser refletida também nos testes, com a criação de novos casos de testes e ajustes nos já existentes.

Mais uma vez, é importante que o Gerente de Operação e Manutenção conheça bem o sistema para saber que procedimentos devem ser seguidos. Se a manutenção do sistema não for feita pelo time de desenvolvimento, a ajuda de alguns membros da equipe será necessária durante a migração entre versões.

5.1.5.10 MANUTENÇÃO DE CONTEÚDO

É muito comum que as Aplicações Web precisem ter seu conteúdo atualizado constantemente, já que o objetivo é atrair um número cada vez maior de usuários. Esse aspecto é particularmente crítico em Aplicações Informativas e de *Workflow*, que precisam acompanhar a velocidade com que as informações mudam.

Por isso, é importante que a Manutenção de Conteúdo possa ser realizada num curto intervalo de tempo. Enquanto a Manutenção de Código propõe ciclos de duas a quatro semanas, os ciclos para mudança de conteúdo podem se resumir a alguns minutos ou horas.

Isso só é possível porque, geralmente, o código da aplicação não precisa sofrer nenhuma alteração. Apenas as mídias e as páginas Web devem ser atualizadas e testadas.

Além de agilizar a criação de novas páginas Web, os *templates* da interface e o mapa de navegação, criados na fase de desenvolvimento do sistema, permitem que o próprio Criador de Conteúdo monte e adicione a nova página ao *site*, minimizando os gastos com profissionais especializados.

5.1.5.11 MANUTENÇÃO DE REDE

Embora não seja ligada ao desenvolvimento propriamente dito da aplicação, a Manutenção de Rede é uma atividade fundamental para garantir o bom funcionamento do *site*.

A partir do momento em que o sistema passa a ser utilizado pelos usuários, é necessário que o Gerente de Operação e Manutenção verifique periodicamente os equipamentos e monitore os acessos ao *site* para que problemas de conexão e falhas técnicas sejam rapidamente detectados e resolvidos.

Como essa atividade não pressupõe experiência em programação ou habilidades de design gráfico, qualquer profissional com conhecimentos técnicos de informática e redes de computadores pode ser escolhido para a função.

5.1.6 ATIVIDADES GERENCIAIS

Para que uma metodologia de desenvolvimento de software seja eficaz, é preciso que ela proponha ações gerenciais que facilitem o acompanhamento dos projetos e a coordenação do trabalho dos desenvolvedores, de modo a garantir o cumprimento das metas do projeto nos cronogramas e orçamentos estabelecidos desde o início.

No caso do Desenvolvimento Global de Aplicações Web, as ações gerenciais merecem um destaque ainda maior que nos projetos tradicionais, pois a participação de times de desenvolvimento geograficamente dispersos e heterogêneos trabalhando sob forte pressão do *time-to-market* ocasiona riscos adicionais.

Ao longo das fases apresentadas no item 5.1.5 (Descrição das Atividades) já foram sugeridas várias medidas de apoio à gerência do projeto, como o uso de times independentes seguindo o Desenvolvimento Baseado em Componentes, a realização de ciclos de desenvolvimento rápidos com participação efetiva do Cliente, e o uso de tecnologias de comunicação para reduzir a distância entre os participantes.

Para que essas medidas sejam adotadas com sucesso, a participação dos Mediadores é fundamental. Afinal, são eles que coordenam as atividades dos times e representam o elo com o Cliente durante o desenvolvimento. Também é responsabilidade deles identificar problemas e coletar métricas para antecipar riscos, além de manter o espírito de equipe entre participantes com perfis diferenciados.

O acompanhamento de cada time é feito diariamente através de reuniões informais e rápidas, com duração de 10 a 15 minutos, em que os desenvolvedores relatam o progresso de suas atividades e as dificuldades encontradas. Além de fortalecer o espírito de equipe dos participantes, as reuniões ajudam os Mediadores a identificar problemas tão logo eles apareçam, e são uma boa oportunidade para a coleta de métricas. O Anexo B (Modelos de artefatos da metodologia *WideWorkWeb*) apresenta um modelo de formulário que poderá ser usado para a coleta de métricas.

Ao final da semana, cada Mediador prepara um relatório em que descreve resumidamente os avanços e contratempos de seu time, e envia por e-mail para os demais Mediadores. Assim, todos se mantêm informados sobre os acontecimentos e podem transmitir para seus times a confiança de que o projeto está progredindo. O Anexo B (Modelos de artefatos da metodologia *WideWorkWeb*) apresenta um modelo para a elaboração do relatório gerencial.

A realização de reuniões virtuais, ou até presenciais, entre os Mediadores pode ser necessária caso os relatórios indiquem problemas que tenham impacto sobre as diretrizes do

projeto. Também podem ocorrer reuniões entre os participantes das equipes, mas elas devem ser evitadas para reduzir gastos com viagens e uso de vídeo-conferência.

Finalmente, para ter uma visão mais ampla do trabalho das equipes, cada Mediador é responsável pela criação e manutenção de uma página Web contendo informações relevantes sobre seu time e suas atribuições. Dados sobre os participantes, cronogramas, metas, alocação de atividades e métricas de acompanhamento devem estar disponíveis para que qualquer pessoa envolvida no projeto possa ter acesso a essas informações.

A página deve servir ainda como repositório para todos os artefatos gerados pelo time, fornecendo uma visão real do desenvolvimento em cada instante. Essa transparência em relação às informações do projeto garante aos Mediadores uma segurança maior para a tomada de decisões, e serve de motivação para os times, que conseguem perceber a importância de suas contribuições para o sistema como um todo.

5.1.7 ARTEFATOS

Conforme discutido no Capítulo 2 (Desenvolvimento de Aplicações Web), a criação de Aplicações Web demanda uma metodologia de desenvolvimento mais leve e adaptativa que os processos tradicionais para permitir ciclos de desenvolvimento rápidos e com a participação de profissionais de diferentes áreas. Para isso, a produção excessiva de documentação deve ser evitada, minimizando a quantidade de artefatos a serem gerados e sua complexidade.

Entretanto, é preciso considerar que projetos envolvendo times distribuídos requerem uma preocupação maior em relação ao registro das atividades, uma vez que a comunicação informal entre os participantes é dificultada pela distância. Ou seja, o desempenho dos times, que antes era discutido em reuniões face-a-face, passa a ser acompanhado pelos Mediadores através de relatórios e métricas.

A Tabela 5.1 (Artefatos a serem produzidos no Desenvolvimento Global de Aplicações Web) apresenta uma lista completa dos artefatos necessários no Desenvolvimento Global de Aplicações Web, com uma breve descrição sobre seu conteúdo e a indicação da fase do ciclo de desenvolvimento em que ele deve ser produzido, e o Anexo B (Modelos de Artefatos da Metodologia *WideWorkWeb*) apresenta modelos para a produção de alguns desses artefatos. É importante observar que a maioria dos artefatos propostos consiste apenas de documentos de texto simples, como listas e relatórios, que exigem pouco tempo e esforço dos desenvolvedores e não requerem ferramentas especiais para edição.

Também é válido destacar que a relevância de cada artefato para o projeto dependerá principalmente do contexto de desenvolvimento definido. Por exemplo, se o objetivo é construir uma aplicação puramente informativa, a ênfase será a criação de conteúdo e interface; muito pouco ou nenhum código precisará ser produzido.

Já no caso do projeto envolver um único time ou equipes fisicamente próximas, os relatórios gerenciais e o *site* para acompanhamento do projeto podem ser substituídos pela presença do Mediador atuando diretamente com os desenvolvedores. Ou seja, a lista de artefatos apresentada poderá ser adaptada de acordo com a realidade de cada projeto, cabendo aos Mediadores determinar que documentos são críticos.

Artefato	Descrição	Quando é Produzido
Análise de Riscos	▪ Documento produzido pelos Mediadores que descreve os riscos potenciais envolvidos em cada etapa do desenvolvimento, bem como medidas alternativas para minimizar o impacto desses riscos no cronograma e no orçamento do projeto (vide Anexo B).	No início de cada fase do ciclo
Statement Of Work (SOW)	▪ Contrato formal que estabelece diretrizes para o projeto considerando os interesses de negócios das empresas envolvidas ▪ Descreve condutas relacionadas a quatro aspectos importantes: a formação de parcerias entre empresas, a participação de times de desenvolvimento distribuídos, a produção de software para diferentes mercados, e as metas do projeto em relação a prazos e custos (vide Anexo B).	Na fase *Contexto do Desenvolvimento*
User Stories	▪ Descrevem os requisitos funcionais para a aplicação; são definidas e priorizadas pelos Clientes, com a ajuda dos Mediadores ▪ Podem ser definidas pelos desenvolvedores para indicar melhoramentos a serem feitos (*user stories* corretivas)	Na fase *Análise do Problema*, e durante o desenvolvimento (*user stories* corretivas)
Critérios de Aceitação	▪ Lista de requisitos não-funcionais definidos pelos Clientes para a aplicação ▪ Exemplos: tempo de resposta aceitável para consulta, configuração mínima de hardware, velocidade de conexão, etc.	Na fase *Análise do Problema*

Tabela 5.1 – Artefatos a serem produzidos no Desenvolvimento Global de Aplicações Web

Artefato	Descrição	Quando é Produzido
Modelo Arquitetural	▪ Documento contendo uma breve descrição sobre os componentes que farão parte do sistema e informações sobre a comunicação entre eles (Ex: parâmetros de acesso); indica ainda quais componentes serão produzidos e quais serão reusados ▪ Indica as tecnologias que serão utilizadas na implementação, considerando o reuso, a manutenibilidade e a portabilidade da aplicação final	Na fase *Análise do Problema*
Modelo de Interface	▪ Composto por esboços para o *layout* das páginas e por um mapa de navegação que define a hierarquia do *site* ▪ Contém decisões sobre o método de geração das páginas do *site* (arquivos estáticos ou dinâmicos)	Na fase *Análise do Problema*
Métricas de Acompanhamento	▪ Lista de medidas escolhidas com o objetivo de coletar dados para estimativas em projetos futuros e para garantir o atendimento aos cronogramas e orçamentos do projeto (vide Anexo B) ▪ Exemplos: número de *user stories*, número de mídias produzidas, esforço estimado e esforço real	São definidas na fase *Análise do Problema* e coletadas durante todo o projeto
Plano de *Releases*	▪ Documento indicando o escopo e a duração estimada de cada ciclo de produção-testes-integração ▪ Define o escalonamento das atividades entre os times	Na fase *Decisões de Projeto*, podendo sofrer alterações ao longo do projeto

Tabela 5.1 - Artefatos a serem produzidos no Desenvolvimento Global de Aplicações Web (cont.)

Artefato	Descrição	Quando é Produzido
Relatório Gerencial	▪ Documento contendo informações sobre a coordenação do trabalho dos times que servirão como base para a tomada de decisões (atividades realizadas, problemas encontrados, riscos, etc) (vide Anexo B) ▪ Cada Mediador produz um relatório semanal sobre seu time e repassa-o para os demais Mediadores	A partir da fase *Decisões de Projeto*; as atualizações ocorrem durante todo o projeto
***Site* para Acompanhamento**	▪ Repositório com informações sobre as atividades dos times (metas, alocação de tarefas, etc) e os artefatos gerados; contém ainda os cronogramas a serem seguidos e as métricas coletadas ▪ Cada time possui seu próprio *site*, e os dados e artefatos são atualizados constantemente pelo Mediador e pelos desenvolvedores ▪ Permite que cada participante tenha uma visão geral do progresso de seu time e das demais equipes	A partir da fase *Decisões de Projeto*; as atualizações ocorrem durante todo o projeto
Detalhamento das *User Stories*	▪ Lista de passos necessários para a implementação de cada *user story*	Na fase *Produção de Código*
Código	▪ Conjunto de componentes necessários para o funcionamento do sistema ▪ Durante a codificação, os desenvolvedores devem adotar práticas de desenvolvimento, como as propostas por eXtreme Programming [BECK, 2000], e técnicas de Internacionalização de software, como as sugeridas pelo consórcio X/Open [X/OPEN, 1994]; também é importante incluir comentários explicativos dentro do código	Na fase *Produção de Código*

Tabela 5.1 - Artefatos a serem produzidos no Desenvolvimento Global de Aplicações Web (cont.)

Artefato	Descrição	Quando é Produzido
Lista de Mídias	▪ Relação dos elementos que precisam ser produzidos para compor o conteúdo do *site* ▪ Contém o nome, a descrição e o tipo (imagem, som, vídeo, etc) de cada mídia que será criada	Na fase *Produção de Conteúdo*
Conteúdo	▪ Conjunto de mídias que, juntamente com o *layout*, formam a identidade visual da aplicação ▪ Deve ser adaptado para a heterogeneidade dos usuários (diferentes culturas, diversos *browsers*, etc)	Na fase *Produção de Conteúdo*
Documentação do Sistema	▪ Produzida para auxiliar o usuário na utilização da aplicação ▪ No caso das Aplicações Web, pode ser resumida a uma lista com as perguntas e respostas mais freqüentes (FAQ) e um mapa do *site* indicando a hierarquia das páginas	Na fase *Produção de Conteúdo*
Interface	▪ Conjunto de modelos de *layout* (*templates*) que são integrados ao conteúdo para a criação das páginas Web da aplicação ▪ É criada segundo critérios definidos no Modelo de Interface, como cores e disposição dos elementos na tela, e de acordo com os aspectos culturais do público-alvo	Na fase *Produção de Interface*

Tabela 5.1 - Artefatos a serem produzidos no Desenvolvimento Global de Aplicações Web (cont.)

Artefato	Descrição	Quando é Produzido
Registro de Erros e Modificações	▪ As falhas encontradas durante o desenvolvimento da aplicação e as correções feitas devem ser notificadas para os participantes do projeto, uma vez que a propriedade dos artefatos é compartilhada por todos ▪ Essa é também uma forma de registrar as experiências dos participantes durante o projeto, seja na adoção de uma nova tecnologia ou no uso de uma ferramenta, para que essas informações sejam utilizadas em futuros projetos	Nas fases *Produção de Código*, *Produção de Conteúdo* e *Produção de Interface*
Casos de Teste e Cenários	▪ Devem ser criados para cada elemento produzido (código, conteúdo e interface) para identificar erros e garantir o atendimento aos critérios de aceitação previamente definidos ▪ Os casos de teste automatizados devem ser preferidos pelos desenvolvedores para minimizar os esforços; no caso dos testes manuais, os cenários indicam a lista de passos a serem seguidos para verificar a eficácia do sistema	Nas fases *Produção de Código*, *Produção de Conteúdo*, *Produção de Interface* e *Testes e Integração*
Plano de *Deployment*	▪ É composto por scripts de instalação e *checklists* de atividades que descrevem os procedimentos necessários para a configuração e operação do sistema (Ex: lista de arquivos a serem copiados, seqüência de realização dos testes, etc) ▪ Deve incluir ainda um plano de contingência (*rollback plan*), que indica as ações necessárias para desfazer as mudanças e voltar ao estado anterior do sistema, caso a instalação de uma nova versão falhe (vide Anexo B)	Na fase *Operação do Sistema*
Post-Mortem	▪ Relatório que contém as opiniões dos usuários e do Cliente sobre a aplicação produzida, indicando possíveis melhorias para versões futuras ▪ Aponta ainda os problemas encontrados durante o desenvolvimento e as soluções adotadas, para servir de experiência para projetos posteriores (vide Anexo B)	Ao fim de cada ciclo de produção-testes-integração

Tabela 5. 1 - Artefatos a serem produzidos no Desenvolvimento Global de Aplicações Web (cont.)

5.1.8 MÉTRICAS PARA SUPORTE EXECUTIVO

Além de serem usados como indicadores de progresso das atividades em um projeto de desenvolvimento de software, os dados coletados junto aos times podem servir como base para estimativas mais realistas do esforço necessário em projetos posteriores.
.

Por isso, os Mediadores devem escolher com precisão a coleção de métricas que serão observadas no desenvolvimento para que elas caracterizem bem o projeto e permitam o acompanhamento efetivo do trabalho das equipes, sem que esforços adicionais sejam necessários para seu monitoramento.

Nessa escolha, é importante considerar que as Aplicações Web possuem uma estrutura diferente das demais aplicações: os elementos código, conteúdo e interface aparecem combinados na composição das páginas Web.

Ou seja, as métricas comumente usadas em projetos tradicionais, como linhas de código e ponto por função [PRESSMAN, 1995], podem ser bastante úteis para estimar o esforço despendido na produção do código da aplicação, mas são pouco significativas em relação à criação de seu conteúdo e interface, principalmente quando são usadas páginas geradas dinamicamente. Além disso, as medidas convencionais não refletem a realidade do Desenvolvimento Baseado em Componentes recomendado na metodologia aqui apresentada.

Tomando como referência dois importantes estudos sobre a produção de Aplicações Web [MENDES, 2001][ZETTEL, 2001], a Figura 5.3 (Métricas para Acompanhamento dos Projetos) apresenta uma coleção de métricas que podem ser usadas pelos Mediadores e pelo Gerente de Qualidade para acompanhar o trabalho dos times distribuídos.

Além de serem quantificáveis e objetivas, podendo ser coletadas automaticamente com o auxílio de ferramentas, as métricas sugeridas consideram os diferentes artefatos gerados no processo, permitindo que os Mediadores identifiquem riscos e atuem diretamente junto aos participantes da equipe responsáveis pelo artefato.

As métricas relacionadas ao reuso dos elementos são particularmente significativas nos ciclos de manutenção da aplicação, pois indicam a qualidade da arquitetura projetada inicialmente.

A coleta desses dados deve ser feita durante todo o desenvolvimento, e os Mediadores são responsáveis pela freqüente atualização das métricas de seus times nos *Sites* de Acompanhamento.

Embora as métricas propostas sejam bons indicadores de progresso, elas só poderão servir como base para estimativas futuras se forem observadas dentro do contexto em que foi realizado o projeto, uma vez que a organização das equipes e o tipo de aplicação criada têm impacto direto sobre a complexidade e a duração do desenvolvimento.

Por isso, é importante que os Mediadores registrem, ao fim de cada ciclo de desenvolvimento, informações sobre os participantes envolvidos (quantidade, competências, atribuições no projeto, localização geográfica), sobre a Aplicação Web criada (categoria, modelo arquitetural, mercados-alvo) e sobre os recursos utilizados (tecnologias e ferramentas). O Anexo B (Modelos de artefatos da metodologia *WideWorkWeb*) apresenta modelos de formulários que poderão ser usados para a coleta de métricas.

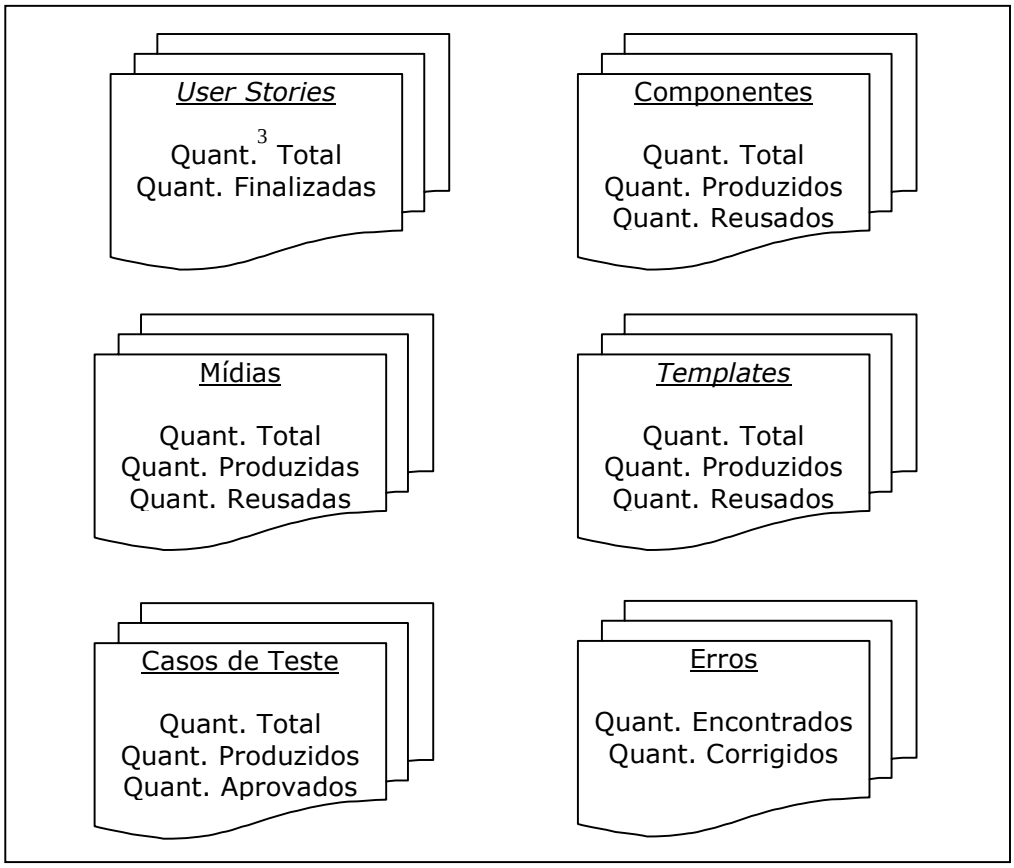

Figura 5.3 – Métricas para acompanhamento dos projetos

5.2 Ciclo de Vida das Aplicações Web

Mais que analisar o processo de desenvolvimento de uma aplicação, é importante que sejam discutidos os aspectos de negócios que conduzem a evolução dessa aplicação em relação ao mercado.

Afinal, conhecendo e avaliando os diferentes estágios do ciclo de vida de uma aplicação, os desenvolvedores poderão planejar melhor os investimentos e a alocação de profissionais para os projetos, uma vez que cada fase possui características e demandas diferenciadas.

[3] Quant. = Quantidade

Václav Rajlich e Keith Bennett [RAJLICH, 2000] apresentam uma definição para o ciclo de vida das aplicações em geral que se baseia nas atividades que integram a manutenção do sistema em cada fase. São cinco os estágios:

- **Desenvolvimento Inicial (*Initial Development)*:** vai desde o início do projeto até que a primeira versão executável da aplicação seja finalizada pelos desenvolvedores e colocada no mercado. A arquitetura do sistema é definida nessa etapa, de acordo com os requisitos iniciais. É importante que a equipe de desenvolvimento seja formada por profissionais especializados e experientes, que dominem as tecnologias a serem usadas.

- **Evolução *(Evolution)*:** são feitas modificações e extensões na aplicação para atender as necessidades dos usuários. A arquitetura e as funcionalidades do sistema vão sendo adaptadas aos novos requisitos, e novas versões são disponibilizadas freqüentemente. A demanda do mercado pela aplicação é alta, e os desenvolvedores devem se esforçar para mantê-la nesse estágio o maior tempo possível.

- **Atualização *(Servicing)*:** as alterações na arquitetura do sistema deixam de ser viáveis, seja por sua complexidade ou pelo investimento necessário. Os desenvolvedores corrigem apenas pequenas falhas da aplicação, que se torna cada vez menos atraente para o mercado.

- **Suporte *(Phaseout)*:** a empresa decide não investir mais em mudanças e correções no sistema, e tenta gerar receita a partir da última versão produzida pelo maior tempo possível. Não há mais investimentos, e a aplicação entra em declínio.

- **Finalização *(Closedown)*:** a aplicação é retirada do mercado. Se já houver um outro sistema que possa substituí-la disponível, os usuários são incentivados a migrar para essa nova aplicação. A empresa resolve pendências legais em relação ao código.

É importante observar que a evolução da aplicação ao longo dessas etapas tem conseqüências diretas sobre a formação da equipe de desenvolvimento e sobre a arquitetura do sistema. A participação de profissionais especializados, fundamental nas primeiras etapas do ciclo de vida da aplicação, deixa de ser relevante à medida que a arquitetura do sistema caminha para a estagnação, que pode ser acelerada por um projeto inicial pouco flexível e por modificações mal planejadas.

Daí a importância de uma análise precisa do ciclo de vida de uma aplicação. Conhecendo esses diferentes estágios, os gerentes de projeto podem conduzir as decisões de modo a otimizar o uso dos recursos para aproveitar as oportunidades do mercado. Além disso, podem evitar que investimentos desnecessários sejam feitos em aplicações que caminham para o declínio.

Considerando agora o caso específico das Aplicações Web, a Figura 5.4 (Ciclo de Vida das Aplicações Web) apresenta uma recomendação para a definição de seu ciclo de vida, tendo como base as atividades definidas na metodologia *WideWorkWeb* (item 5.1 - Metodologia para o Desenvolvimento de Aplicações Web num Cenário Global). Assim como no ciclo genérico proposto por Rajlich e Bennett [RAJLICH, 2000], o tipo de manutenção aplicado em cada estágio é o critério que marca a evolução da aplicação desde sua concepção até sua decadência.

Essa ênfase na fase de manutenção do sistema é bastante propícia para a análise das Aplicações Web, uma vez que os ciclos de manutenção são empregados com freqüência pelos desenvolvedores na tentativa de acompanhar as mudanças de requisitos do mercado.

A estória a seguir ilustra bem as fases do ciclo de vida das Aplicações Web. Tome-se o exemplo de uma aplicação transacional de comércio eletrônico. Após a análise de outros *sites* semelhantes, são definidos os requisitos para a aplicação e as equipes começam o desenvolvimento. Uma vez aprovada pelo Cliente, a aplicação é disponibilizada aos usuários e têm início as campanhas de divulgação. É o fim do Desenvolvimento Inicial.

À medida que os usuários visitam o *site* e adquirem produtos on-line, os administradores da loja virtual acompanham as mudanças no mercado e investem em novos serviços para os usuários, como e-mails com ofertas personalizadas e *chats* para pronto atendimento às dúvidas dos internautas. Além disso, o conteúdo das páginas é atualizado constantemente para indicar a disponibilidade dos itens no estoque e mudanças de preço. O monitoramento do tráfego do *site* também é feito para evitar que o sistema saia do ar ou apresente falhas. A aplicação está em plena Evolução.

Após um certo tempo, que pode variar de alguns meses a alguns anos, o *site* já oferece uma quantidade razoável de funcionalidades aos usuários, e a direção da loja virtual decide investir no visual da aplicação, oferecendo uma interface mais atraente, em vez de continuar incrementando sua arquitetura. Os Criadores de Código são afastados da equipe, que passa a ser composta basicamente por Criadores de Conteúdo e de Interface. É a fase de Atualização da aplicação.

Apesar do sucesso inicial da nova interface, a quantidade de visitas à loja virtual começa a diminuir, pois as aplicações concorrentes já apresentam novos serviços que não estão disponíveis no *site*, como leilões de itens raros do acervo. A queda nas receitas força a direção da empresa a reduzir gastos dispensando os Criadores de Conteúdo e de Interface. Apenas as despesas necessárias para manter a aplicação no ar são permitidas. O sistema está na fase de Suporte.

As informações do *site* deixam de ser atualizadas freqüentemente, e o interesse dos usuários diminui ainda mais. A loja virtual se torna cada vez menos competitiva no mercado e as despesas começam a superar as receitas. A direção da empresa decide encerrar suas operações e tirar a aplicação do ar, esclarecendo aos poucos usuários restantes que seus serviços não estarão mais disponíveis. A Finalização da aplicação marca o fim de seu ciclo de vida.

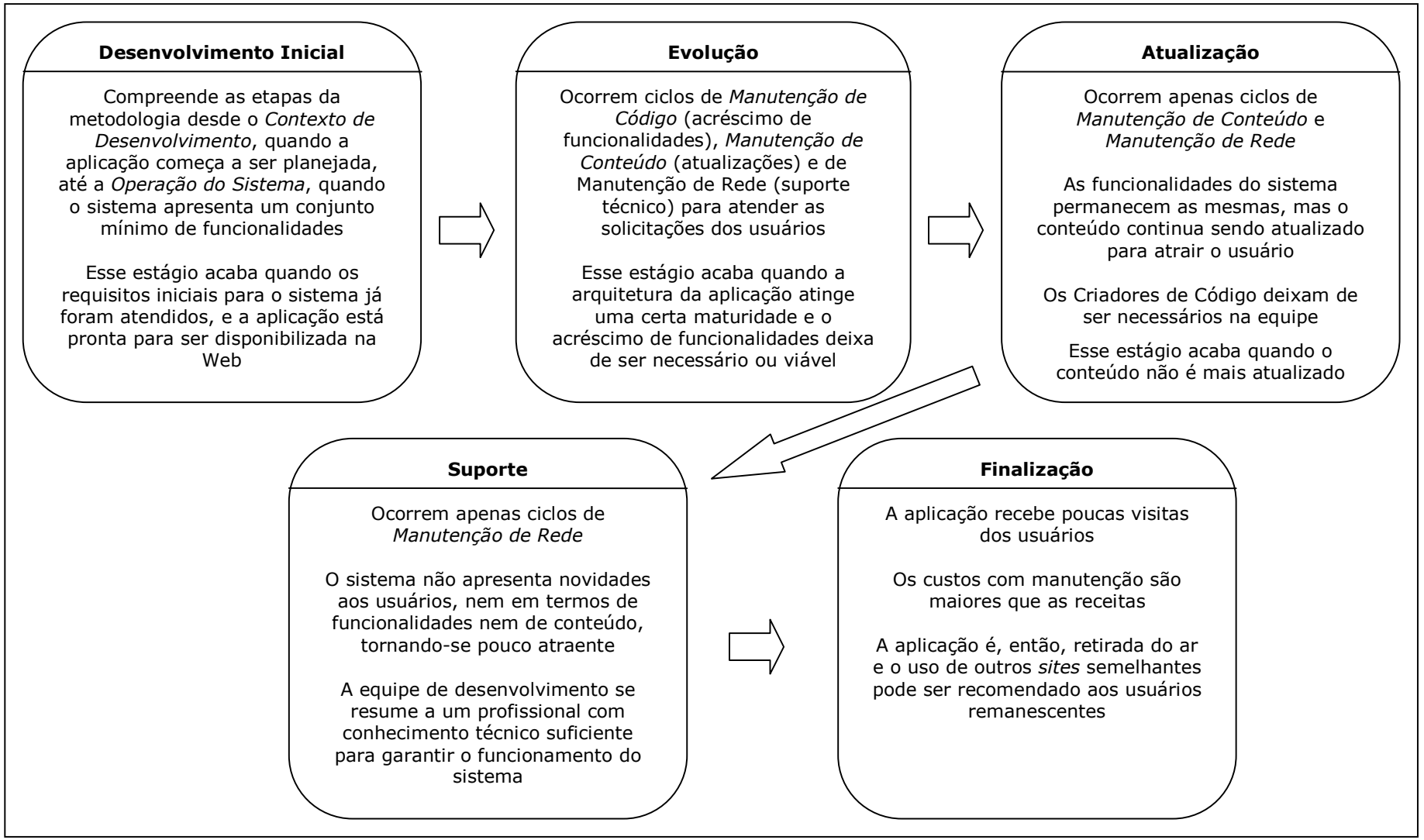

Figura 5.4 – Ciclo de vida das Aplicações Web

5.3 Considerações Finais

Após analisar, nos capítulos anteriores, as características e os desafios do Desenvolvimento de Aplicações Web e do Desenvolvimento Global de Software, o presente capítulo apresentou uma proposta de metodologia para auxiliar os profissionais de software envolvidos em projetos nessas áreas.

O escopo da metodologia é a produção de Aplicações Web transacionais (sistemas de *e-commerce*) por times pequenos e que podem estar distribuídos entre organizações ou mesmo entre países. Simplicidade, interatividade, agilidade e flexibilidade são os valores que dirigem suas atividades.

Ao longo do capítulo foram descritos em detalhes os papéis desempenhados pelos participantes envolvidos no desenvolvimento, a seqüência de atividades a serem realizadas, e os artefatos que devem ser gerados em cada etapa. Todas essas informações aparecem sintetizadas na Tabela 5.2 (Visão geral da metodologia *WideWorkWeb*), que oferece uma visão geral da metodologia.

É importante observar que na Manutenção de Código e na Manutenção de Conteúdo têm início novos ciclos de desenvolvimento, compostos por um subconjunto das demais etapas. Como a decisão de quais atividades deverão ser realizadas dependerá dos requisitos para a aplicação, não é possível definir com precisão que artefatos serão gerados, nem que participantes estarão envolvidos nessas etapas.

Foram sugeridas, ainda neste capítulo, ações gerenciais e métricas de suporte executivo para auxiliar no acompanhamento do trabalho dos times. Finalmente, foi apresentada uma recomendação para a definição do ciclo de vida das Aplicações Web baseada no tipo de manutenção realizada.

Para consolidar a metodologia, serão analisados no próximo capítulo os principais tipos de testes necessários no desenvolvimento de Aplicações Web, bem como exemplos de casos de teste e cenários para os testes manuais.

Além disso, como toda metodologia de desenvolvimento requer um suporte computacional adequado na realização de suas atividades, serão apresentadas no próximo capítulo as principais ferramentas disponíveis no mercado que poderão auxiliar os desenvolvedores.

Etapa	Objetivos	Sugestões	Artefatos Gerados	Participantes
Contexto do Desenvolvimento	▪ Definir as parcerias de negócios, a organização dos times e os mercados-alvo para a aplicação	▪ Realizar reuniões presenciais para discutir as decisões	▪ Análise de Riscos[4] ▪ *Statement Of Work* (SOW)	▪ Representantes das empresas envolvidas
Análise do Problema	▪ Definir requisitos funcionais e não-funcionais para a aplicação	▪ Avaliar aplicações semelhantes ▪ Realizar reuniões presenciais para discutir os requisitos ▪ Escolher tecnologias considerando reuso, manutenibilidade e portabilidade	▪ *User Stories* ▪ Modelo Arquitetural ▪ Modelo de Interface ▪ Métricas de Acompanhamento ▪ Critérios de Aceitação	▪ Clientes ▪ Mediadores das equipes
Decisões de Projeto	▪ Alocar tarefas e recursos, e definir cronogramas	▪ Atribuir *User Stories* independentes para cada time ▪ Usar ferramentas compartilhadas e repositório central de código dentro de cada time	▪ Plano de *Releases* ▪ *Site* de Acompanhamento[5] ▪ Relatório Gerencial	▪ Mediadores das equipes

Tabela 5.2 – Visão Geral da Metodologia *WideWorkWeb*

[4] A Análise de Riscos deve ser gerada no início de cada etapa.
[5] O *Site* de Acompanhamento e o Relatório Gerencial são atualizados durante todo o ciclo de produção-testes-integração.

Etapa	Objetivos	Sugestões	Artefatos Gerados	Participantes
Produção de Código	▪ Gerar módulos de software para atender os requisitos da aplicação	▪ Usar técnicas de *eXtreme Programming* (XP) ▪ Usar técnicas de Internacionalização ▪ Incluir comentários no código	▪ Detalhamento das *User Stories* ▪ Código (componentes) ▪ Registro de Erros e Modificações ▪ Casos de Teste e Cenários	▪ Mediador da equipe ▪ Criadores de Código ▪ Gerente de Testes ▪ Gerente de Qualidade
Produção de Conteúdo	▪ Gerar mídias (texto, imagem, som, etc) para compor o conteúdo do *site* ▪ Gerar a documentação do sistema para o usuário	▪ Usar definições do Modelo de Interface e Critérios de Aceitação ▪ Adaptar culturalmente as mídias	▪ Lista de Mídias ▪ Conteúdo (mídias) ▪ Documentação do Sistema ▪ Registro de Erros e Modificações ▪ Casos de Teste e Cenários	▪ Mediador da equipe ▪ Criadores de Conteúdo ▪ Gerente de Testes ▪ Gerente de Qualidade
Produção de Interface	▪ Gerar *templates* para as páginas Web da aplicação e montar a estrutura do *site*	▪ Usar definições do Modelo de Interface e Critérios de Aceitação ▪ Adaptar culturalmente o visual das páginas	▪ Interface (*templates*) ▪ Registro de Erros e Modificações ▪ Casos de Teste e Cenários	▪ Mediador da equipe ▪ Criadores de Interface ▪ Gerente de Testes ▪ Gerente de Qualidade

Tabela 5.2 – Visão Geral da Metodologia *WideWorkWeb* (cont.)

Etapa	Objetivos	Sugestões	Artefatos Gerados	Participantes
Testes e Integração	▪ Integrar e testar o código, o conteúdo e a interface produzidos para a aplicação ▪ Gerar um protótipo do sistema para ser avaliado	▪ Integrar e testar cada *User Story* isolada ▪ Depois integrar as *User Stories* da aplicação gradativamente para identificar falhas ▪ Realizar testes beta com representantes dos mercados-alvo	▪ Registro de Erros e Modificações ▪ Casos de Teste e Cenários ▪ *Post-Mortem*	▪ Mediadores das equipes ▪ Criadores de Código ▪ Gerente de Testes ▪ Gerente de Qualidade ▪ Cliente
Operação do Sistema	▪ Instalar a aplicação e disponibilizar o sistema na Web	▪ Usar scripts de instalação e *checklists* de atividades ▪ Criar plano de contingência (*rollback plan*) para reverter mudanças	▪ Plano de *Deployment* ▪ Registro de Erros e Modificações	▪ Mediadores das equipes ▪ Criadores de Código ▪ Gerente de Operação e Manutenção
Manutenção de Código	▪ Adicionar funcionalidades à aplicação ▪ Fazer a migração do sistema para uma nova tecnologia ▪ Corrigir falhas da aplicação	▪ Iniciar novo ciclo de desenvolvimento (a escolha das atividades que deverão ser realizadas dependerá dos requisitos para a aplicação)	▪ O conjunto de artefatos que serão gerados dependerá das atividades previstas pelo novo ciclo de desenvolvimento	▪ A escolha dos profissionais envolvidos dependerá das atividades previstas pelo novo ciclo de desenvolvimento

Tabela 5.2 – Visão Geral da Metodologia *WideWorkWeb* (cont.)

Etapa	Objetivos	Sugestões	Artefatos Gerados	Participantes
Manutenção de Conteúdo	▪ Atualizar conteúdo ▪ Incluir novas mídias ▪ Modificar a interface da aplicação	▪ Iniciar novo ciclo de desenvolvimento (a escolha das atividades que deverão ser realizadas dependerá dos requisitos para a aplicação)	▪ O conjunto de artefatos que serão gerados dependerá das atividades previstas pelo novo ciclo de desenvolvimento	▪ A escolha dos profissionais envolvidos dependerá das atividades previstas pelo novo ciclo de desenvolvimento
Manutenção de Rede	▪ Detectar e resolver problemas técnicos e falhas de conexão	▪ Verificar periodicamente os equipamentos ▪ Monitorar acessos ao *site*	▪ Registro de Erros e Modificações	▪ Gerente de Operação e Manutenção

Tabela 5.2 – Visão Geral da Metodologia *WideWorkWeb* (cont.)

Capítulo 6

6. Testes e Ferramentas para o Desenvolvimento Web

Conforme visto no Capítulo 2 (Desenvolvimento de Aplicações Web), a alta competitividade do mercado tem levado os desenvolvedores a se preocuparem cada vez mais com a qualidade das Aplicações Web produzidas.

Na busca da satisfação dos usuários, requisitos não-funcionais como confiabilidade, usabilidade e segurança tornam-se importantes diferenciais, e a melhor maneira de garantir que sejam atendidos é realizar um conjunto de testes efetivo durante e após o desenvolvimento.

Neste capítulo, serão destacadas as particularidades dos testes de Aplicações Web dentro da metodologia *WideWorkWeb* apresentada no capítulo anterior. Inicialmente, serão descritos os principais tipos de testes realizados em Aplicações Web, com indicações dos aspectos mais importantes a serem observados em cada categoria.

Em seguida, serão apresentados exemplos de casos de teste e cenários que devem ser produzidos durante o desenvolvimento da aplicação, bem como algumas recomendações para o preparo e execução dos testes.

De um modo geral, para que os desenvolvedores realizem com sucesso não só os testes das Aplicações Web, mas todas as atividades que compõem a metodologia *WideWorkWeb*, é fundamental que eles disponham de ferramentas de desenvolvimento adequadas.

Por isso, o objetivo da segunda parte deste capítulo é apresentar sugestões de ferramentas disponíveis no mercado que poderão auxiliar todos os envolvidos na produção de Aplicações Web dentro de um contexto global.

6.1 Testes Web

Embora seja extremamente importante para garantir a qualidade das aplicações, na maioria dos projetos, a fase de testes costuma ser deixada para o final do desenvolvimento, estando sujeita à disponibilidade de tempo e de orçamento.

Já no caso das Aplicações Web, que têm ciclos de produção mais curtos, é recomendável que os testes sejam planejados desde o início do projeto, e executados paralelamente ao desenvolvimento, pois a satisfação dos usuários é uma prioridade. Mais

que desejável, minimizar as falhas da aplicação deve ser uma meta para as empresas, uma vez que cada atualização é disponibilizada simultaneamente para inúmeros usuários e a presença de erros pode comprometer a credibilidade da aplicação.

É importante destacar que testar uma Aplicação Web é uma tarefa bem mais complexa que testar uma aplicação tradicional, pois os desenvolvedores não têm controle sobre a configuração do ambiente em que ela será executada. Ou seja, é preciso garantir o funcionamento da aplicação em diferentes sistemas operacionais, plataformas de hardware, e *browsers*, pois as máquinas clientes não seguem um padrão.

Além disso, devem ser realizados testes que considerem as características próprias de cada elemento da aplicação (código, conteúdo e interface), e que verifiquem requisitos como usabilidade e performance. Os principais tipos de testes são apresentados a seguir.

6.1.1 CATEGORIAS DE TESTES WEB

São várias as classificações e nomenclaturas utilizadas pelos autores [BECKER, 2002] [GERRARD, 2000a] [GERRARD, 2000b] [MACINTOSH, 2000] [PHILOSOPHE, 2000] [RAMLER, 2002] [STOUT, 2001a] para descrever os tipos de testes Web. Entretanto, todos são unânimes na ênfase dada aos requisitos não-funcionais das aplicações, que têm impacto direto sobre a qualidade percebida pelos usuários.

Para sintetizar as idéias desses autores e compor a metodologia *WideWorkWeb*, foram escolhidas dez categorias principais de testes que merecem atenção especial dos desenvolvedores. São elas: (1) unidade, (2) funcional, (3) *browser*, (4) performance, (5) segurança, (6) usabilidade, (7) conteúdo, (8) navegação, (9) integração e (10) aceitação.

- **Teste de Unidade**

O objetivo do teste de unidade é verificar o funcionamento de cada componente do código da aplicação de acordo com sua especificação. Por isso, é bastante recomendável que os testes de unidade sejam escritos antes mesmo do código para que os desenvolvedores esclareçam todas as dúvidas sobre a estrutura e as funcionalidades do componente.

Antes de ser integrado ao restante do sistema, cada componente deve passar por um teste de unidade para garantir que não apresenta mais falhas. Como são realizados ao longo de todo o desenvolvimento, os testes de unidade costumam ser automatizados na forma de scripts ou com a ajuda de ferramentas.

- **Teste Funcional**

Para verificar se a aplicação atende aos requisitos iniciais definidos pelo Cliente, são realizados testes funcionais que simulam as ações dos usuários no sistema, como o preenchimento de um formulário, ou a realização de uma busca.

Dois aspectos principais devem ser avaliados pelos desenvolvedores em um teste funcional: a realização de transações e a validação de dados.

Nas transações, é importante observar se a comunicação entre o *browser* e o banco de dados ocorre corretamente, sem perda de informações, e como o sistema se comporta quando uma transação é interrompida ou refeita. Por exemplo, o que acontece se, durante uma compra on-line, o usuário tentar parar o carregamento da página de confirmação, ou voltar para o passo anterior? As informações são perdidas? A compra é registrada duas vezes? Essas questões devem ser respondidas no início do desenvolvimento e testadas nesse momento.

Quanto à validação de dados, os desenvolvedores devem utilizar dados válidos e inválidos no teste para checar se o processamento ocorre corretamente, ou se as mensagens de erro apropriadas são exibidas. Se num determinado campo de um formulário de cadastro são permitidos apenas caracteres numéricos, é preciso verificar o que acontece quando um usuário tenta inserir letras ou sinais de pontuação.

❖ **Teste de *Browser***

Principalmente no caso de aplicações de *e-commerce*, é importante garantir que o maior número possível de usuários possa ter acesso ao sistema. Para isso, os desenvolvedores precisam considerar a heterogeneidade das máquinas onde será executada a aplicação.

Ou seja, combinações dos diferentes *browsers*, sistemas operacionais, e plataformas de hardware disponíveis no mercado devem ser testadas para observar se o código é executado corretamente e se a aparência do *site* não se modifica.

O uso de *frames*, *plug-ins*, animações, JavaScript e algumas *tags* HTML merece atenção especial dos desenvolvedores. Quanto à interface, é preciso observar as cores, a posição dos elementos na tela, o uso de tabelas e menus.

Como não seria viável testar todas as combinações possíveis, seja pelo tempo, seja pelo custo, é importante escolher um conjunto de opções que correspondam à maior parte do público-alvo da aplicação e garantir o funcionamento adequado do *site* nessas configurações.

É certo que essa não é uma escolha fácil, dada a diversidade de usuários, principalmente em mercados globais. Entretanto, essa decisão deve ser tomada pelo Cliente no início do projeto, levando em conta seus interesses de negócios.

❖ **Teste de Performance**

Além da configuração das máquinas, a taxa de conexão dos usuários também é bastante variável. Enquanto alguns utilizam banda larga, outros acessam a aplicação através de *modems*.

O objetivo dos desenvolvedores no teste de performance é medir o tempo de resposta para as transações e a velocidade de *download* das páginas e verificar se os valores encontram-se dentro dos padrões definidos pelos critérios de aceitação para cada tipo de conexão. Assim, é possível identificar arquivos muito grandes ou objetos que são exibidos apenas parcialmente.

Mais que medir a taxa de transmissão dos dados, os desenvolvedores devem observar o desempenho da aplicação quando utilizada simultaneamente por uma grande quantidade de pessoas. É preciso testar os limites do sistema para saber em que circunstâncias sua performance fica comprometida.

❖ **Teste de Segurança**

Os desenvolvedores devem se certificar de que a aplicação não apresenta falhas que permitam o acesso de usuários não autorizados. Ou seja, é preciso garantir que o sistema está protegido contra os ataques mais comuns dos *hackers*.

É muito importante checar se a aplicação é tolerante a falhas durante as transações, para evitar que dados dos usuários, como números de cartão de crédito e senhas, sejam expostos.

Não permitir que os diretórios do sistema possam ser listados e usar mensagens criptografadas para troca de dados são algumas medidas de segurança que devem ser verificadas nesse teste.

❖ **Teste de Usabilidade**

Esse tipo de teste é utilizado para avaliar a impressão dos usuários sobre a facilidade de uso e adequação da aplicação a suas necessidades. É recomendável que representantes do público-alvo da aplicação possam apresentar dúvidas, sugestões e críticas.

No caso das Aplicações Web, os desenvolvedores podem disponibilizar, através da Internet, uma versão beta do sistema, e colher opiniões usando formulários on-line ou e-mails.

De qualquer forma, antes do teste com os usuários, é preciso verificar alguns aspectos importantes, como a consistência do *layout* entre as páginas e o uso de conceitos de Internacionalização e Localização no *site*.

Para que os usuários possam se familiarizar com o funcionamento da aplicação, é conveniente que todas as páginas apresentem uma aparência semelhante, mantendo as cores e a posição dos menus, por exemplo. Os desenvolvedores devem checar visualmente as páginas do *site* para ver se o padrão está sendo mantido. Essa tarefa é facilitada com o uso de *templates* durante a criação das páginas.

Se a aplicação tiver sido desenvolvida para diferentes mercados, é necessário verificar se todas as mensagens exibidas nas páginas estão no idioma correto e se os formatos de dados usados (ex: endereço, telefone, data) estão de acordo com as convenções da cultura de cada mercado. Também é preciso checar as cores usadas e a disposição dos elementos na tela, respeitando as convenções de cada povo.

- **Teste de Navegação**

O objetivo desse teste é verificar se a navegação entre as páginas da aplicação ocorre da maneira prevista, ou seja, se todos os *links* estão corretos e fazem referência aos arquivos desejados.

Para isso, os desenvolvedores podem montar um grafo para indicar a dependência entre as páginas, e procurar *links* incorretos ou para arquivos que já foram movidos. Essa verificação pode ser automatizada com a ajuda de ferramentas.

- **Teste de Conteúdo**

É preciso garantir a qualidade do conteúdo que será apresentado ao usuário. No caso dos textos, devem ser verificadas as regras gramaticais dos idiomas e a completude das informações. Para as imagens e gráficos, é necessário observar a nitidez e o uso das cores. Também é preciso verificar se o código HTML não apresenta erros, como *tags* incompletas ou atributos sem valores.

Essa verificação pode ser feita pelos próprios desenvolvedores durante a produção do conteúdo, com a ajuda de corretores ortográficos, dicionários e editores gráficos.

- **Teste de Integração**

À medida que vão sendo produzidos, os componentes do código vão sendo integrados para compor o sistema. Essa integração ocorre gradativamente, um componente de cada vez, e deve ser seguida sempre de uma verificação das interfaces dos componentes para garantir que a comunicação entre eles ocorra como esperado. São os testes de integração.

Os testes de integração também são realizados no conteúdo e na interface do sistema, através do carregamento das páginas no *browser* e da navegação pelo sistema.

Como a metodologia WideWorkWeb propõe o desenvolvimento do sistema na forma de *user stories* independentes, os testes de integração são executados quando o código, o conteúdo e a interface são unidos para compor cada *user story*, e quando as *user stories* são integradas para formar o sistema.

Durante o teste de integração do sistema como um todo, os desenvolvedores observam como se dá a comunicação entre as diferentes camadas da arquitetura da aplicação, vistas no Capítulo 2 (Desenvolvimento de Aplicações Web). É preciso garantir que os dados sejam trocados corretamente entre o *browser*, o servidor Web, o servidor de aplicação, o servidor de banco de dados e, se for o caso, os sistemas legados e os sistemas externos à aplicação (Ex: operadora de cartão de crédito para compras on-line).

Outro momento importante em que são realizados os testes de integração é durante a manutenção do sistema, quando são chamados também de testes de regressão. Cada vez que um novo código é acrescentado à aplicação, ou que são feitas atualizações em seu conteúdo ou interface, ou até quando são instaladas novas versões para os programas, os desenvolvedores devem testar a integração dos elementos para verificar se o funcionamento do sistema continua ocorrendo corretamente.

- **Teste de Aceitação**

É realizado com o Cliente antes da aplicação ser disponibilizada ao público, mas também pode ser feito aos poucos, durante o desenvolvimento, quando houver disponibilidade de tempo.

O objetivo é avaliar se a aplicação atende às expectativas do Cliente e se ela está de acordo com os requisitos funcionais e os critérios de aceitação definidos no início do projeto.

Para checar os requisitos funcionais, o Cliente pode escolher algumas atividades que serão comumente realizadas pelos usuários, como realizar uma compra ou comparar preços de produtos em um *site* de comércio eletrônico, e executar os passos necessários para completar as tarefas observando o comportamento do sistema e a aparência das páginas.

Já no caso dos requisitos não-funcionais, identificados pelos critérios de aceitação, o Cliente pode analisar os relatórios gerados a partir da realização dos testes para comparar os resultados obtidos com os esperados.

A Tabela 6.1 (Exemplos de critérios de aceitação para Aplicações Web) apresenta exemplos de critérios de aceitação definidos para uma aplicação de

comércio eletrônico, que podem ser conferidos pelo Cliente durante o teste de aceitação [BECKER, 2002].

Aspecto	Critério de Aceitação
***Layout* das páginas**	As páginas devem conter as cores padrão da empresa.
Consistência do design	A barra de menu deve aparecer no topo de cada página. O menu deve conter apenas texto; nenhuma figura deve ser usada.
Padrões de design	O *site* deve ser acessível para *browsers* Netscape ou Internet Explorer a partir da versão 3.x para ambos. A organização da página deve ser da esquerda para a direita.
Performance	A velocidade de *download* de uma página Web não deve passar de 5 segundos usando um *modem* com velocidade de 28.8 Kbytes/s.
Conteúdo das páginas	Todas as informações devem ser coletadas, armazenadas e atualizadas num banco de dados. O conteúdo das páginas deve ser sempre atual, e não deve ser anterior a 60 dias. Todas as informações devem estar em inglês.
Segurança	O *site* deve oferecer o protocolo de segurança SSL2 para o acesso a todas as contas e para as transações de pagamento, registro e mudança de senhas.
Navegação	A navegação no *site* deve ser simplificada pela implementação de uma única barra de menu no topo de cada página, com *links* sublinhados e botões do tipo padrão.

Tabela 6.1 – Exemplos de critérios de aceitação para Aplicações Web

6.1.2 CASOS DE TESTE E CENÁRIOS

Como os ciclos de desenvolvimento Web são curtos, é muito importante que os desenvolvedores planejem bem a realização dos testes. Para isso, devem identificar quais são os aspectos críticos de suas aplicações e que tipos de testes são mais relevantes.

O primeiro passo é determinar os cenários que serão testados, tomando como base as *user stories* definidas inicialmente para representar as funcionalidades da aplicação. Assim, um cenário corresponde ao conjunto de passos necessários para que o usuário realize uma determinada tarefa no *site*. Alguns exemplos de cenários comuns em aplicações de comércio eletrônico são o cadastro de clientes (Figura 6.1 – Exemplo de cenário para Aplicações Web), a busca de itens no catálogo de produtos, a escolha de itens para compra

e o pagamento de pedidos. Cada cenário deve caracterizar bem os requisitos para o ambiente em que será realizado, como a configuração da máquina, o sistema operacional, o tipo e a versão do *browser*, para que a avaliação dos resultados seja precisa.

Cenário: Cadastro de um cliente
Passos:

1. Num ambiente Windows 98, carregue a versão 3.0 do Internet Explorer.
2. No campo URL, digite http://www.shopping.com.br e tecle ENTER.
3. No menu esquerdo, clique no link Cadastre-se.
4. No formulário exibido, preencha os campos indicados:
    a. Nome de usuário (login)
    b. Senha para acesso
    c. E-mail
    d. Nome completo
    e. CPF
    f. Endereço
5. Clique no botão OK que aparece no final do formulário.

Figura 6.1 – Exemplo de cenário para Aplicações Web

Os cenários fornecem uma visão macro do funcionamento do sistema, pois sua realização costuma envolver a comunicação das diferentes camadas da aplicação, e são muito usados em testes de aceitação e integração, além dos testes funcionais. Devido à sua complexidade, dificilmente são automatizados, tendo de ser realizados manualmente.

A partir da definição dos cenários, são analisadas as condições para a realização de cada um de seus passos. São os chamados casos de teste, utilizados para verificar se o resultado obtido em uma determinada operação corresponde ao resultado esperado, definido pelos requisitos do sistema.

Por exemplo, considerando o cenário da Figura 6.1 (Exemplo de cenário para Aplicações Web), podem ser definidas restrições para o preenchimento de cada campo do formulário. Supondo que o campo *login* deva conter uma cadeia de 6 caracteres alfanuméricos, seria preciso testar o que aconteceria se o usuário tentasse cadastrar um *login* com 4 ou 8 caracteres, ou ainda se deixasse o campo em branco.

Mesmo não sendo viável desenvolver casos de teste para todas as possibilidades, é preciso escolher um conjunto de condições que simule os comportamentos mais prováveis dos usuários, principalmente os erros mais comuns, para garantir a qualidade da aplicação.

Cada caso de teste é composto por uma condição, uma descrição dos passos a serem seguidos para realizar a verificação, a indicação do resultado esperado, e informações sobre a aprovação ou reprovação no teste e os erros encontrados [RICE, 2002]. A Figura 6.2 (Cenário para autenticação de usuários numa Aplicação Web) apresenta um exemplo de cenário para autenticação de usuários numa aplicação Web e a Tabela 6.2 (Casos de teste

para a autenticação de usuários numa Aplicação Web) apresenta os respectivos casos de teste para esse cenário.

Cenário: Autenticação de um usuário
Passos:

1. Num ambiente Windows 98, carregue a versão 3.0 do Internet Explorer.
2. No campo URL, digite http://www.shopping.com.br e tecle ENTER.
3. No menu esquerdo, clique no link Autenticação.
4. No formulário exibido, preencha os campos indicados:
    a. Nome de usuário (login)
    b. Senha para acesso
5. Clique no botão OK que aparece ao lado do campo Senha.

Figura 6.2 – Cenário para autenticação de usuários numa Aplicação Web

Condições de Teste	Ações	Resultados Esperados	Aprovado/ Reprovado	Erro
Login correto Senha correta	1. Inserir no campo *login* um nome de usuário válido (já cadastrado no sistema) 2. Inserir no campo senha o código de autenticação do respectivo usuário	A mensagem "Usuário autenticado" é exibida.	Aprovado	
Login correto Senha incorreta	1. Inserir no campo *login* um nome de usuário válido (já cadastrado no sistema) 2. Inserir no campo senha uma seqüência de caracteres diferente do código de autenticação do respectivo usuário	A mensagem "Senha incorreta" é exibida.	Reprovado	Foi exibida a mensagem "Usuário não cadastrado"
Login incorreto Senha incorreta	1. Inserir no campo *login* um nome de usuário inválido (não cadastrado no sistema) 2. Inserir no campo senha uma seqüência qualquer de caracteres	A mensagem "*Login* incorreto" é exibida.	Aprovado	
Login em branco Senha incorreta	1. Não inserir nenhum caractere no campo *login.* 2. Inserir no campo senha uma seqüência qualquer de caracteres	A mensagem "O *login* não foi informado" é exibida.	Aprovado	
Login em branco Senha em branco	1. Não inserir nenhum caractere no campo *login.* 2. Não inserir nenhum caractere no campo senha.	A mensagem "O *login* não foi informado" é exibida.	Reprovado	Foi exibida a mensagem "Senha incorreta"
Login correto Senha em branco	1. Inserir no campo *login* um nome de usuário válido (já cadastrado no sistema) 2. Não inserir nenhum caractere no campo senha.	A mensagem "A senha não foi informada" é exibida.	Aprovado	

Tabela 6.2 – Casos de teste para a autenticação de usuários numa Aplicação Web

A maioria dos casos de teste relacionados ao código são automatizados na forma de testes de unidade para as aplicações, facilitando o trabalho dos desenvolvedores.

É importante destacar que nos cenários e casos de teste apresentados anteriormente não são indicados valores de teste, mas sim parâmetros de teste. Dessa forma, os desenvolvedores podem escrever um mesmo script de teste para ser utilizado com diferentes dados.

São os Testes Orientados a Dados (*Data Driven Tests*), uma abordagem sugerida para garantir o reuso dos cenários e casos de teste [HENDRICKSON, 2000]. Durante a execução dos scripts, os parâmetros podem ser informados manualmente ou lidos a partir de um arquivo específico. Além de permitir que um conjunto muito maior de dados seja testado sem esforços adicionais, essa prática facilita a manutenção das aplicações,

reduzindo a necessidade de mudanças nos casos de teste cada vez que uma nova condição é acrescentada.

Além do código, os casos de teste são escritos para avaliar também a interface e o conteúdo da aplicação. Neste caso, as condições funcionam como uma *checklist* do que precisa ser examinado pelo desenvolvedor. Alguns desses testes podem ser automatizados com a ajuda de ferramentas. A Figura 6.3 (Exemplos de casos de teste para a interface) apresenta um exemplo de casos de teste para verificar os *links* de uma Aplicação Web [PHILOSOPHE, 2000].

Casos de Teste de *Link*: para cada *link* em cada página, verificar se:

1. O *link* funciona (não está quebrado)
2. O *link* aponta para a página correta
3. O texto do *link* descreve claramente a página que será exibida
4. O *link* segue os padrões de design do *site* (cor, fonte, etc)

Figura 6.3 – Exemplo de casos de teste para a interface

6.1.3 RECOMENDAÇÕES PARA OS TESTES WEB

O sucesso da realização dos testes das Aplicações Web começa a ser definido antes mesmo do início do desenvolvimento, na fase de Análise do Problema, quando são estabelecidos critérios de aceitação para o sistema.

Auxiliado pelos Mediadores, o Cliente define parâmetros que servirão para avaliar o atendimento aos requisitos da aplicação, principalmente os não-funcionais. Esses parâmetros serão usados pelos desenvolvedores na produção dos testes e durante as simulações de uso do sistema.

Ainda na fase Análise do Problema, os Mediadores são responsáveis pela escolha dos testes a serem realizados a partir das categorias de teste descritas no item 6.1.1 (Categorias de Testes Web). O objetivo é priorizar os testes de acordo com a relevância de cada aspecto para a qualidade da aplicação. Por exemplo, testes de segurança são mais críticos para aplicações transacionais do que para aplicações informativas, que precisam de ênfase nos testes de conteúdo.

Os testes devem começar a serem escritos desde o início do desenvolvimento, e, no caso dos testes de unidade, antes mesmo do código. Mais que ajudar a planejar melhor a aplicação, essa prática incentiva os desenvolvedores a encararem os testes como atividades inerentes a seu trabalho, e não como uma etapa a ser realizada no final do projeto.

Além do Gerente de Testes, é importante que os Criadores de Código, Conteúdo e Interface participem da elaboração dos testes. Produzindo cenários e casos de teste para cada elemento da aplicação isoladamente, é possível antecipar e corrigir falhas, minimizando os riscos de integração de cada *user story*, e do sistema como um todo.

Mas, como elaborar cenários e casos de teste é uma tarefa difícil e que pode consumir muito tempo, o que não é desejável para a produção de Aplicações Web, o Gerente de Testes, juntamente com o Mediador do projeto, deve determinar que cenários e casos de teste precisam ser elaborados e automatizados, considerando principalmente as possibilidades de reuso e a relevância de cada componente. Em alguns casos, os casos de teste podem ser substituídos por inspeções e revisões no código, ou pela simples análise visual da interface, sem que procedimentos específicos precisem ser estabelecidos e documentados.

Também é preciso que a equipe de desenvolvimento disponha de ferramentas adequadas para a realização dos testes. Automatizando cenários e casos de teste, é possível agilizar os ciclos de produção e, principalmente, a manutenção da aplicação. Cada vez que um novo componente (código, conteúdo ou interface) precisa ser acrescentado ao sistema, devem ser realizados os testes no componente e, após a integração, na aplicação resultante.

Como a manutenção das Aplicações Web ocorre constantemente, com o acréscimo de funcionalidades e a atualização de conteúdo, os desenvolvedores devem ser criteriosos na escolha dos testes a serem realizados cada vez que a aplicação é modificada, para que o esforço seja minimizado. Sempre que possível, os testes manuais devem ser substituídos pelos testes automatizados, que podem ser executados mais rapidamente e facilitam o reuso.

Outro aspecto complementar ao uso de boas ferramentas é a existência de um ambiente de testes eficaz. Para isso, é preciso que as máquinas utilizadas possam simular a configuração das máquinas clientes onde a aplicação será executada, dando aos desenvolvedores uma visão mais próxima do comportamento real do sistema. Até aspectos simples, como o tamanho do banco de dados e a quantidade de memória disponível, podem afetar o desempenho da aplicação e merecem atenção.

Dada a diversidade de sistemas operacionais, plataformas de hardware, *browsers* e taxas de conexão, simular todos os ambientes possíveis seria inviável. É preciso definir um escopo de teste que englobe as opções dos usuários potenciais.

Durante a manutenção, a realização de testes se torna ainda mais complexa, pois não é possível tirar a aplicação do ar para fazer a integração nem verificar as falhas encontradas. Para que o desenvolvimento, os testes e a manutenção do sistema possam ocorrer de maneira concorrente, sem prejuízos para nenhuma das atividades, Andréa MacIntosh e Wolfgang Strigel [MACINTOSH, 2000] propõem um ambiente mínimo, formado por três servidores:

- Servidor de Desenvolvimento: é usado pelos especialistas na criação de código, conteúdo e interface para a aplicação.

- Servidor Intermediário: é atualizado periodicamente com novas versões dos elementos da aplicação. Pode servir como plataforma de integração e de testes, e deve ter uma configuração semelhante à das máquinas clientes do

público-alvo. As modificações feitas não afetam nem os desenvolvedores nem o sistema em funcionamento.

- Servidor de Produção: disponibiliza a aplicação para os usuários. Seus arquivos são atualizados periodicamente a partir do Servidor Intermediário.

No caso de projetos com times distribuídos, é recomendável que esse ambiente seja reproduzido em cada centro de desenvolvimento.

De um modo geral, mesmo em projetos com cronogramas curtos e recursos limitados, é possível realizar um conjunto mínimo de testes para garantir a qualidade da aplicação [PHILOSOPHE, 2000]. A prioridade deve ser assegurar o funcionamento correto da aplicação, testando os *links* e as principais atividades (*user stories*) e observando se os resultados obtidos correspondem às expectativas. Feito isso, os desenvolvedores podem observar alguns aspectos não-funcionais, como o funcionamento da aplicação em diferentes *browsers* e plataformas, e o uso de *cookies* e *JavaScript* nas páginas.

6.2 Ferramentas para o Desenvolvimento de Aplicações Web num Cenário Global

Além de adotar uma metodologia de desenvolvimento que reflita a realidade dos desenvolvedores, para que um projeto tenha sucesso é preciso que as ferramentas adequadas estejam disponíveis.

No caso específico do desenvolvimento de Aplicações Web por times distribuídos, as ferramentas necessárias podem ser classificadas em três grandes categorias, de acordo com seus propósitos: Ferramentas de Comunicação, Ferramentas de Desenvolvimento e Ferramentas de Gestão.

❖ Ferramentas de Comunicação

Como os times estão distribuídos, é fundamental que eles disponham de tecnologias de comunicação para amenizar os problemas decorrentes da distância. É desejável que, além dos meios de comunicação convencionais, como telefone e fax, cada centro de desenvolvimento possua ainda equipamentos para áudio e vídeo-conferência. O contato entre os integrantes de um mesmo time e entre os Mediadores dos times pode ocorrer também através de e-mails, *chats* e listas de discussão.

As ferramentas usadas para controlar as reuniões virtuais com áudio e vídeo-conferência devem garantir o sincronismo na troca de dados e evitar que falhas na conexão prejudiquem a qualidade das imagens e do som.

Um recurso importante presente em algumas ferramentas, como NetMeeting e Groove, é o compartilhamento de *desktop* entre duas máquinas conectadas. Assim, durante uma reunião virtual, um participante pode ter acesso aos

artefatos disponíveis na máquina de seu interlocutor, e vice-versa, permitindo que haja troca de idéias durante a apresentação e quaisquer dúvidas sejam esclarecidas.

- **Ferramentas de Desenvolvimento**

Como as Aplicações Web são compostas por três elementos distintos (código, conteúdo e interface), ferramentas específicas para cada um desses artefatos são necessárias.

Na Produção de Código, os desenvolvedores utilizam ferramentas como editores de código e de scripts, compiladores, Sistemas Gerenciadores de Bancos de Dados (SGBDs), servidores Web e servidores de aplicação, além das ferramentas CASE comuns nos projetos tradicionais. Uma vez que o uso da linguagem Java é predominante na produção de Aplicações Web, os desenvolvedores costumam utilizar ferramentas compatíveis a seus padrões, como o compilador JDK e a ferramenta para testes de unidade JUnit. Editores de páginas Web também podem ser necessários na hora de escrever o código que será embutido nas páginas, como rotinas para autenticação de dados em formulários.

Já na Produção de Conteúdo, o grande destaque são as ferramentas para edição de imagens e sons, e para produção de animações e vídeos, que costumam ser ricas em recursos como cores e efeitos visuais. Corretores ortográficos e tradutores também são úteis para gerar conteúdo textual em diferentes idiomas.

Na Produção de Interface, os desenvolvedores podem contar com editores especiais, do tipo WYSIWYG[6], que evitam que o código HTML precise ser escrito manualmente. Alguns desses editores, como Dreamweaver e Front Page, permitem que sejam adicionados componentes de interface e de JavaScript nas páginas, e auxiliam na montagem da hierarquia do *site*. Alguns editores de imagens também podem ser usados pelos desenvolvedores para trabalhar a aparência das páginas.

Muito importantes ainda para o desenvolvimento de Aplicações Web são as ferramentas para automatização de testes. Para cada categoria apresentada no item 6.1.1 (Categorias de Testes Web) existem ferramentas específicas, que simulam o uso da aplicação e ajudam os desenvolvedores na hora de definir casos de teste.

[6] What You See Is What You Get: tipo de editor que apresenta na tela a aparência final da mídia em cada instante, permitindo que o desenvolvedor visualize as modificações instantaneamente.

❖ **Ferramentas de Gestão**

Embora a gestão de projetos seja considerada o grande desafio do Desenvolvimento de Aplicações Web num cenário global, já existem no mercado algumas ferramentas eficientes para auxiliar os Mediadores.

Para lidar com times distribuídos, é importante que os gerentes utilizem ferramentas para o planejamento e escalonamento das atividades, de modo que o trabalho das equipes ocorra em paralelo e da forma mais independente possível.

Uma ferramenta para controle de versões também é necessária para que o acesso aos repositórios de código seja controlado. O registro de erros e a notificação automática de modificações evita que os times e os Mediadores percam tempo trabalhando com versões antigas ou corrigindo defeitos já removidos.

Em projetos com times distribuídos, ou que tenham como objetivo a produção de aplicações para diferentes mercados, o uso de uma boa ferramenta para controle de versões é imprescindível.

Podem ser usadas ainda ferramentas especiais para a coleta e acompanhamento de métricas do projeto. No caso da metodologia *WideWorkWeb*, que propõe um conjunto simplificado de métricas quantificáveis, a coleta pode ser feita pela simples observação dos repositórios de artefatos.

A Tabela 6.3 (Exemplos de ferramentas) apresenta alguns exemplos de ferramentas disponíveis no mercado que podem ser usadas pelos desenvolvedores na produção de Aplicações Web em projetos distribuídos. Sugestões de outras ferramentas podem ser encontradas em [BENNATAN, 2002][ZETTEL, 2001][BURDMAN, 1999][SOFTWAREQA, 2001].

É importante destacar que, embora muitas ferramentas estejam disponíveis no mercado, ainda não existem ambientes integrados que ofereçam aos desenvolvedores todos os recursos necessários em projetos desse tipo, como o que a empresa Fujitsu utiliza internamente [AYOAMA, 1998] [GAO, 1999].

Além disso, a maioria das ferramentas são soluções proprietárias, que representam um alto investimento para os projetos, principalmente quando os centros de desenvolvimento estão distribuídos. CVS, GNATS, Nessus, Bugzilla e Empirix são bons exemplos de ferramentas gratuitas ou de código aberto que atendem as necessidades dos usuários e que podem ser usadas pelos participantes distribuídos.

Propósito	Ferramentas	
Comunicação entre os Times	MS NetMeeting (http://www.microsoft.com) TeamPortal [HANDEL, 2000] Groove (http://www.groove.net) InterPOD e TeamSmart [TAKAHASHI, 2000]	
Produção de Código	WebSphere (http://www-3.ibm.com/software/awdtools/studiositedev/) JDK (http://www.jdk.com)	
Produção de Conteúdo	Adobe Photoshop (http://www.adobe.com) Macromedia Freehand (http://www.macromedia.com) Macromedia Flash (http://www.macromedia.com) Paint Shop Pro (http://www.jasc.com) MS Gif Animator (http://www.microsoft.com)	
Produção de Interface	MS Front Page (http://www.microsoft.com) Macromedia Dreamweaver (http://www.macromedia.com)	
Automatização de Testes	Testes de Unidade	Junit (http://www.junit.org)
	Testes de Performance	SmartTest (http://www.inw.com) EasyWebLoad (http://www.easywebload.com)
	Testes de Interface	Empirix (http://www.empirix.com) RealValidator (http://arealvalidator.com/)
	Testes de Navegação	SiteSweeper (http://www.sitetech.com) Web Link Validator (http://www.relsoftware.com)
	Testes de Segurança	SecurityMetrics Appliance (http://www.securitymetrics.com) Nessus Security Scanner (http://www.nessus.org/)
	Testes de *Browser*	Browser Cam (http://www.browsercam.com)
Gerência de Projeto	MS Project (http://www.microsoft.com) Project KickStart (http://www.projectkickstart.com) Milestones (http://www.kidasa.com) MILOS (http://wwwagr.informatik.uni-kl.de/~milos/)	
Controle de Versões	Concurrent Versions System – CVS (http://www.cvshome.org) Clear Case Multisite (http://www.rational.com)	
Registro de Erros e Modificações	Clear Quest Multisite (http://www.rational.com) Groove (http://www.groove.net) GNATS (http://www.gnu.org/software/gnats/) Bugzilla (http://www.mozilla.org/bugs) Bugbase (http://www.threerock.com/products_bugbase.php)	

Tabela 6.3 – Exemplos de Ferramentas

Capítulo 7

7. Conclusão

Neste capítulo, serão analisadas as contribuições deste trabalho para a comunidade de software, considerando a relevância e originalidade dos resultados obtidos. Serão ainda apresentadas sugestões para trabalhos futuros que poderão usufruir os conhecimentos aqui reunidos para aprofundar pesquisas e melhorar seus processos de desenvolvimento.

7.1 Contribuições

Dentre os resultados gerados com este trabalho, o maior deles certamente foi a proposta de uma abordagem de desenvolvimento que une duas fortes tendências da produção de software atual: a criação de Aplicações Web e a participação de times distribuídos no desenvolvimento de software para mercados culturalmente distintos.

A partir de uma análise detalhada de cada uma dessas abordagens, indicando suas vantagens e seus desafios, foram apresentadas soluções práticas na forma de uma metodologia de desenvolvimento que pode ser adotada por todos aqueles que desejem conduzir projetos nessas áreas.

É importante destacar a riqueza das revisões bibliográficas realizadas, principalmente no caso do Desenvolvimento Global de Software, pois ainda não existem publicações em português sobre o assunto.

Os principais resultados obtidos aparecem listados a seguir:

- análise do cenário de Desenvolvimento Web, considerando as características e a arquitetura das aplicações, bem como as particularidades de seu desenvolvimento;

- avaliação da adequação de algumas metodologias propostas pela comunidade de software, como os Processos Agile, em especial eXtreme Programming e Scrum, para o desenvolvimento de Aplicações Web;

- mapeamento das mudanças provocadas pelo Desenvolvimento Global de Software nos aspectos de negócio, de desenvolvimento e gerenciais do ciclo de vida das aplicações;

- indicação dos principais desafios decorrentes da coordenação de equipes geograficamente dispersas e apresentação das soluções adotadas por grandes empresas como Alcatel e Bell Labs;

- avaliação do impacto do uso de técnicas de Internacionalização e Localização na produção de aplicações para diferentes mercados, com dicas a serem seguidas pelos desenvolvedores;

- proposta de uma metodologia de desenvolvimento para a produção de Aplicações Web num cenário Global (*WideWorkWeb*), com indicação detalhada das atividades a serem realizadas, dos papéis que os desenvolvedores devem desempenhar, bem como dos artefatos necessários e das métricas a serem coletadas durante o projeto, além de técnicas gerenciais para garantir o sucesso dos projetos;

- recomendação para a definição do ciclo de vida das Aplicações Web, tendo como base os diferentes tipos de manutenção que podem ser aplicados a esse tipo de software;

- análise sobre as particularidades dos testes de Aplicações Web, com indicações das principais categorias de teste e de exemplos de cenários e casos de teste que os desenvolvedores devem produzir, além de recomendações para a realização dos testes;

- indicação de ferramentas proprietárias e de código aberto disponíveis no mercado para auxiliar os desenvolvedores na produção de Aplicações Web e no acompanhamento de projetos.

7.2 Trabalhos Futuros

Como tanto o Desenvolvimento de Aplicações Web como o Desenvolvimento Global de Software são tendências que vêm despertando cada vez mais interesse da comunidade de software, é bastante natural que o presente trabalho origine novas pesquisas e aprofundamentos sobre o assunto, seja na forma de projetos de iniciação científica ou dissertações de mestrado ou doutorado.

Dentre os aspectos que não foram cobertos no escopo desta dissertação, pode-se destacar os seguintes:

- validação da metodologia *WideWorkWeb* junto a times reais distribuídos participando de projetos de desenvolvimento de Aplicações Web;

- avaliação das ferramentas disponíveis no mercado, junto a equipes de desenvolvedores de Aplicações Web envolvidos em projetos globais, para identificar os méritos e as principais deficiências dessas ferramentas;

- especificação e produção de um conjunto integrado de ferramentas de suporte à metodologia WideWorkWeb, que possa ser adotado pelos desenvolvedores.

7.3 Considerações Finais

Poder realizar pesquisas sobre áreas tão interessantes e ricas em detalhes como o Desenvolvimento de Aplicações Web e o Desenvolvimento Global de Software foi, sem dúvidas, uma maravilhosa oportunidade de aprendizado e amadurecimento, principalmente por saber que este trabalho poderá servir como referência para muitas pessoas da comunidade de software, em especial as empresas e pesquisadores brasileiros, que desejem ter uma visão ampla sobre essas duas tendências, e conhecer uma proposta de metodologia de desenvolvimento para uni-las num mesmo projeto.

Para que o acesso a esses resultados não fique restrito à comunidade acadêmica, espera-se que o material reunido nesta pesquisa possa gerar uma publicação de alcance nacional, tornando-se leitura fundamental para todos aqueles que desejem participar de projetos desse tipo.

Anexo A

A. Internacionalização e Localização de Software

Com a transformação de mercados locais em mercados globais, as empresas de software perceberam que para permanecerem competitivas, e tornarem seus produtos atraentes para usuários de todo o mundo, precisariam considerar em seus projetos um público-alvo bastante heterogêneo.

A adaptação do software para diferentes plataformas e idiomas tornou-se uma condição para a entrada no mercado global. Segundo Coronado [CORONADO, 2001], os usuários estão cada vez mais exigentes: eles buscam produtos que atendam suas necessidades, facilitem a realização de suas atividades, e, preferencialmente, sejam apresentados em seus próprios idiomas.

No caso específico de aplicações de *e-commerce*, a satisfação do usuário é um fator ainda mais crítico, uma vez que a competitividade é muito grande e as empresas precisam apresentar diferenciais para conquistar clientes e criar oportunidades de negócio.

Por isso, algumas empresas já começam a adaptar seus processos de desenvolvimento para incluir uma participação maior do usuário, seja na coleta de requisitos ou na realização de testes de usabilidade. É a abordagem do projeto de software centrado no usuário [CLOYD, 2001].

Neste cenário, o maior desafio para as empresas é produzir aplicações que possam ser customizadas de maneira fácil, rápida e barata. Uma solução proposta por Collins [COLLINS, 2002] é tentar isolar o núcleo do software das partes dependentes do usuário durante o desenvolvimento. Assim, a estrutura do software seria semelhante para todos os mercados e apenas os aspectos culturais precisariam ser adaptados.

É exatamente este o objetivo das estratégias de Internacionalização e Localização de Software. Tomando como base as definições de vários autores [LUONG, 1995], [GRIBBONS, 1997], [CORONADO, 2001] e [COLLINS, 2002], essas técnicas podem ser assim conceituadas:

- **Internacionalização**: é a concepção do software como um *framework* culturalmente neutro, que possa ser adaptado para diferentes mercados mais fácil e eficientemente. Para isso, o código deve ser independente de plataforma e idioma, e deve ser preparado para suportar diferentes interfaces e funcionalidades. Todos esses aspectos precisam ser considerados nas fases de projeto e codificação do software.

- **Localização**: é o processo de análise e adaptação do software para atender às necessidades e preferências de mercados distintos, através da tradução do produto, e, em alguns casos, do acréscimo de características específicas do mercado. A Localização se aplica a todos os aspectos do software ligados à comunicação com o usuário, como interface, mensagens do sistema, manuais e tutoriais, e também a aspectos do funcionamento interno da aplicação, como tipos de dados e seqüências de ordenação alfabética.

Apesar de parecerem atividades semelhantes, a Internacionalização se diferencia da Localização por não ter como produto final versões adaptadas do software. A Internacionalização apenas prepara o software para ser localizado, incluindo o código que proporcionará os ajustes para os diferentes contextos.

A Localização pode ocorrer em dois diferentes níveis, dependendo das características do software que serão modificadas para atender os usuários. Quando são considerados apenas os aspectos técnicos do software, a Localização é classificada como Superficial. Se forem feitas também adaptações relativas ao contexto cultural dos usuários, a Localização é dita Cultural.

O objetivo da Localização Superficial é a adaptação funcional do software para um mercado específico. A linguagem utilizada na interface, nas mensagens do sistema e nos manuais deve ser traduzida para o idioma do público-alvo. Nesta atividade, a equipe precisa do conhecimento técnico da terminologia usada no software associado ao conhecimento semântico da linguagem dos usuários para que as traduções sejam coerentes e não causem desentendimentos. Construir um glossário durante o desenvolvimento pode ajudar a identificar termos difíceis de traduzir e que precisem de substituição, além de contribuir para que a consistência dos termos em todo o software seja mantida.

A mudança de idioma ocasiona ainda ajustes nos elementos gráficos da interface. Quando traduzidos, alguns termos podem apresentar até o dobro do tamanho original, exigindo botões e menus maiores. Se essa extensão textual for tratada na Internacionalização, os ajustes podem ser minimizados.

Além disso, os formatos de apresentação de alguns dados variam conforme o país. Datas, horários, telefones e endereços, valores monetários, e números de documentos como passaportes e cartões de crédito são alguns exemplos de dados que precisam ser apresentados de acordo com as convenções locais. Alterações na definição interna de variáveis e bancos de dados podem ser necessárias e devem ser previstas na Internacionalização.

A Localização Cultural busca a satisfação do usuário. A idéia é estudar as tradições e costumes locais para escolher cores, imagens e ícones semanticamente adequados. Collins [COLLINS, 2002] aponta o mercado asiático como um grande desafio para a Localização, uma vez que a cultura oriental apresenta características bastante peculiares que diferem das práticas ocidentais.

As cores e sons, geralmente utilizados para destacar ou transmitir informações aos usuários, podem acabar causando problemas se não forem adaptados. A associação de um som a uma mensagem de erro é vista pelos ocidentais como uma simples advertência, mas pode causar constrangimento em ambientes de trabalho orientais. No caso das cores, as contradições são ainda maiores. Gribbons [GRIBBONS, 1997] ilustra as diferentes interpretações que uma mesma cor pode ter, dependendo da cultura do país, como mostra a Tabela A.1 (Variação nos significados das cores entre países).

	COR				
PAÍS	**VERMELHO**	**AZUL**	**VERDE**	**AMARELO**	**BRANCO**
Estados Unidos	perigo	masculinidade	segurança	covardia	pureza
França	aristocracia	liberdade	criminalidade	temporário	neutralidade
Japão	raiva / perigo	vilão, bandido	futuro / juventude	nobreza	morte

Tabela A.1 – Variação nos significados das cores entre países

A escolha dos ícones também deve ser criteriosa para que seu significado seja o mesmo para todos os usuários. Por exemplo, um carrinho de compras num *site* de comércio eletrônico não será um símbolo intuitivo em países que utilizam sacolas ou cestas para fazerem compras. Neste caso, o uso dos ícones acaba confundindo os usuários.

A organização das telas na interface é outro aspecto que requer ajustes. A existência de diferentes orientações para a leitura e escrita (de cima para baixo e de baixo para cima, da esquerda para a direita e da direita para a esquerda) é considerada na definição da ordem de apresentação das informações. No caso da Localização Superficial de um software americano para o mercado chinês, os termos seriam representados através de símbolos, mas o *layout* da interface não seria modificado. Por isso, a Localização Cultural é mais completa e também mais complexa.

As recomendações de Localização para aplicações tradicionais também podem ser aplicadas para páginas Web, mas alguns aspectos adicionais como performance, consistência na aparência das páginas e estilo de navegação precisam ser tratados. Becker [BECKER, 2001] apresenta vários exemplos de empresas com atuação global e comenta os pontos positivos e negativos dos *sites* de cada uma delas.

A Internacionalização e a Localização podem ser aplicadas de maneira independente. Um produto pode ser preparado para a Internacionalização mesmo que a entrada em outros mercados não esteja prevista em curto prazo. Ou ainda, um produto pode precisar de Localização mesmo que não tenha sido preparado para isso. O certo é que um projeto de software feito para múltiplos mercados pode reduzir o tempo e os gastos necessários para a Localização, que não são poucos. Tomando-se como exemplo a Microsoft, que possui produtos localizados em até 46 línguas e trabalha com a Internacionalização desde a concepção de seus produtos, a estimativa mínima de custos com a Localização para cada produto é de U$ 300.000,00! [COLLINS, 2002]. Vale

salientar que a maioria dos produtos Microsoft sofre apenas uma Localização Superficial, com a padronização dos elementos da interface.

Por isso, a Localização deve ser uma decisão cuidadosa. Os investimentos com a customização do produto dependem de vários fatores, dentre eles o tamanho do mercado que se deseja atingir e o nível esperado de satisfação dos usuários [GRIBBONS, 1997].

Luong [LUONG, 1995] aponta algumas perguntas que precisam ser respondidas pelas empresas antes de optar pela Localização:

- **Vale a pena localizar?** Se a empresa desejar testar uma nova tecnologia, ou lançar um produto inicialmente em um único mercado para verificar sua aceitação, a Localização não é uma alternativa indicada. O investimento necessário com pesquisas de mercado e tradutores especializados seria alto e arriscado, principalmente no caso de empresas que estão iniciando seus negócios. Já para as grandes corporações, a Localização pode ser a chave para a expansão de suas possibilidades comerciais.

- **Quando deve ser feita a Localização do produto?** Dependendo do objetivo da empresa, durante ou após o desenvolvimento. Caso a meta seja o lançamento simultâneo do software em vários mercados, os esforços para a Localização devem ser concentrados durante o desenvolvimento do software, envolvendo um investimento financeiro e de tempo maior, e exigindo também um projeto de Internacionalização desde o começo.

- **Que aspectos do produto serão localizados?** A Localização Superficial é necessária na maioria dos casos. Já a Localização Cultural exige pesquisas e testes de usabilidade nos diferentes mercados e, embora seja mais onerosa, pode produzir melhores resultados por ser capaz de satisfazer os usuários mais plenamente. A decisão sobre o tipo de Localização está associada a vários aspectos, como a demanda do mercado (questionar se a adaptação cultural é realmente imprescindível para a aceitação do produto), o tamanho do mercado (mercados pequenos e restritos podem não compensar os investimentos), e ainda a disponibilidade de recursos da empresa.

- **Quantas versões do software serão desenvolvidas?** No caso da Localização apenas Superficial, uma versão única do software pode ser distribuída para todos os mercados. Para isso, basta que a linguagem e os outros aspectos técnicos possam ser determinados em tempo de execução pelo próprio usuário. Por outro lado, se a estratégia adotada for a Localização completa do software, considerando aspectos técnicos e culturais, costumam ser geradas várias versões do produto, uma para cada mercado.

- **O processo de Localização será terceirizado?** A empresa pode optar pelo envolvimento da própria equipe de desenvolvimento do software no processo de

Localização, ou pela contratação de tradutores ou empresas especializadas para o trabalho. Ambas as alternativas apresentam vantagens e desvantagens. Se por um lado a equipe de desenvolvimento conhece bem o código e não teria dificuldades em modificá-lo, ela dificilmente terá a percepção dos diferentes mercados necessária nesse processo. As empresas que oferecem serviços de Localização têm a experiência e o conhecimento cultural necessário, mas podem onerar os custos, além de serem um fator adicional a ser gerenciado. A terceirização pode ser uma boa alternativa para liberar a equipe de desenvolvimento para outros projetos, mas, conseqüentemente, não será agregado nenhum conhecimento para ser usado em projetos posteriores.

- **Para que mercados o software será localizado?** É preciso decidir se o software será localizado para um mercado específico ou para vários. A empresa deve estudar as possibilidades de negócio nos mercados para descobrir os mais viáveis e promissores. Para *sites* e Aplicações Web, Collins [COLLINS, 2002] afirma que apenas sete línguas (inglês, japonês, alemão, francês, espanhol, português e sueco) são suficientes para atingir 90% dos usuários on-line.

Anexo B

B. Modelos de Artefatos da Metodologia *WideWorkWeb*

O Capítulo 5 (A Metodologia *WideWorkWeb*) apresentou uma proposta de modelo para o desenvolvimento de Aplicações Web Transacionais para mercados heterogêneos por times distribuídos geograficamente e entre empresas. O objetivo deste anexo é ilustrar, através de modelos, alguns dos artefatos que deverão ser produzidos ao longo do desenvolvimento, tornando mais claro o entendimento de cada artefato e sua importância para o contexto dos projetos.

Serão apresentados modelos para seis artefatos importantes: Statement Of Work, análise de riscos, métricas, relatório gerencial, plano de *deployment* e *post-mortem*.

B.1 Statement Of Work

O *Statement Of Work* (SOW) representa um contrato formal que deve ser firmado entre as empresas participantes de um projeto de desenvolvimento de software. Nele são descritas questões como a participação de cada empresa no projeto, o tipo de parceria firmada, os objetivos de mercado, as metas relativas a prazos e custos, e os critérios para a propriedade intelectual dos resultados gerados. A Figura B.1 (Modelo de Artefato – *Statement Of Work*) ilustra como seria um contrato dessa natureza.

[Nome das Empresas]
[Nome do Projeto]
Data:

Statement OF Work

1. Descrição do Projeto

[Uma breve descrição sobre os objetivos do projeto e as principais características da aplicação que será produzida.]

2. Metas de Mercado

[Definição sobre os mercados-alvo pretendidos para a aplicação e indicação sobre a importância de cada mercado-alvo para o negócio. Este item deve conter também decisões sobre o lançamento simultâneo de versões do produto em vários mercados ou a priorização de alguns mercados em detrimento de outros. Também é importante destacar se a adaptação do produto para os diferentes mercados será apenas técnica ou cobrirá também os aspectos culturais.]

3. Empresas Participantes

[Descrição das empresas que atuarão conjuntamente no desenvolvimento, com destaque para as competências de cada uma que poderão ser úteis para o projeto.]

4. Parceria e Responsabilidades

[Indicação do tipo de parceria que será estabelecida pelas empresas (parceria estratégica, *joint venture*, *outsourcing*, etc), e do papel de cada uma durante o desenvolvimento. Destaque também para a descrição dos centros de desenvolvimento que serão envolvidos no projeto sob a responsabilidade de cada empresa.]

5. Composição dos Times

[Quantidade de participantes e competências necessárias em cada centro de desenvolvimento do projeto. Se os times já estiverem definidos na assinatura do contrato, indicar o nome de cada desenvolvedor, a experiência na área de domínio do projeto e a função que exercerá. Caso a composição dos times ainda não tenha sido estabelecida, descrever ao menos os participantes que atuarão como mediadores de cada time e que responderão pelo trabalho de suas equipes.]

Figura B.1 – Modelo de Artefato – *Statement Of Work*

6. Metas de Projeto

[Definição de etapas para o projeto, com cronogramas e uma lista de artefatos a serem produzidos. Deve ser indicado ainda o orçamento pretendido para o projeto.]

7. Propriedade Intelectual

[Definição sobre os direitos de propriedade intelectual dos resultados (código, documentação, metodologias, tecnologias, etc) que serão gerados durante o projeto. Este item define se os direitos autorais sobre esse material serão garantidos conjuntamente a todas as empresas envolvidas, ou apenas àquelas que produziram efetivamente os artefatos. Ainda neste item, devem ser definidos limites para o acesso de participantes de uma equipe a informações ou artefatos pertencentes às demais empresas do projeto. Ou seja, uma espécie de controle para que a privacidade de uma empresa seja preservada durante sua interação com outras empresas, e as informações obtidas não sejam utilizadas de forma indevida após o término do projeto por empresas concorrentes.]

8. Adesão às Normas do Projeto

[Trecho que descreve o comprometimento de cada empresa com o atendimento aos padrões de qualidade aqui definidos, às metas de desenvolvimento, aos cronogramas e aos orçamentos. Ainda neste item, devem ser descritas as punições para o não cumprimento das obrigações das empresas, como o pagamento de multas ou a redução da participação nos lucros do projeto.]

9. Responsabilidades Pós-Desenvolvimento

[Logo no início do projeto, é necessário definir como serão realizadas as atividades após o desenvolvimento (suporte, manutenção, operação do sistema, localização para novos mercados, etc.), indicando qual será a responsabilidade de cada empresa. Por exemplo, é necessário saber se a operação de uma aplicação Web projetada para diferentes mercados será feita em um único centro ou se cada centro será responsável por uma das versões geradas. Essa decisão deve ser tomada o quanto antes, pois, apesar de depender diretamente dos interesses de negócio da empresa, pode afetar também o modo como as equipes serão organizadas para o desenvolvimento.]

Assinatura das Empresas Parceiras _____

Figura B.1 – Modelo de Artefato – Statement Of Work (cont.)

B.2 Análise de Riscos

Este é um documento muito importante que deve ser produzido desde o início do desenvolvimento. O objetivo é identificar riscos potenciais para o projeto que possam afetar cronogramas e orçamentos, Os principais riscos num projeto de desenvolvimento de Aplicações Web para múltiplos mercados por times distribuídos costumam estar associados a atrasos na realização das atividades, mudanças na escolha de tecnologia e ferramentas utilizadas durante o desenvolvimento, e divergências culturais que atrapalham o entendimento dos requisitos e do escopo das atividades de cada equipe.

No caso da metodologia WideWorkWeb, que propõe o Desenvolvimento Baseado em Componentes, os riscos também podem estar associados à produção de um mesmo componente por equipes diferentes, ou ainda à produção de componentes incompatíveis.

A Figura B.2 (Modelo de Artefato – Análise de Riscos) ilustra um exemplo de documento que poderá auxiliar os mediadores na identificação e análise de riscos.

[Nome do Projeto]
[Nome do Mediador que identificou o risco]
[Data da última atualização do documento]

Risco # [Número do risco; pode ser um código]

1. Descrição

   [Indica o tipo de risco. Ex: atraso na entrega de um determinado artefato.]

2. Probabilidade

   [Indica a probabilidade do risco se transformar num problema real. Pode ser representada por um grau de 1 a 10, ou pelos níveis baixa, média e alta.]

3. Impacto

   [Descreve as conseqüências para o projeto caso o risco se transforme num problema real. Ou seja, que atividades seriam afetadas, quais cronogramas e orçamentos teriam que ser refeitos e que participantes teriam ligação direta com essas mudanças.]

4. Ações de Controle

   [Indica ações que podem ser tomadas para evitar que o risco se torne um problema real, ou, pelo menos, para minimizar seu impacto sobre o projeto. Para cada ação de controle proposta, deve ser indicado o participante responsável, a data prevista para avaliar se a ação foi eficaz, e as falhas observadas caso a ação não tenha obtido sucesso.]

Figura B.2 – Modelo de Artefato – Análise de Riscos

B.3 Métricas

Em um projeto de desenvolvimento de software, as métricas podem ser coletadas não apenas para avaliar o progresso das atividades, mas também para servir de referência para estimativas em projetos futuros. Por isso, é possível separar as métricas em duas categorias: métricas de projeto e métricas de acompanhamento.

As métricas de projeto são, na verdade, uma descrição das características principais do projeto em relação aos participantes, à aplicação desenvolvida e aos recursos utilizados. Ou seja, são as métricas que definem o contexto em que o projeto foi realizado, tornando mais precisa a interpretação das métricas de acompanhamento na hora de fazer estimativas para projetos futuros. Essas métricas costumam ser coletadas apenas no final do projeto.

As métricas de acompanhamento representam os parâmetros utilizados pelos gerentes de projeto (ou Mediadores no caso da Metodologia *WideWorkWeb*) para observar o cumprimento dos prazos do projeto e, assim, identificar riscos e falhas no trabalho das equipes. A coleta dessas métricas ocorre quase diariamente em cada equipe de desenvolvimento. Por isso, para ter uma visão geral dos resultados, é necessário observar os valores ao final de cada ciclo de desenvolvimento.

Conforme visto no item 5.1.8 (Métricas para Suporte Executivo), a metodologia WideWorkWeb propõe métricas de acompanhamento baseadas em seis aspectos importantes do desenvolvimento: *user stories*, componentes, mídias, *templates*, casos de teste e erros. As métricas propostas para esses aspectos são todas quantificáveis, tornando mais fácil sua coleta e análise.

A Figura B.3 (Modelo de Artefato – Métricas de Projeto) apresenta um exemplo de formulário a ser preenchido pelos Mediadores ao final do projeto, e a Figura B.4 (Modelo de Artefato – Métricas de Acompanhamento) apresenta um exemplo de formulário a ser preenchido pelos Mediadores ao longo do projeto.

[Nome das Empresas]
[Nome do Projeto]
[Período de Desenvolvimento]

Métricas de Projeto

1. **Sobre os Centros de Desenvolvimento**

[Quantidade de centros de desenvolvimento, localização e quantidade de participantes em cada centro, descrição das semelhanças/diferenças culturais entre os centros]

2. **Sobre os Participantes**

[Para cada centro de desenvolvimento, dados como nome, função exercida, empresa, e competências de cada participante]

3. **Sobre a Aplicação Desenvolvida**

[Descrição sobre o tipo de Aplicação Web desenvolvida (transacional, workflow, informativa, etc), o modelo arquitetural utilizado e os mercados para os quais a aplicação foi adaptada. Pode-se incluir também detalhes técnicos, como quantidade total de componentes de software, mídias, *templates* Web e casos de teste produzidos, para refletir a magnitude da aplicação]

4. **Sobre os Recursos Utilizados**

[Indicação das ferramentas utilizadas em cada centro e das tecnologias escolhidas para implementação da aplicação.]

5. **Sobre a Duração do Projeto**

[Indicação sobre a duração estimada para o projeto e a duração real, com comentários sobre as principais razões para o não cumprimento do prazo, em caso de atraso.]

Figura B.3 – Modelo de Artefato – Métricas de Projeto

[Nome do Projeto]
[Nome do Mediador]
[Data da Última Atualização]
[Indicação do Ciclo de Desenvolvimento]

Métricas de Acompanhamento

User Stories	**Quant. Total**	**Quant. Finalizadas**

Componentes	**Quant. Total**	**Quant. Produzidos**	**Quant. Reusados**

Mídias	**Quant. Total**	**Quant. Produzidas**	**Quant. Reusadas**

Templates	**Quant. Total**	**Quant. Produzidos**	**Quant. Reusados**

Casos de Teste	**Quant. Total**	**Quant. Produzidos**	**Quant. Aprovados**

Erros	**Quant. Encontrados**	**Quant. Corrigidos**

Figura B.4 – Modelo de Artefato – Métricas de Acompanhamento

B.4 Relatório Gerencial

Como os projetos previstos na metodologia *WideWorkWeb* envolvem times distribuídos, é necessário estabelecer alguns mecanismos de controle para acompanhar o progresso das atividades. Embora as métricas descritas no item anterior sejam bons indicadores, não é possível obter maiores detalhes sobre o que realmente ocorre em cada equipe. Ou seja, é possível perceber que existem atrasos, mas não há como saber quais são as causas desses atrasos, nem como avaliar se os atrasos constituem riscos para o projeto.

Para que os avanços de cada time sejam acompanhados de perto, a proposta da metodologia é ter um Mediador em cada centro de desenvolvimento. Além de coletar as métricas de seu time, cada Mediador é responsável também pela realização de reuniões diárias para avaliação de resultados e discussão de problemas. Assim, cada Mediador pode ter uma visão geral das questões que precisam ser resolvidas e agir o quanto antes.

Mas, para que se tenha uma visão sobre o progresso do projeto como um todo, é necessário que os Mediadores troquem informações sobre seus times. Como o uso de meios de comunicação como áudio e vídeo-conferência deve ser evitado para reduzir custos, é desejável que essa comunicação seja feita na forma de relatórios semanais, que indicam o desempenho de cada equipe no período. Assim, cada Mediador tem a oportunidade de analisar como os demais times estão desempenhando suas atividades e como seu time será afetado caso alguma das outras equipes esteja com problemas.

Os relatórios gerenciais podem ser enviados aos Mediadores através de e-mail ou fax. A Figura B.5 (Modelo de Artefato – Relatório Gerencial) apresenta um exemplo de relatório que poderia ser utilizado pelos Mediadores.

[Nome do Projeto]
[Nome do Mediador]
[Período Referente Ao Relatório]

Relatório Gerencial

1. Atividades programadas

[Descrição das atividades que estavam previstas para serem realizadas no período referente ao relatório (Ex: produção de um determinado artefato, execução de testes de interface, etc). Se desejar, o Mediador pode indicar, para cada atividade, o(s) participante(s) responsável(is).]

2. Atividades realizadas

[Indicação das atividades que foram efetivamente realizadas no período. Se desejar, o Mediador pode indicar o tempo consumido, em horas, para cada atividade.]

3. Problemas e soluções

[Neste item, devem ser indicados quaisquer problemas encontrados na realização das atividades, desde dúvidas em relação à implementação de uma determinada funcionalidade, até a dificuldade em utilizar uma nova ferramenta de desenvolvimento. Para cada problema encontrado, deve ser indicada também a solução adotada (ex: conversas com o cliente, treinamento para os desenvolvedores, etc). Além de permitir que os outros Mediadores conheçam a realidade de cada time, esse relato é também uma forma de aprendizado a partir da experiência, uma vez que os Mediadores podem perceber maneiras de solucionar problemas que podem ocorrer em seus próprios times.]

4. Riscos

[O Mediador deve indicar o impacto dos problemas encontrados no período em relação aos riscos do projeto. Por exemplo, se a realização de uma atividade demorou o dobro do tempo estimado, o Mediador deve indicar que existe um risco de que seu cronograma não seja cumprido no prazo esperado. Neste caso, o Mediador deve indicar também que medidas estão sendo tomadas para combater esse risco.]

5. Comentários Gerais

[Este espaço é reservado para quaisquer outras observações que o Mediador deseje incluir, seja uma dica sobre uma nova tecnologia, ou a sugestão de uma técnica de motivação que teve bons resultados em seu time.]

Figura B.5 – Modelo de Artefato – Relatório Gerencial

B.5 Plano de *Deployment*

Uma vez que a aplicação já foi concluída e o cliente já aprovou a versão final, o próximo passo é dar início aos procedimentos para a instalação da aplicação. Ou seja, é preciso identificar que ferramentas precisarão ser instaladas, que arquivos deverão ser copiados, que testes precisam ser realizados, etc. Todas essas informações devem estar contidas no Plano de *Deployment* da aplicação.

É preciso destacar ainda que a implantação ou *deployment* do sistema pode ocorrer em duas circunstâncias principais: quando se tem uma aplicação nova que será posta em operação pela primeira vez, ou quando é necessário atualizar a versão de uma aplicação já em operação.

O primeiro caso é mais simples, pois mesmo que a implantação falhe, o processo pode ser repetido e as falhas observadas podem ser corrigidas. Já no segundo caso, atualizar uma aplicação que está “no ar” significa deixar seus usuários temporariamente sem acesso, o que pode causar insatisfação. Além disso, caso a instalação falhe, não é possível continuar realizando testes no servidor, tornando a aplicação indisponível. É preciso remover a versão nova e restaurar a antiga, ou seja, desfazer todos os procedimentos de instalação e prosseguir com os testes em outra máquina. A lista de ações necessárias para desfazer a implantação de um sistema é denominada *Rollback Plan*.

A Figura B.6 (Modelo de Artefato – Plano de *Deployment*) apresenta um exemplo do Plano de *Deployment* sugerido na metodologia *WideWorkWeb*.

[Nome das Empresas]
[Nome do Projeto]

Plano de *Deployment*

1. *Backup*

[O deployment pode ser necessário tanto em aplicações novas como em atualizações de versões. Este item diz respeito ao segundo caso, e deve ser preenchido com uma lista de todos os arquivos que precisarão ser copiados antes que a instalação da nova versão seja feita. Também devem ser indicados outros procedimentos, como a remoção de ferramentas. Se a aplicação estiver sendo colocada em operação pela primeira vez, não é necessário preencher esse item.]

2. Instalação de ferramentas

[Para cada ferramenta que precise ser instalada (Ex: compilador, SGBD, etc), é preciso indicar onde a ferramenta está disponível (Ex: num disquete, num CD, numa pasta da rede, etc), em que ordem deverá ocorrer a instalação, e quais os parâmetros necessários (Ex: senhas, diretório destino para os arquivos gerados pela aplicação, etc).]

3. Configuração

[Neste item, devem ser indicados os passos necessários para a configuração das máquinas onde a aplicação ficará em operação, do servidor Web, do servidor de aplicação e do browser. Para cada passo, é preciso indicar os parâmetros de configuração, se houver.]

4. Criação de diretórios

[Descrição das características dos diretórios que serão criados para o projeto, como nome do diretório e localização pretendida no sistema de arquivos do computador. Essas informações devem ser organizadas de acordo com a seqüência de criação que o desenvolvedor deverá seguir.]

5. Inclusão de arquivos

[Neste item, devem ser listados os arquivos que deverão ser copiados para a máquina de operação do sistema. Para cada arquivo, é preciso indicar qual a origem (de onde o arquivo será copiado) e qual o destino (em que diretório do sistema o arquivo deverá ser inserido).]

Figura B.6 – Modelo de Artefato – Plano de *Deployment*

6. **Execução de *scripts***

[Pode ser que a instalação da aplicação requeira a execução de alguns *scripts* (conjunto de comandos sequenciais). Neste caso, é preciso indicar que *scripts* deverão ser executados, em que ambiente (ex: dentro de um programa específico) e em que ordem.]

7. Realização de testes

[Uma vez que todos os passos indicados para a instalação da aplicação já tenham sido realizados, é preciso indicar uma seqüência de testes que deverão ser realizados de modo a verificar o funcionamento adequado do sistema. Esses testes podem ser automatizados, na forma de scripts, ou manuais. Em ambos os casos, é preciso indicar neste item do plano onde estão localizados os scripts ou cenários de teste necessários, e a seqüência de execução. Também é preciso indicar para cada teste o resultado esperado (ex: mensagem de confirmação) para que as pessoas envolvidas na instalação possam avaliar o sucesso da operação.]

8. Rollback Plan

[O objetivo deste item é indicar uma lista de ações que deverão ser realizadas caso a instalação falhe, ou para que a operação seja efetuada novamente, se for uma aplicação nova, ou para que a configuração anterior seja restaurada, se for uma nova versão para uma aplicação já existente. A lista deve indicar que ferramentas precisarão ser desinstaladas, que arquivos precisarão ser removidos ou substituídos, que parâmetros deverão ser configurados, e que testes deverão ser realizados após o processo. No caso da instalação de novas versões para uma aplicação já existente, é importante que as ações definidas no primeiro item do plano (*Backup*) tenham sido realizadas para que os dados anteriores não tenham sido perdidos.]

Figura B.6 – Modelo de Artefato – Plano de *Deployment* (cont.)

B.6 *Post-Mortem*

Esse é o documento final do projeto onde devem ser resumidas as experiências dos participantes e a impressão deles e do cliente sobre a aplicação produzida. O principal objetivo é fazer uma reflexão sobre as técnicas adotadas que sirva de aprendizagem para projetos futuros. A documentação dessa reflexão pode beneficiar outras equipes que passem pelos mesmos problemas relatados ou ainda que participem de projetos semelhantes.

A Figura B.7 (Modelo de Artefato – *Post-Mortem*) apresenta um exemplo de *Post-Mortem* que pode ser utilizado em projetos que adotem a metodologia *WideWorkWeb*. Para cada sessão, são indicadas perguntas que podem induzir à reflexão sobre os vários aspectos do projeto.

[Nome das Empresas]
[Nome do Projeto]

Post-Mortem

1. Descrição do projeto

[Uma breve explicação sobre o tipo de aplicação desenvolvida e uma visão geral sobre suas principais funcionalidades.]

2. Aspectos de desenvolvimento

[Este item deve conter uma reflexão sobre as práticas adotadas durante o desenvolvimento. Abaixo são indicados cinco pontos principais que precisam ser considerados, e alguns exemplos de perguntas que podem ajudar na reflexão:

Aspecto 2.1: Técnicas de desenvolvimento

- A Internacionalização do código foi considerada desde o começo? Quais as conseqüências observadas para o projeto?
- Permitir a propriedade coletiva do código ajudou ou dificultou sua manipulação?
- O uso dos padrões de codificação foi adotado por todos os participantes?
- A concepção da aplicação como elementos distintos (código, interface, código) foi eficaz? A integração desses elementos foi complexa?

Aspecto 2.2: Uso de ferramentas

- As ferramentas necessárias estavam disponíveis nos centros?
- Como as ferramentas escolhidas se destacavam das demais disponíveis no mercado (maior confiabilidade, menor preço, melhor interface, melhor adequação aos recursos já disponíveis, etc)?
- Qual o procedimento para a adoção de uma nova ferramenta (tentativa e erro, treinamento dos desenvolvedores, uso de tutoriais, etc)?

Figura B.7 – Modelo de Artefato – *Post-Mortem*

Aspecto 2.3: Definição de *user stories*

- As *user stories* foram bem definidas? Os desenvolvedores sabiam exatamente o que precisava ser feito?
- A seqüência de desenvolvimento das *user stories* facilitou a produção da aplicação? De alguma maneira a implementação de uma *user story* foi afetada por problemas em outra *user story*?
- As *user stories* implementadas conseguiram atender as exigências do cliente?
- Com que freqüência foi necessário definir *user stories* corretivas?

Aspecto 2.4: Alocação de atividades

- A alocação de atividades entre os times permitiu uma maior independência no trabalho das equipes?
- As pessoas alocadas para cada atividade já possuíam alguma experiência na função desempenhada? Elas tinham as competências necessárias?
- Houve necessidade de acúmulo de papéis por parte de alguns desenvolvedores?

Aspecto 2.5: Produção de componentes

- Os componentes definidos eram coesos e fracamente acoplados?
- Os desenvolvedores consideraram as interfaces de comunicação definidas para a comunicação dos componentes no momento de sua concepção, ou houve problemas na hora de integrar o trabalho?
- Algum componente precisou ser desenvolvido duas vezes por ser necessário em user stories que estavam sendo implementadas paralelamente?
- Houve reuso de componentes vindos de outros repositórios, ou mesmo dos componentes produzidos para o projeto?]

3. Aspectos gerenciais

[As práticas adotadas para a gerência do projeto também devem ser discutidas de modo a identificar o que funcionou realmente e o que poderia ser melhorado. Abaixo são indicados cinco pontos principais que precisam ser considerados, e alguns exemplos de perguntas que podem ajudar na reflexão:

Aspecto 3.1: Reuniões de acompanhamento

- As reuniões informais de acompanhamento possibilitaram a detecção de problemas? Ajudaram a motivar o time?
- A freqüência das reuniões precisa ser diária ou poderia ser mais espaçada?
- A duração das reuniões é suficiente para as discussões?
- Foram realizadas reuniões de acompanhamento apenas dentro de cada time, ou houve reuniões (presenciais, com vídeo-conferência) envolvendo todos os participantes do projeto? Qual a opinião dos participantes sobre essas reuniões?

Figura B.7 – Modelo de Artefato – *Post-Mortem* (cont.)

Aspecto 3.2: Coleta de métricas

- As métricas adotadas foram úteis no acompanhamento dos times?
- Alguns riscos puderam ser detectados e combatidos graças à observação das métricas?
- Houve dificuldade para coletar as métricas?
- Houve resistência por parte dos desenvolvedores em adotar as métricas?
- Seria necessário acrescentar ou retirar alguma métrica? Qual? Por quê?

Aspecto 3.3: Comunicação Time-Mediador-Cliente

- A presença de um Mediador dentro de cada time facilitou a integração do time?
- A participação do Mediador como elo entre as equipes foi eficaz? As barreiras culturais e organizacionais foram amenizadas com a atuação do Mediador?
- O Mediador transmitiu com clareza as opiniões do cliente e dos mediadores dos demais times?

Aspecto 3.4: Cumprimento de prazos e orçamentos

- Os prazos e orçamentos estimados para o projeto foram muito diferentes dos prazos e orçamentos reais?
- Os times demonstraram comprometimento em relação às metas do projeto?
- Que circunstâncias provocaram alterações no cronograma/orçamento do projeto? Essas circunstâncias poderiam ter sido previstas?

Aspecto 3.5: Atendimento a critérios de qualidade

- Os critérios de qualidade definidos para a aplicação foram observados desde o começo?
- Houve critérios que não puderam ser atendidos? Por quê?
- Os requisitos não-funcionais da aplicação foram suficientemente testados para garantir sua adequação aos critérios de aceitação do cliente?
- As versões produzidas para diferentes mercados atendiam as características culturais pretendidas?]

4. Aspectos de processo

[Neste item, a reflexão é feita em relação ao processo de desenvolvimento adotado, ou seja, considerando as atividades de desenvolvimento e os artefatos produzidos. É preciso avaliar se a definição das fases foi adequada para o projeto ou se algumas fases precisariam de ajustes, se os artefatos produzidos foram suficientes ou houve artefatos em excesso que atrapalharam a evolução do projeto, se a (in-)formalidade da metodologia foi positiva para o projeto ou se houve dificuldade por parte dos desenvolvedores no entendimento das etapas, etc.]

Figura B.7 – Modelo de Artefato – *Post-Mortem* (cont.)

5. Práticas a serem seguidas

[Neste item, uma lista resumida das práticas que obtiveram bons resultados no projeto e que são recomendadas para projetos futuros (Ex: realização de reuniões de acompanhamento, uso de técnicas de motivação, etc). Para cada prática, é necessário indicar qual o seu propósito e quais as conseqüências de sua adoção.]

6. Práticas a serem evitadas

[Este item deve conter uma lista resumida das práticas que fracassaram no projeto e que devem ser evitadas em projetos futuros. Para cada prática, é necessário indicar qual o seu propósito inicial e por que ela não é eficaz.]

7. Relato de problemas

[Neste ponto do documento deve ser incluído um resumo sobre os principais problemas encontrados durante o projeto e as soluções adotadas para os mesmos. É uma espécie de documentação das experiências das equipes que poderá ser utilizada em projetos posteriores para lidar com problemas semelhantes e antecipar dificuldades.]

8. Opiniões sobre a aplicação

[Este item deve conter uma visão geral sobre as opiniões do cliente e dos próprios desenvolvedores em relação à aplicação desenvolvida, destacando os pontos fortes do produto e também as melhorias que poderiam ser feitas em uma nova versão e os defeitos que ainda precisam ser corrigidos.]

Figura B.7 – Modelo de Artefato – *Post-Mortem* (cont.)

Referências Bibliográficas

[ALLEN, 2002] Allen, P. *Microsoft Update: Crossing Application and Business Boundaries*, Component Development Strategies, Vol. XII, No. 4, April 2002.

[ALMEIDA, 2003] Almeida, R. R., Nunes Filho, R. R. G., Vasconcelos C. R. *Comércio Eletrônico - Guia do Desenvolvedor*, Relatório Técnico DSC/001/03, Departamento de Sistemas e Computação - Universidade Federal de Campina Grande.

[AMBLER, 2002a] Ambler, S. *Agile Modeling*, Wiley Computer Publishing, New York, 2002.

[AMBLER, 2002b] Ambler, S. *All the Right Questions*, Software Development, December 2002, disponível em http://www.sdmagazine.com/documents/s=826/sdm0212f/ (último acesso: Julho/2003)

[AYOAMA, 1998] Aoyama, M. *Web-based Agile Software Development*, IEEE Software, Vol. 15, No. 6, November-December 1998.

[BATTIN, 2001] Battin, R.D., Crocker, R.; Kreidler, J., Subramanian, K. *Leveraging Resources in Global Software Development*, IEEE Software, Vol. 18, No. 2, March-April 2001.

[BECK, 2000] Beck, K. *Extreme Programming Explained: Embrance Change*, Addison-Wesley, May 2000.

[BECKER, 2001] Becker, S.A., Mottay, F.E. *A Global Perspective on Web Site Usability*, IEEE Software, Vol. 18, No. 1, January-February 2001.

[BECKER, 2002] Becker, S.A., Berkemeyer, A. *Rapid Application Design and Testing of Web Usability*, IEEE Multimedia, Vol. 9, No. 4, October-December 2002.

[BENNATAN, 2002] Bennatan, E. M. *What Is Happening to the Global Software Village? Is There Still A Case for Distributed Software Development?*, Cutter Consortium, Executive Report, Vol. 3, No. 1, 2002

[BRERETON, 2000] Brereton, P., Budgen, D. *Component-based Systems: A Classification of Issues*, IEEE Computer, Vol. 33, No. 11, November 2000.

[BURDMAN, 1999] Burdman, J. R. *Collaborative Web Development: Strategies and Best Practices for Web Teams*, Addison-Wesley, Upper Saddle River, NJ 07458.

[CARMEL, 1999] Carmel, E. *Global Software Teams: Collaborating Across Borders and Time Zones*, Prentice Hall, Upper Saddle River, NJ 1999.

[CARMEL, 2001] Carmel, E., Agarwal, R. *Tactical Approaches for Alleviating Distance in Global Software Development*, IEEE Software, Vol. 18, No. 2, March-April 2001.

[CLOYD, 2001] Cloyd, M. H. *Designing User-Centered Web Applications in Web Time*, IEEE Software, Vol. 18, No. 1, January-February 2001.

[COLLINS, 2002] Collins, R. W. *Software Localization for Internet Software: Issues and Methods*, IEEE Software, Vol. 19, No. 2, March-April 2002.

[CONALLEN, 1999] Conallen J. *Modeling Web Application Architectures with UML*, Rational Software, White Paper, disponível em http://www.rational.com/products/whitepapers/100462a.jsp (último acesso: Julho/2003)

[CORONADO, 2001] Coronado, J., Livermore, C. *Going Global with the Product Design Process*, IEEE, December 2001.

[COURTNEY, 2001] Courtney, P.E. *Testing E-Commerce*, ADT Magazine, disponível em http://www.adtmag.com/print.asp?id=3803 (ultimo acesso: Julho/2003)

[DESHPANDE, 2001] Deshpande, Y., Hansen, S. *Web Engineering: Creating a Discipline Among Disciplines*, IEEE Multimedia, Vol. 8, No. 2, April-June 2001.

[DRAPER, 1985] Draper, S.W., Norman, D.A. *Software Engineering for User Interfaces*, IEEE Transactions on Software Engineering, 1985.

[EBERT, 2001] Ebert, C., De Neve, P. *Surviving Global Software Development*, IEEE Software, Vol. 18, No. 2, March-April 2001.

[FAVELA, 2001] Favela, J., Pena-Mora, F. *An Experience in Collaborative Software Engineering Education*, IEEE Software, Vol. 18, No. 2, March-April 2001.

[FOWLER, 2000] Fowler, M. *Put Your Process on a Diet*, Software Development, December 2000, disponível em http://www.sdmagazine.com/articles/2000/0012/ (último acesso: Julho/2003)

[FOWLER, 2001] Fowler, M. *Is Design Dead?*, Software Development, April 2001, disponível em http://www.sdmagazine.com/articles/2001/0104/ (último acesso: Julho/2003)

[GAO, 1999] Gao, J.Z., Chen, C., Toyoshima, Y., Leung, D.K. *Engineering on the Internet for Global Software Production*, IEEE Computer, Vol. 32, No. 5, May 1999.

[GERRARD, 2000a] Gerrard, P. *Risk-Based E-Business Testing: Part 1 – Risks and Test Strategy*, disponível em http://www.evolutif.co.uk/articles/EBTestingPart1.pdf (último acesso: Julho/2003)

[GERRARD, 2000a] Gerrard, P. *Risk-Based E-Business Testing: Part 2 – Test Techniques and Tools*, disponível em http://www.evolutif.co.uk/articles/EBTestingPart2.pdf (último acesso: Julho/2003)

[GINIGE, 2001a] Ginige, A., Murugesan, S. *Web Engineering – An Introduction*, IEEE Multimedia, Vol. 18, No. 1, January-March 2001.

[GINIGE, 2001b] Ginige, A., Murugesan, S. *The Essence of Web Engineering*, IEEE Multimedia, Vol. 8, No. 2, April-June 2001.

[GINIGE, 2001c] Ginige, A., Murugesan, S. *Web Engineering:A Methodology for Developing Scalable, Maintanable Web Applications*, Cutter IT Journal, Vol. 14, No. 7, July 2001.

[GLASS, 2001] Glass, R. L. *Who's Right in the Web Development Debate?*, Cutter IT Journal, Vol. 14, No. 7, July 2001.

[GRIBBONS, 1997] Gribbons, W.M. *Designing for the Global Community*, Professional Communication Conference, 1997. IPCC '97 Proceedings. Crossroads in Communication, 1997 IEEE International , 1997.

[HANDEL, 2000] Handel, M., Wills, G. *TeamPortal: Providing Team Awareness on the Web*, Proceedings of the International Workshop on Awareness and the WWW, held at the ACM CSCW'2000 Conference, December 2000, Philadelphia, PE, disponível em http://www2.mic.atr.co.jp/dept2/awareness/submissionsFinal/410-Handel.pdf (último acesso: Julho/2003)

[HARMON, 1998] Harmon, P. *Component Methodologies*, Component Development Strategies, Vol. VIII, No. 11, November 1998.

[HEEKS, 2001] Heeks, R., Krishna, S., Nicholsen, B., Sahay, S. *Synching or Sinking: Global Software Outsourcing Rrelationships*, IEEE Software, Vol. 18, No. 2, March-April 2001.

[HENDRICKSON, 2000] Hendrickson, E. *Advanced Automation Techniques: Data Driven Testing*, disponível em http://www.qualitytree.com/autotest/advanced.htm (último acesso: Julho/2003)

[HERBSLEB, 2001] Herbsleb, J. D., Moitra, D. *Global Software Development*, IEEE Software, Vol. 18, No. 2, March-April 2001.

[HIGHSMITH, 2002] Highsmith, J. *Extreme Programming*, Cutter Consortium, White Paper, 2002.

[HOHMANN, 2001] Hohmann L., *Managing Small-Team Rapid Web Development*, Cutter IT Journal, Vol. 14, No. 7, July 2001.

[HUTCHENS, 1997] Hutchens, K., Oudshoorn, M., Maciunas, K. *Web-based Software Engineering Process Management*, System Sciences, 1997, Proceedings of the Thirtieth Hawaii International Conference, Vol. 1, 1997.

[JAGIELSKI, 1999] Jagielski, J. *The Apache Success History*, New Architect Magazine, disponível em http://www.newarchitectmag.com/archives/1999/10/jagielski/ (último acesso: Julho/2003)

[KAROLAK, 1998] Karolak, D.W. *Global Software Development*. IEEE Computer Society Press, California, 1998.

[KIRCHER, 2001] Kircher, M., Jain, P., Corsaro, A., and Levine, D. *Distributed Extreme Programming*, XP2001 - eXtreme Programming and Flexible Processes in Software Engineering,Villasimius, Sardinia, Italy, May 21-23, 2001.

[KRUCHTEN, 2000] Kruchten, P. *The Rational Unified Process: An Introduction*, 2nd edition, Addison-Wesley, 2000.

[LEUF, 2001] Leuf, B., Cunningham W. *The Wiki Way – Quick Collaboration on the Web*, Addison-Wesley, Upper Saddle River, March 2001.

[LOWE, 2001] Lowe D., Henderson-Sellers B. *OPEN to Change: Addressing Web Development Process Differences*, Cutter IT Journal, Vol. 14, No. 7, July 2001.

[LUONG, 1995] Luong, T.V., Lok, J.S.H., Taylor, D.J., Driscoll K. *Internationalization-Developing Software for Global Markets*, New York: John Wiley, 1995.

[MACINTOSH, 2000] MacIntosh, A., Strigel, W. *"The Living Creature" – Testing Web Applications*, disponível em http://www.qalabs.com/resources/thelivingcreature.pdf (último acesso: Julho/2003)

[MAR, 2001] Mar, K., Schwaber, K. *Scrum With XP*, disponível em http://www.controlchaos.com/XPKane.htm (último acesso: Julho/2003)

[MAURER, 2002] Maurer, F., Martel, S. *Extreme Programming – Rapid Development for Web-Based Applications*, IEEE Internet Computing Online, January-February 2002.

[MENDES, 2001] Mendes, E., Mosley, N., Counsell, S. *Web Metrics – Estimating Design and Authoring Effort*, IEEE Multimedia, Vol. 8, No. 1, January-March 2001.

[MOCKUS, 2001] Mockus, A., Weiss, D. M. *Globalization by Chunking: A Quantitative Approach*, IEEE Software, Vol. 18, No. 2, March-April 2001.

[MURRU, 2003] Murru, O., Deias, R., Mugheddu, G. *Assessing XP at a European Internet Company*, IEEE Software, Vol. 20 , No. 3, May-June 2003.

[MURUGESAN, 1999] Murugesan, S. *Leverage Global Software Development and Distribution Using the Internet and Web*, Cutter IT Journal, Vol. 12, No. 3, March 1999.

[MURUGESAN, 2001] Murugesan, S., Deshpande, Y. *Web Engineering*, Vol. LNCS 2016, Springer-Verlag, Heidelberg, Germany, April 2001.

[NICHOLS, 2002] Vaughan-Nichols, S. J. *Web Services: Beyond the Hype*, IEEE Computer, Vol. 35, No. 2, February 2002.

[NIELSEN, 2000] Nielsen, J., Norman, D. A. *Usability On The Web Isn't A Luxury*, InformationWeek, February 2000, disponível em http://www.informationweek.com/773/web.htm (útlimo acesso: Julho/2003)

[NOYES, 2002] Noyes, B. *Develop on the Edge with Extreme Programming*, Managing Development, disponível em http://www.fawcette.com/resources/managingdev/methodologies/xp/ (último acesso: Julho/2003)

[OFFUTT, 2002] Offutt, J. *Quality Attributes of Web Software Applications*, IEEE Software, Vol. 19, No. 2, March-April 2002.

[PHILOSOPHE, 2000] Referência eletrônica: http://www.philosophe.com (último acesso: Julho/2003)

[POLLICE, 2001] Pollice, G. *Using the Rational Unified Process for Small Projects: Expanding Upon Extreme Programming*, Rational Software, White Paper, disponível em http://www.rational.com/products/whitepapers/tp183.jsp (último acesso: Julho/2003)

[POWELL, 2000] Powell, T. *Web Design: The Complete Reference*, Osborne McGraw-Hill, Berkeley, California, 2000.

[PRESSMAN, 1995] Pressman, R. S. *Engenharia de Software*, 3a. Edição, Makron Books, São Paulo, 1995.

[PRESSMAN, 1998] Pressman, R. *Roundtable: Can Internet-Based Applications Be Engineered?*, IEEE Software, Vol. 15, No. 5, September-October 1998.

[PRESSMAN, 2000] Pressman, R. *What a Tangled Web We Weave*, IEEE Software, Vol. 17 No. 1, January-February 2000.

[PRESSMAN, 2001] Pressman, R. *Introduction – Web Engineering: An Adult's Guide to Developing Internet-Based Applications*, Cutter IT Journal, Vol. 14, No. 7, July 2001.

[RAJLICH, 2000] Rajlich, V.T., Bennett, K.H. *A Staged Model for the Software Life Cycle*, IEEE Computer, Vol. 33, No. 7, July 2000.

[RAMLER, 2002] Ramler, R., Schwinger, W., Altmann, J. *Testing Web Quality Aspects*, Software Competence Center Hagenberg, Technical Report SCCH-TR-0216, March 2002, disponível em http://www.schwinger.at/LIB/SCCHTR0216.doc (último acesso: Julho/2003)

[REPENNING, 2001] Repenning, A., Ioannidou, A., Payton, M., Wenming Ye, Roschelle, J. *Using Components for Rapid Distributed Software Development*, IEEE Software, Vol. 18, No. 2, March-April 2001.

[RICE, 2002] Rice, R.W. *How to Develop Test Cases and Test Scripts for Testing Web Applications*, disponível em http://www.riceconsulting.com/new/index.php?option=articles&task=viewarticle&artid=10 (último acesso: Julho/2003)

[RISING, 2000] Rising, L., Janoff, N. S. *The Scrum Software Development Process for Small Teams*, IEEE Software, Vol. 17, No. 4, July-August 2000.

[SCHWABER, 1995] Schwaber, K. *SCRUM Development Process*, disponível em http://jeffsutherland.com/oopsla/schwapub.pdf (último acesso: Julho/2003)

[SLIWA, 2002] Sliwa, C. *XP, Scrum Join Forces*, Computerworld, March 2002, disponível em http://www.computerworld.com/softwaretopics/software/appdev/story/0,10801,69183,00.html (último acesso: Julho/2003)

[SOFTWAREQA, 2001] Referência eletrônica: http://www.softwareqatest.com/qatweb1.html (último acesso: Julho/2003)

[STAL, 2002] Stal, M. *Web Services: Beyond Component-Based Computing*, Communications of the ACM, Vol. 45, No. 10, October 2002.

[STOUT, 2001a] Stout, G. A. *Testing A Website – Best Practices*, The Revere Group, disponível em http://www.reveregroup.com/138_006-RevereThoughtLeadership.pdf (último acesso: Julho/2003)

[STOUT, 2001b] Stout, G. A. *Deploying A Website – Best Practices*, The Revere Group, disponível em http://www.reveregroup.com/138_007-RevereThoughtLeadership.pdf
(último acesso: Julho/2003)

[TAKAHASHI, 2000] Takahashi, K., Yana, E. *A Hypermedia Environment for Global Collaboration*, IEEE Multimedia, Vol. 7, No. 4, October-December 2000.

[THOMSETT, 1998] Thomsett, R. *Outsourcing: The Great Debate*, Cutter IT Journal, Vol. 11, No. 7, July 1998.

[X/OPEN, 1994] X/Open Guide, *Internationalisation Guide*, Version 2, X/Open Company Limited, U.K.

[ZETTEL, 2001] Zettel, J., Maurer, F., Münch, J., Wong, L. *LIPE: A Lightweight Process for E-Business Startup Companies Based on Extreme Programming*, Proceedings of the 3rd International Conference on Product Focused Software Process Improvement (PROFES 2001), September 10-13, 2001, Kaiserslautern, Germany, disponível em http://sern.ucalgary.ca/~milos/papers/2001/ZettelEtAl2001.pdf
(último acesso: Julho/2003)