Metodologias Ágeis
Engenharia de Software sob Medida

Metodologias Ágeis
Engenharia de Software sob Medida

José Henrique Teixeira de Carvalho Sbrocco
Paulo Cesar de Macedo

Metodologias Ágeis

Engenharia de Software sob Medida

1ª Edição

EDITORA ÉRICA

www.editoraerica.com.br

Dados Internacionais de Catalogação na Publicação (CIP)
(Câmara Brasileira do Livro, SP, Brasil)

Sbrocco, José Henrique Teixeira de Carvalho

Metodologias ágeis: engenharia de software sob medida / José Henrique Teixeira de Carvalho Sbrocco, Paulo Cesar de Macedo. -- 1. ed. -- São Paulo: Érica, 2012.

Bibliografi a.
ISBN 978-85-365-0979-2

1. Engenharia de software I Título.

12-04652 CDD-005.1

Índice para catálogo sistemático:
1. Engenharia de Software 005.1

Coordenação Editorial:	Rosana Arruda da Silva
Capa:	Maurício S. de França
	Rosemeire Mendes Cavalheiro
Editoração e Finalização:	Graziele Karina Liborni
	Carla de Oliveira Morais
	Rosana Aparecida Alves dos Santos
Avaliação Técnica:	Adilson da Silva Lima

Seu cadastro é muito importante para nós

Ao preencher e remeter a ficha de cadastro constante no site da Editora Érica, você passará a receber informações sobre nossos lançamentos em sua área de preferência.

Conhecendo melhor os leitores e suas preferências, vamos produzir títulos que atendam suas necessidades.

Contato com o editorial: editorial@editoraerica.com.br

Editora Érica Ltda. | Uma Empresa do Grupo Saraiva
Rua São Gil, 159 - Tatuapé
CEP: 03401-030 - São Paulo - SP
Fone: (11) 2295-3066 - Fax: (11) 2097-4060
www.editoraerica.com.br

Dedicatória

Aos meus pais José Henrique (*in memoriam*) e Maria José, cujos exemplos e cultura me guiam e alimentam minha curiosidade sobre o mundo;

Aos meus filhos Mariana, Lucas e Gabriel pela inspiração e aprendizado recebidos, muitas vezes maiores do que eu pude proporcionar;

À minha neta Lara; que ao nascer faça de sua pequena visão de mundo uma inspiração para grandes realizações, seguindo sempre um caminho repleto de luz e paz;

À minha esposa Silvana pelo incentivo, dedicação à família e pela oportunidade de experimentar a vida acompanhada de amor.

José Henrique T. C. Sbrocco

Aos meus pais pela educação que me deram ao longo da vida;

À minha esposa Rita pelo amor, confiança, carinho e incentivo para a realização desta obra;

À minha filha Maria Luísa, por ter se tornado a nova fonte de felicidade em minha vida;

Aos meus irmãos Fábio (*in memoriam*), Flávio e Gerson por me apoiarem e acreditarem em mim em todos os momentos.

Paulo Cesar de Macedo

"Dá força ao cansado, e ao desfalecido renova o vigor".
Isaías 40:29

Agradecimentos

À Editora Érica pela oportunidade e confiança depositada neste novo trabalho;

Ao professor Francisco Bianchi, coordenador do curso de Gestão da Tecnologia da Informação, da Faculdade de Tecnologia (FATEC) de Itu, que nos honrou com o prefácio desta obra;

A todos os professores e funcionários da FATEC de Itu, que são parte desta família motivadora e vencedora;

Aos nossos companheiros do Núcleo de Inovação da FATEC Itu e da Associação de Pesquisa, Ensino e Desenvolvimento (ASPED), pelo incentivo e inspiração recebidos durante a realização deste trabalho;

A todos os alunos das instituições nas quais ministramos nossas aulas (FATEC Itu, FATEC Mogi Mirim, Faculdade Santa Lúcia e UNIMEP Piracicaba).

Sumário

Prefácio

A partir da década de 1960, os computadores, até então criados para fins militares e científicos, começaram a ser utilizados comercialmente, ou seja, foram descobertos por empresas industriais e comerciais que viram no uso dessa ferramenta, entre outras, uma forma de eliminar esforços repetitivos de seus colaboradores, por meio do processamento de seus dados usando sistemas computacionais.

Surge, nesse momento, uma nova e importante profissão que é a de Analista de Sistemas. Edward Yourdon, analista de sistemas veterano da década de 1970, escreveu em seu livro intitulado "*Análise Estruturada Moderna*" (Editora Campus, 3 ed., 1990, p.2) que, "na realidade, a análise de sistemas é mais interessante que tudo que conheço, excetuando, talvez, sexo e certos tipos de vinhos da Austrália". Essa visão de Yourdon pode ser considerada atual, pois até hoje essa interessante profissão seduz e atrai inúmeros jovens que, semestral ou anualmente, procuram na área da Ciência da Computação os cursos de graduação, principalmente em Análise e Desenvolvimento de Sistemas e em Sistemas de Informação, como uma forma de inserção no mundo da Tecnologia da Informação.

No Brasil, no início da década de 1970, tínhamos menos que 800 computadores utilizados comercialmente, em sua maior parte por empresas de grande porte. Essas empresas possuíam grandes Centros de Processamentos de Dados, os denominados CPDs, onde analistas de sistemas e programadores de computadores, dentre outros profissionais, interagiam na construção de sistemas computacionais, na maioria das vezes complexos e com grande número de programas.

Nesse ambiente, um dos grandes problemas encontrados pelos Analistas de Sistemas era a falta de metodologias, ou mesmo métodos, padronizadas que auxiliassem no desenvolvimento de projetos de sistemas. Essa deficiência acarretava inúmeros problemas relacionados, principalmente, com a implantação e manutenção do sistema. Apenas para ilustrar, alguns desses problemas foram: a baixa aderência do sistema, que não permitia seu compartilhamento às vezes entre empresas do mesmo grupo; a dificuldade em documentar adequadamente o sistema, problema este sentido no momento da manutenção desse sistema, geralmente feita por analistas de sistemas e programadores que não participaram do projeto de construção; a falta de padrões para determinação da qualidade do sistema; e, principalmente, o tempo e os custos excessivos despendidos na sua construção que, geralmente, levava anos para ser concluída e consumia volumosos investimentos.

Entre o final da década de 1970 e início da década de 1980, analistas de sistemas veteranos, como Edward Yourdon, Tom DeMarco, Chris Gane, Trish Sarson, entre outros, vivenciando essa dificuldade em desenvolver sistemas sem uma metodologia adequada, passaram a observar, pesquisar e testar novas metodologias aplicadas ao desenvolvimento de projetos de sistemas. Dos estudos e pesquisas destes e de outros profissionais da análise de sistemas resultaram as metodologias aplicadas ao desenvolvimento de sistemas, desde sua forma clássica até a atual, orientada a objetos.

Atualmente, considerando a iniciativa desses profissionais-autores e os avanços das tecnologias da informação, surgem novas metodologias de desenvolvimento de sistemas que, cada vez mais, vêm sendo aperfeiçoadas e adequadas à velocidade do desenvolvimento de projetos de sistemas, tão exigida pelo mercado de desenvolvimento de sistemas. É nesse contexto que esta obra se insere. Está voltada, principalmente, para profissionais que acompanham a velocidade do mundo da tecnologia da informação.

A notória experiência dos autores, tanto no desenvolvimento de sistemas como no gerenciamento de projetos, é compartilhada nesta importante obra, na área da Engenharia de Software, que proporciona ao leitor (alunos em curso ou mesmo egressos dos cursos já citados e, até mesmo, profissionais da área de desenvolvimento de sistemas não oriundos desses cursos) um importante momento para obter informação, gerar uma porção de conhecimento que, agregando a um conhecimento anterior, irá gerar conhecimento novo.

A forma como o livro está organizado, e escrito em linguagem direta com o leitor, propicia uma leitura dinâmica, envolvente e participativa, permitindo, ao final de cada capítulo, o exercício do aprendizado. Os exercícios propostos no final de cada capítulo estão apresentados de uma forma que induz o leitor a percorrer, mesmo que mentalmente, o que foi aprendido em outras disciplinas, não somente as correlacionadas com a área da Ciência da Computação, mas também com as de outras ciências, como as áreas da Administração e Gestão, integrantes das Ciências Humanas.

"Todos nós somos alimentados e abrigados pelo trabalho de outros homens e devemos pagar honestamente por ele, não apenas com o trabalho escolhido para nossa satisfação íntima, mas com o trabalho que, segundo a opinião geral, os sirva." (Albert Einstein: O Lado Humano. Rápidas Visões Colhidas de seus Arquivos, de Helen Dukas e Banesh Hoffmann; tradução de Lucy de Lima Coimbra, Editora Universidade de Brasília, 1984, pág. 48).

Considerando isso, podemos concluir que este livro é de grande valia e serventia para alunos, profissionais e até mesmo usuários de Sistemas de Informação, principalmente os baseados em computadores. Finalmente, acredito que, se os profissionais envolvidos com projetos de desenvolvimento de sistemas usarem as abordagens descritas pelos autores neste livro, certamente serão capazes de desenvolver, em curto espaço de tempo, sistemas de informação altamente eficazes e confiáveis.

Prof. Me. Francisco Bianchi

Coordenador do Curso Superior de Tecnologia em Gestão da Tecnologia da Informação,
da Faculdade de Tecnologia de Itu - "Dom Amaury Castanho".
Mestre em Informática pela Pontifícia Universidade Católica de Campinas.
Mestre em Ciência da Informação pela Pontifícia Universidade Católica de Campinas.

Apresentação

Podemos dizer que o grande problema enfrentado no desenvolvimento de software pode ser resumido da seguinte maneira: queremos desenvolver um software rapidamente, que seja flexível e que tenha o menor custo possível. Essa motivação sempre fez com que diversas linhas de ações fossem estudadas e aplicadas ao desenvolvimento de software, objetivando resolver tais problemas pelo menos parcialmente. Estamos nos referindo a ações que abrangem desde a investigação de novas técnicas de desenvolvimento, passando por estudos que visam melhorar a compreensão de aspectos fundamentais envolvidos no processo de desenvolvimento de software, além de revisões dos modelos clássicos de ciclo de vida de desenvolvimento, entre outros esforços. Desde os primórdios da computação, tais esforços acabaram seguindo caminhos isolados, muitas vezes até conflitantes. Contudo, nos últimos anos, novos paradigmas têm surgido e obtido cada vez mais adeptos: são as chamadas metodologias de desenvolvimento ágeis.

Sabemos que o desenvolvimento de software *ad hoc*, ou seja, que não segue nenhuma metodologia específica, em geral produz resultados insatisfatórios, principalmente se considerarmos sistemas de software complexos. Por outro lado, modelos de engenharia de software tradicionais podem não se mostrar adequados em todos os casos, uma vez que sua filosofia pode contribuir para "engessar" projetos que não precisam, na prática, de todo o formalismo solicitado.

A partir da década de 1980, a necessidade de alternativas metodológicas começou a incomodar as organizações que trabalhavam com projetos. Surgiram então diversos novos modelos de desenvolvimento de projetos, que incluíam também a elaboração de software, com a particularidade de serem considerados ágeis. Foi a origem das chamadas "Metodologias de Desenvolvimento Ágeis", que objetivam minimizar o risco dos projetos pela proposta de desenvolvimento em curtos períodos de tempo, conhecidos como iterações. Conforme teremos a oportunidade de estudar nesta obra, cada iteração é considerada um "miniprojeto", uma vez que inclui todas as tarefas necessárias para implementar a parte que está sendo desenvolvida, como planejamento, análise de requisitos, projeto, codificação, teste e documentação.

Outra importante característica das metodologias de desenvolvimento ágeis é que elas priorizam comunicações em tempo real a documentos escritos, produzindo, consequentemente, pouca documentação quando comparadas a outros métodos. Embora os métodos ágeis apresentem diferenças entre suas práticas, eles compartilham inúmeras características em comum, as quais foram estudadas e formalizadas em 2001, quando um grupo de especialistas em desenvolvimento de software estabeleceu princípios comuns que eram compartilhadas por várias metodologias. O resultado foi a criação de um documento conhecido como "manifesto ágil", que apresentou considerações importantes e inovadoras, conforme ilustra a tabela a seguir.

Considerações do manifesto ágil

Pessoas e interações	ao contrário de processos e ferramentas
Software executável	ao contrário de documentação extensa e confusa
Colaboração do cliente	ao contrário de constantes negociações de contratos
Respostas rápidas para mudanças	ao contrário de seguir planos previamente definidos

Observa-se, na tabela, que o fato de haver uma maior preocupação com a coluna da esquerda, não significa que os elementos da coluna da direita são desconsiderados. Apesar de podermos utilizar muitos dos paradigmas considerados ágeis para qualquer tipo de projeto, o foco principal desta obra está nos projetos de software.

Como será possível analisar ao estudar tais propostas de processos, as metodologias de desenvolvimento de software apresentam vantagens e desvantagens, limitações e restrições. Contudo, percebe-se, por meio de resultados empíricos, que o uso de metodologias ágeis apresenta excelentes resultados em termos de qualidade, confiança, prazo de entrega e custo. Certamente mais estudos complementares são necessários, considerando projetos com características diferentes, equipes específicas etc. Mesmo assim, podemos perceber atualmente que a adoção de metodologias ágeis tem sido uma tendência mundial.

Tendo em vista o crescente interesse pelo uso de metodologias ágeis, esta obra tem por objetivo descrever seus principais paradigmas, considerando uma análise comparativa e as diferenças entre metodologias ágeis e tradicionais de desenvolvimento de software. Para isso, traz um estudo completo sobre as características e aplicações das metodologias ágeis, uma vez que essa nova tendência tem se mostrado uma escolha viável para atender aos requisitos necessários da atual realidade de mercado.

Inicialmente são apresentados conceitos teóricos importantes, como processos e projetos, seguidos de tópicos relacionados com a engenharia de software, os quais precisam ser entendidos independentemente do paradigma metodológico utilizado. Objetivando uma base para comparações, características das metodologias de desenvolvimento de software tradicionais também são abordadas, além de considerações iniciais sobre as metodologias ágeis. Os capítulos de 6 a 12 descrevem as principais metodologias ágeis utilizadas atualmente (FDD, DSDM, ASD, Crystal, XP, SCRUM, Iconix). Após a exposição de suas principais características, um capítulo complementar apresenta uma visão comparativa entre as metodologias tradicionais e as consideradas ágeis. Essa comparação é utilizada como pré-requisito para o entendimento dos critérios que ajudam na escolha da metodologia ágil que melhor se ajusta às necessidades de determinado projeto, sendo descritos na sequência.

A obra também apresenta uma proposta de aplicação de metodologia ágil adaptada para o cenário acadêmico, bem como considerações complementares sobre gestão de projetos, importantes de serem entendidas independentemente da metodologia ou característica do projeto. Por fim, outras possibilidades de uso de metodologias ágeis também são citadas, seguidas de considerações finais sobre o tema. Todos os capítulos trazem uma lista de exercícios para ajudar na fixação do conteúdo teórico.

Nota:

Os sites mencionados estavam disponíveis até a data de publicação deste livro.

Sobre os autores

José Henrique Teixeira de Carvalho Sbrocco

Construiu sua experiência profissional ao longo de mais de 28 anos desenvolvendo projetos de sistemas informatizados em universidades, centros de pesquisa, empresas de consultoria e tecnologia da informação. Possui sólido embasamento conceitual através de MBA Executivo (gestão empresarial), além de ser especialista em sistemas de computação (ênfase em inteligência artificial), redes e banco de dados. Também realizou diversos cursos técnicos durante sua experiência como analista de sistemas e gestor de projetos.

Atuou como analista de mercado internacional, responsável pelo gerenciamento de negócios envolvendo prospecção mercadológica, qualificação de oportunidades, elaboração de propostas comerciais, elaboração e revisão de contratos, apresentações corporativas e de soluções tecnológicas para países da América Latina, África e Oceania, realizando visitas a clientes, negociações de contratos, acompanhamento de projetos e treinamentos.

Iniciou sua carreira como docente de cursos de graduação em Computação e Administração há 16 anos. É conferencista em eventos empresariais e universidades com temas relacionados à computação, gestão organizacional e desenvolvimento de recursos humanos. Escritor, publicou em 2011 o livro "UML 2.3: Teoria e Prática", pela Editora Érica. Atualmente é professor universitário, coordenador do escritório de projetos e vice-coordenador do Núcleo de Informática Aplicada da Faculdade de Tecnologia de Itu (FATEC), além de presidente da Associação de Pesquisa, Ensino e Desenvolvimento (ASPED), entidade dedicada ao desenvolvimento de soluções tecnológicas inovadoras e à promoção de pesquisas.

Paulo Cesar de Macedo

É doutorando em Computação, Mestre em Ciências da Computação, pós-graduado em Sistemas de Informação e especialista em Redes de Computadores. Possui experiência de 15 anos em empresas de tecnologia e docência no ensino técnico e superior. Possui graduação em Pedagogia e licenciatura para atuar no ensino profissionalizante.

Possui sólidos conhecimentos em engenharia de software, participando como orientador de projetos, trabalhos de conclusão e iniciação científica.

Em sua carreira profissional atuou como consultor de infraestrutura de redes de computadores; possui certificações CCNA e FURUKAWA.

Atualmente leciona nos cursos de graduação em Gestão e Sistemas de Informação e pós-graduação em Engenharia de Software.

Sobre o material disponível na Internet

O material disponível no site da Editora Érica contém PDF com as respostas dos exercícios.

Para utilizar o arquivo, é preciso ter instalado em sua máquina o Acrobat Reader versão 8 ou versão mais recente.

respostas.exe - 467KB

Procedimento para Download

Acesse o site da Editora Érica: www.editoraerica.com.br. A transferência do arquivo disponível pode ser feita de duas formas:

- **Por meio do módulo de pesquisa.** Localize o livro desejado, digitando palavras-chave (nome do livro ou do autor). Aparecem os dados do livro e o arquivo para download. Com um clique o arquivo executável é transferido.
- **Por meio do botão "Download".** Na página principal do site, clique no item "Download". É exibido um campo no qual devem ser digitadas palavras-chave (nome do livro ou do autor). Aparecem o nome do livro e o arquivo para download. Com um clique o arquivo executável é transferido.

Procedimento para Descompactação

Primeiro passo: após ter transferido o arquivo para sua máquina, verifique o diretório em que se encontra e dê um duplo clique nele. Aparece uma tela do programa WinZip Self-Extractor que conduz ao processo de descompactação. Abaixo do Unzip to Folder há um campo que indica o destino do arquivo que será copiado para o disco rígido do seu computador.

C:\Metodologias Ágeis

Segundo passo: prossiga a instalação, clicando no botão Unzip, o qual se encarrega de descompactar o arquivo. Logo abaixo dessa tela aparece a barra de status que monitora o processo para que você acompanhe. Após o término, outra tela de informação surge, indicando que o arquivo foi descompactado com sucesso e está no diretório indicado. Para sair dessa tela, clique no botão OK. Para finalizar o programa WinZip Self-Extractor, clique no botão Close.

1

Introdução

Objetivos

- *Definir o público-alvo da obra, bem como a maneira como ela está organizada.*
- *Contextualizar o leitor sobre as necessidades básicas de uma organização.*
- *Apresentar o conceito de processo, enfatizando os processos de negócio.*

1.1 Considerações iniciais

Nos últimos dez anos as metodologias ágeis têm surgido motivadas pela necessidade de buscar alternativas para os modelos tradicionais de desenvolvimento de projetos. Essa necessidade se justifica quando, por exemplo, trabalhamos com projetos cujo escopo e tempo são reduzidos. Neste caso, se aplicarmos técnicas de engenharia tradicionais, podemos observar que elas possuem grande ênfase em projetar antes de construir. Por isso o ideal seria utilizar uma metodologia de desenvolvimento que permita alterar constantemente o projeto, sem comprometer a qualidade. Em outras palavras, utilizar metodologias tradicionais em projetos de curta duração poderia atrasar sua concepção, trazendo prejuízos ao cliente. As metodologias ágeis visam, entre outras coisas, proporcionar que o cliente tire proveito da aplicação o quanto antes, fazendo com que receba constantemente partes do software na medida em que são concluídas.

1.2 Público-alvo

Com base na pertinência do estudo desses novos paradigmas, esta obra é de relevante interesse não apenas para estudantes de cursos de graduação ou especialização, relacionados à área de computação e administração, mas principalmente para gestores de projetos. É uma leitura indispensável para compreender os propósitos das metodologias ágeis e critérios comparativos com outras metodologias, auxiliando na escolha da que melhor se encaixa com as necessidades de cada projeto.

1.3 Necessidades das organizações

Vamos considerar inicialmente a seguinte constatação: uma organização é composta por um conjunto de colaboradores e também de outros recursos, com o único objetivo de cumprir metas. Sabemos que existem basicamente dois tipos de organizações: aquelas que buscam lucros (com fins lucrativos) e as sem fins lucrativos. A primeira meta de uma organização com fins lucrativos é maximizar os lucros por meio do aumento das receitas ou pela redução dos custos. Já nas organizações sem fins lucrativos, que incluem grupos sociais, religiosos, universidades públicas, entre outras, o lucro não representa sua meta primária. Em um cenário organizacional, dinheiro, pessoas, materiais, máquinas, equipamentos, dados, informações, decisões estão constantemente sendo utilizados, fazendo com que a organização possa ser encarada como um sistema. Desta forma, recursos como materiais, pessoas ou dinheiro, provenientes do ambiente no qual a empresa está inserida, constituem entradas para o sistema organizacional. Essas entradas passam através de um mecanismo de transformação para depois retornarem ao ambiente sob a forma de saídas, que compreendem, geralmente, produtos ou serviços com valor relativo maior do que as entradas sozinhas. Assim, por meio dessa diferença de valor ou de importância, as organizações tentam constantemente alcançar suas metas.

Podendo a empresa ser definida como um sistema, dentro dos mecanismos de transformação podemos encontrar vários subsistemas que contêm processos que ajudam a tornar entradas específicas em produtos ou serviços de maior valor. Os processos utilizados pelas organizações devem ser implantados para explorar oportunidades e resolver problemas, acrescentando valor às partes interessadas (cliente, fornecedor, diretor, empregado etc.). Michel Potter descreveu um conceito muito importante, em 1985, em um artigo da *Harvard Business Review*. Trata-se do conceito da cadeia de valor, o qual esclarece como as organizações podem agregar valor aos seus produtos e serviços, correspondendo, basicamente, a uma cadeia de atividades que inclui logística de recebimento, depósito e armazenamento, produção, estocagem de produtos acabados, logística de entrega, marketing, vendas, serviços de atendimento etc. A ideia é que cada uma dessas atividades seja investigada para que possamos determinar o que fazer para aumentar o valor percebido por um cliente. E esse valor não está só relacionado a valores financeiros, pois dependendo do cliente, esse valor percebido pode significar menor preço, melhor serviço, maior qualidade, entre outros fatores, que certamente irão assegurar o sucesso organizacional.

Considerando os pontos de vista apresentados anteriormente, podemos então fazer a seguinte pergunta: qual seria o papel desempenhado por um sistema de informação nesses processos que agregam valor? A visão tradicional dos sistemas de informação sustenta que as organizações os empregam com o objetivo de monitorar e controlar os seus processos e, consequentemente, as-

segurar a eficácia e a eficiência de suas operações. Como exemplo, podemos dizer que um sistema de informação poderia transformar o *feedback* dos subsistemas de processo em uma informação mais significativa, que poderia ser usada por algum colaborador dentro da organização. Atualmente, uma visão mais contemporânea sustenta que os sistemas de informação estão tão frequentemente ligados ao processo de adição de valor que melhor seria considerá-los como parte do próprio processo, uma vez que na visão tradicional os sistemas de informação eram considerados externos ao processo, servindo apenas para monitorá-lo e controlá-lo.

Assim, a forma pela qual uma organização percebe o papel dos sistemas de informação vai influenciar no modo como ela finaliza seus processos que adicionam valor. Outro aspecto que não podemos nos esquecer de mencionar diz respeito à globalização, que acabou criando oportunidades de usar sistemas de informação para coordenar o trabalho de partes diferentes da empresa e também para estabelecer a comunicação com clientes e fornecedores. Portanto, antes de iniciarmos o grande desafio que é desenvolver software para organizações, devemos refletir sobre suas necessidades, para que possamos caracterizar bem o problema a ser resolvido e escolher a melhor abordagem de desenvolvimento para cada caso.

1.4 Processos

Quando falamos de desenvolvimento de software, inevitavelmente temos de nos remeter a conceitos existentes em uma disciplina mais abrangente, denominada engenharia de software, cujos conceitos básicos serão apresentados no capítulo 3. Por ora, gostaríamos de enfatizar que o fundamento da engenharia de software é sua camada de processo, que representa uma verdadeira "cola" que mantém unidas as camadas de tecnologia, permitindo o desenvolvimento racionalizado de software.

Os processos dessa camada definem uma estrutura para ser utilizada por um conjunto de áreas-chave, ou áreas de processo, estrutura esta estabelecida para a efetiva utilização da tecnologia da engenharia de software. Essas áreas de processo disponibilizam uma base para que possamos ter um controle gerencial de projetos de software. Estabelecem também o contexto no qual os métodos técnicos serão aplicados, os produtos de trabalho (documentos, modelos, relatórios, formulários, dados etc.) que serão feitos, que marcos devem ser estabelecidos, entre outras preocupações. Com a adoção dessa estrutura a qualidade do software pode ser assegurada, permitindo, inclusive, que eventuais modificações ao longo do ciclo de vida do projeto sejam adequadamente geridas.

Um processo de software normalmente possui uma estrutura comum, definida com um pequeno número de atividades que são aplicáveis a todos os projetos de software, independente de seu tamanho ou complexidade. Dentro dessa estrutura comum, existem conjuntos de tarefas de engenharia de software que permitem que as atividades previstas na estrutura comum sejam aplicadas e adaptadas às características do projeto de software e também às necessidades da equipe de projeto envolvida. Além dessas atividades adaptáveis, existem aquelas que cobrem todo o modelo de processo (denominados processos "guarda-chuva"), que determinam, basicamente, como garantir a qualidade do software, como será feita a gestão de configuração, além de definir que medições serão necessárias. Observe que essas atividades "guarda-chuva" são independentes de qualquer atividade, ocorrendo ao longo de todo o processo de desenvolvimento do software. Não podemos, portanto, pensar nas necessidades de uma organização sem pensar em seus processos de negócio.

1.4.1 Conceitos básicos

A seguir apresentamos conceitos e exposições sobre os pontos mais relevantes relacionados aos processos.

Processo e sistema: esses conceitos são fundamentais para a boa compreensão da administração das organizações e para o gerenciamento em geral. Todas as atividades técnicas, gerenciais e administrativas podem ser estudadas sob a forma de processos. Mas o projeto, com as partes construtivas, a equipe que o executa, a organização que o hospeda e o ambiente que o cerca, também é visto sob o enfoque sistêmico.

Processo: qualquer trabalho, operação administrativa, função biológica, produtiva, social etc. pode ser considerado um processo. Entende-se por processo "um conjunto inter-relacionado de recursos e atividades que transformam entradas e saídas". As entradas e saídas são também denominadas insumos e produtos, respectivamente.

Recursos: são os meios necessários ao processo e compreendem, segundo a ISO (*International Organization for Standardization*), gerenciamento, serviços, pessoas, finanças, instalações, equipamentos, técnicas e métodos. Sinteticamente, os recursos são classificados em três categorias: financeiros, humanos e materiais.

Atividade: é qualquer ação ou trabalho específico exercido sobre as entradas e executado pelos recursos, com a finalidade de transformá-los em saídas.

Entrada (ou insumo): é tudo aquilo que é fornecido ao processo para:

- Utilização (um dado ou informação, uma instrução, um instrumento, um serviço de máquina, um trabalho humano etc.)
- Transformação (energia, matéria-prima etc.)
- Consumo (energia, material de escritório etc.)
- Em síntese, pode-se dizer que entradas são "documentos ou itens documentáveis sobre os quais as ações serão executadas".

Produto ou saída: é o resultado de atividades ou processos, podendo ser materiais e equipamentos, materiais processados, informações, serviços ou uma combinação destes.

Tanto as entradas quanto as saídas podem ser:

- Tangíveis (materiais processados, por exemplo)
- Intangíveis (uma informação ou uma decisão, por exemplo)

E o produto pode ser:

- **Intencional**, isto é, um bem ou serviço, constituindo o objetivo declarado do processo;
- **Não intencional**, aquele que se forma e é considerado um resultado não procurado, podendo ter efeitos indesejáveis ou ser valioso (usualmente chamado de subproduto).

Um processo fica definido quando são descritos ou especificados:

- As entradas;
- Os recursos;
- As atividades;
- As saídas.

Não podemos esquecer que todas as ações pertencentes aos processos têm o objetivo de gerar resultados. Portanto, os grupos de processo são ligados pelos resultados que produzem, sendo que o resultado de um processo frequentemente é a entrada de outro processo. Dependendo das conveniências, e segundo uma visão macroscópica, pode-se considerar que um processo seja constituído de vários subprocessos, simplesmente agregando suas partes construtivas mais simples. O processo define quem irá fazer o que e como será atingido o objetivo. Na engenharia de software, por exemplo, o objetivo será construir um software ou melhorar um existente, conforme ilustra a Figura 1.1.

Figura 1.1 - Objetivos da engenharia de software.

De uma maneira resumida, podemos dizer que o processo de desenvolvimento de projetos possui as seguintes regras gerais:

- Oferece um guia para definir as atividades da equipe;
- Especifica quais artefatos devem ser desenvolvidos;
- Direciona as tarefas individuais e da equipe como um todo;
- Oferece critérios para monitoração e medição dos produtos e atividades do projeto;
- Proporciona linhas gerais para os usuários e desenvolvedores do projeto;
- Reduz riscos e torna os projetos mais previsíveis;
- Oferece visões comuns e dissemina a cultura de desenvolvimento;
- Serve como um template que pode ser reutilizado.

O modelo de processo de software

Existem vários modelos de processo de software, que descrevem os processos feitos para atingir o desenvolvimento do software. Esses modelos em geral incluem os seguintes tipos de representações:

- Tarefas
- Artefatos (arquivos, dados etc.)

- Atores
- Decisões (opcional)

A notação usada pode variar, mas normalmente o modelo padrão de processo de software usa uma elipse para representar tarefas e processos. Os artefatos são representados por retângulos e os atores, por figuras ("homem-palito"). Muitos modelos de processo de software não incluem decisões. Neste caso, quando precisarmos indicá-las, usaremos um losango. O fluxo é representado por linhas não nomeadas, que seguem da esquerda para a direita e de cima para baixo. Embora existam outras notações comumente utilizadas, as regras e interpretações corretas para o modelo de processo são:

- Duas tarefas não podem ser conectadas por uma linha. Tarefas devem ser separadas por artefatos.
- Uma tarefa não é executável até que seu artefato de entrada exista.
- Há uma ou mais tarefas de início e uma ou mais tarefas de término.
- Existe um caminho de todas as tarefas para a tarefa terminal.

Um modelo de processo de software pode ser descritivo, isto é, pode descrever o que aconteceu em um projeto de desenvolvimento. Esse modelo geralmente é criado como parte de uma análise final de um projeto e pode ser útil na identificação de problemas no processo de desenvolvimento de software. O modelo de processo de software pode, ainda, ser prescritivo, isto é, pode descrever o que poderá acontecer. Esse modelo pode ser usado para descrever o processo padrão de desenvolvimento de software e ser usado como ferramenta de treinamento, tanto para referências a ocorrências incomuns como para documentação do que supostamente esteja ocorrendo. A Figura 1.2 ilustra um exemplo de modelo de processo.

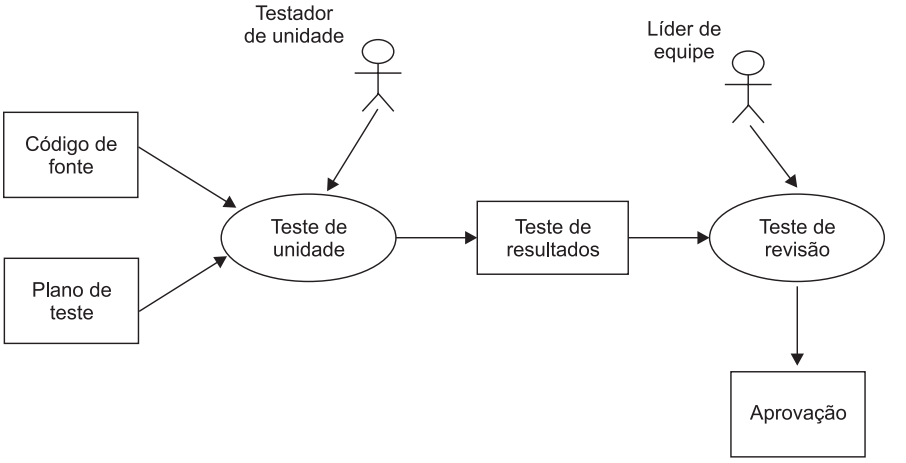


Figura 1.2 - Exemplo de um modelo de processo.

Este modelo possui dois atores: o de teste e o líder da equipe. O testador de unidade, evidentemente, é responsável pelo teste da unidade. Ele usa o código-fonte e o plano de teste para acompanhar o teste de unidade. O resultado do teste é um artefato, os resultados do teste. O líder da equipe revê esses resultados e deve aprovar o teste de unidade. Esse modelo não

mostra explicitamente o que acontece quando o processo é bem-sucedido. Pode ser inferido que o testador continuaria a testar até que o líder ficasse satisfeito. Da mesma forma, se o líder do time não está pronto para dar aprovação, então o processo pode retornar para refazer o teste de unidade.

1.4.2 Processos de negócio

Desde que Taylor escreveu a primeira publicação de caráter científico sobre a administração de empresas (*Princípios de Administração Científica*), é fácil perceber que o mundo empresarial tem experimentado constantes mudanças de paradigma, que incluem diversas propostas sobre modelos de gestão mais adequados. O paradigma mais recente é o modelo de gestão por processos de negócio, cujo objetivo é contribuir para a sistematização da estrutura das organizações. Toda abordagem por processos está sempre associada ao entendimento do conjunto de atividades desenvolvidas na organização, ligadas como se fossem elos de uma corrente. Cada elo trabalha para o todo, sendo parte dele, visando produzir o resultado esperado. As organizações podem ser consideradas um sistema, ou seja, funcionam como um conjunto de processos. Portanto, a identificação e a modelagem desses processos permitem um planejamento eficiente das atividades, a definição de responsabilidades e o uso adequado dos recursos disponíveis. Contudo, identificar e modelar os processos de uma organização é muito mais do que um simples retrato da lógica de entradas e saídas entre pessoas, cargos, departamentos, gerências ou áreas. É um exercício de reflexão e debates cujo objetivo é retratar fielmente, por meio de uma ferramenta visual, todos os aspectos essenciais para o funcionamento de uma organização.

Cada vez mais as empresas vêm percebendo a importância de implementar uma boa administração para que sejam capazes de atender as demandas dos clientes com bons resultados. Sabe-se que a boa administração começa com a compreensão precisa do seu negócio, que muitas vezes apresenta-se complexo. Para lidar com essa complexidade, as empresas vêm utilizando técnicas de engenharia para manipular seus processos, surgindo a chamada engenharia de processos de negócio. Trata-se de um conjunto de técnicas que permitem projetar um negócio de acordo com critérios estabelecidos.

Analogamente à engenharia de processos de negócios, podemos fazer uso da reengenharia de negócios, que neste caso sugere e aplica mudanças no estado atual de um negócio a partir de uma análise das funções existentes, dos seus objetivos e relacionamentos. Esses estudos remetem ao conceito de processos de negócios, que de uma maneira geral representa uma ação coordenada, composta de todas as atividades (que podem ser concorrentes ou paralelas) necessárias para entregar um determinado resultado (produto ou serviço) aos clientes de uma empresa. Os processos de negócio possuem características bem definidas, citadas a seguir:

- São dinâmicos e devem responder a mudanças nas demandas dos clientes e nas condições de mercado.
- Algumas de suas atividades são passíveis de automação (por computadores ou máquinas).
- Possuem atividades que não podem ser automatizadas, pois dependem da inteligência e do julgamento de seres humanos.
- São complexos e envolvem o fluxo de materiais, informações ou acordos de negócio.
- Podem ser distribuídos entre diversas empresas e utilizar plataformas diferentes.

Para identificar, aperfeiçoar e manter os processos de negócio, devemos utilizar uma técnica específica, conhecida como modelagem de negócios. Para entender melhor este conceito, convém dividirmos a modelagem de negócios em dois aspectos a serem estudados: a modelagem da organização e a modelagem dos processos de negócio. Quando modelamos a organização, buscamos o entendimento das suas metas, a identificação das suas unidades organizacionais, a identificação de papéis desempenhados pelas pessoas que trabalham nas unidades organizacionais e a identificação das localidades onde a organização está distribuída, não necessariamente nesta ordem.

Um modelo de negócios permite identificar atividades redundantes que podem ser eliminadas, além de outras atividades propensas a erros e com vocação para serem automatizadas. Portanto, modelos de negócio devem ser usados como base para a análise de requisitos de sistemas computacionais. Isso se torna claro quando lembramos que os sistemas de informação basicamente têm seu foco no fluxo de trabalho de uma empresa e nas informações que transitam nesse fluxo. É interessante perceber que, enquanto o levantamento de requisitos oferece uma visão externa de um sistema, de maneira análoga a modelagem de negócios oferece uma visão externa de uma empresa. A Figura 1.3 ilustra essa analogia.

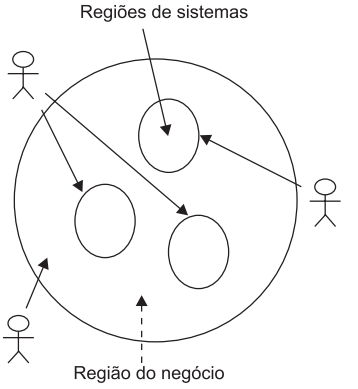


Figura 1.3 - Exemplo de uma visão em camadas.

Devido à crescente complexidade das organizações e à maior abrangência exigida pelos sistemas de informação, já não é suficiente usarmos a modelagem de negócio e a modelagem de software separadamente. Atualmente é necessário haver um alinhamento entre essas duas modelagens, pois isso é determinante para o sucesso ou fracasso de qualquer processo de implementação de sistemas de informação. A partir de uma visão mais interna do negócio, capturada em um modelo de objetos de negócios, pode-se observar os requisitos básicos para implementar o comportamento e as características dos sistemas necessários para dar suporte aos negócios. Os modelos de negócios fornecem uma entrada para a visão de caso de uso e para uma visão lógica do sistema. Também podemos observar pelos modelos de negócio os principais mecanismos no nível da análise de sistemas, denominados mecanismos de análise.

1.5 Exercícios

1) O que motivou o surgimento das metodologias ágeis?

2) Qual é o papel que um sistema de informação desempenha nos processos que agregam valor?

3) O que assegura a qualidade do software, permitindo inclusive que eventuais modificações ao longo do ciclo de vida do projeto sejam adequadamente geridas?

4) Os recursos são os meios necessários ao processo. Basicamente, como os recursos podem ser classificados? Cite exemplos de recursos utilizados em um processo.

5) Que elementos básicos podem caracterizar um processo bem definido?

6) Existem vários modelos de processo de software, que descrevem os processos que são feitos para atingir o desenvolvimento do software. Quais os tipos básicos de representações que esses modelos em geral incluem?

7) Sabe-se que a boa administração começa com a compreensão precisa do seu negócio, que muitas vezes apresenta-se complexo. Como as empresas têm feito para lidar com essa complexidade?

8) Qual é a contribuição que um modelo de negócio traz para uma empresa e qual o seu relacionamento com a análise de requisitos de sistemas computacionais?

Anotações

2

Fundamentos sobre Gestão de Projetos

Objetivos

- *Apresentar uma visão geral dos fundamentos de gestão de projetos.*
- *Argumentar sobre o que é um projeto e como se classificam.*
- *Enfatizar as diferenças entre projetos e processos.*
- *Introduzir conceitos básicos de gerenciamento de projetos.*

2.1 Projetos: por que me preocupar com isso?

Não seria conveniente iniciar o estudo de metodologias de desenvolvimento ágeis antes de conhecer fundamentos sobre a gestão de projetos. O gerenciamento de projetos é uma disciplina que vem sendo formada há muito tempo por pessoas de diversas áreas de conhecimento e especializações, englobando vários países e ambientes, e praticamente todos os tipos de organizações. Para ilustrar a razão pela qual cada vez mais nos preocupamos com projetos e a melhor maneira de geri-los, vamos observar, a seguir, aspectos relacionados ao próprio conceito de projeto.

2.1.1 O que é um projeto?

Projeto é um empreendimento com **características próprias**, tendo **princípio e fim**, conduzido por pessoas, para atingir metas estabelecidas dentro de parâmetros de **prazo**, **custo** e qualidade (segundo o PMBOK - *Project Management Body of Knowledge* - conjunto de práticas em gestão de projetos publicado pelo PMI - *Project Management Institute*. As características do PMBOK serão exploradas no capítulo 4). Considerando essa definição, percebemos que qualquer atividade, mesmo até uma simples ida ao supermercado, poderia ser tratada como um projeto. Neste exemplo,

podemos dizer que a lista de compras a ser feita no supermercado seria o **objetivo** do projeto. O tempo disponível para as compras é o **prazo**. O **custo** do projeto seria o preço das compras. Assim, se você planejar bem, certamente comprará o que precisa, poupará tempo no supermercado e, comprando só o que precisa, consequentemente economizará dinheiro. O objetivo do projeto deve ser redigido com o máximo cuidado, para ser claro, devendo conter:

- **A ação:** definida por um verbo no infinitivo, iniciando a declaração do objetivo (exemplo: projetar, desenvolver, construir, transformar, modernizar, ensaiar, levantar, determinar, obter etc.).
- **O objeto:** sobre o qual a ação se exerce e/ou da qual ele resulta (exemplo: uma ponte, um dispositivo, um processo administrativo ou operacional, um treinamento, uma simulação, um software etc.).
- **Requisitos, restrições ou condições complementares:** de desempenho, de tempo, de local, de qualidade, de quantidade, de áreas de aplicação etc.

Exemplos de objetivos de projetos

1) Um exemplo simples e bem conhecido é o do programa espacial norte-americano: "*Colocar um homem na Lua até o fim da década*".
2) Outro exemplo é o objetivo fixado pelos laboratórios da Bell Telephone: *"Obter novos conhecimentos que possam ser utilizados no desenvolvimento de novos e completamente aperfeiçoados componentes e elementos de aparelhos de sistemas de comunicação"*.
3) E um último: "*Preparar um treinamento de até x horas, para o pessoal da empresa, a fim de capacitá-los à implantação progressiva da administração por projetos a ser efetivada até o fim do próximo ano*".

2.1.2 Propósitos de um projeto de software

- Transformar os requisitos em um projeto que permitirá construir o sistema.
- Nivelar os artefatos existentes com o ambiente de desenvolvimento.
- Identificar os componentes de software e seus relacionamentos.

Centenas de atividades realizadas no dia a dia das empresas estão relacionadas com projetos. Projetos também podem ser definidos como uma **iniciativa não repetitiva**, ou seja, um empreendimento único com um começo e um fim. Assim, como as organizações estão mergulhadas até o pescoço em projetos, a forma pela qual esses múltiplos esforços serão gerenciados determinará se a empresa prosperará ou não. Esta aí uma boa razão para nos preocuparmos com projetos, pois as organizações que implementam com sucesso uma abordagem de projetos encontram reduções de custos através do menor retrabalho e aumento nos lucros pelo término dentro do prazo. Observe que não devemos confundir operações repetitivas com projetos. Operações e projetos são diferentes, principalmente pelo fato de que projetos são temporários e exclusivos, enquanto as operações são contínuas e repetitivas.

2.2 Processos X projetos

Antes de apresentarmos conceitos relacionados ao gerenciamento de projetos, é bom lembrar que projetos são diferentes de processos. Projetos estão relacionados com algo novo, que tem início e fim bem definidos. Já os processos ocorrem de maneira contínua, com a finalidade de produzir produtos ou serviços idênticos. Processos normalmente são estabelecidos a partir da conclusão de um projeto. Os projetos produzem um produto único, que é diferente em pelo menos uma característica de qualquer outro produto existente. Já produtos decorrentes de processos serão sempre idênticos. Apesar de conceitualmente diferentes, podemos perceber que tanto os processos quanto os projetos são limitados em escopo e recursos empregados.

Todo processo passa por fases que vão desde a concepção até sua conclusão. No caso específico de processos de gerenciamento de projetos, fazemos uso de um conjunto de fases inter-relacionadas e coordenadas que têm por finalidade atingir os objetivos pretendidos. Essas fases representam o chamado ciclo de vida do projeto. Quando estamos gerenciando um projeto, devemos trabalhar basicamente com dois conjuntos simultâneos de processos: os processos de gerenciamento de projetos, utilizados para descrever e organizar o trabalho do projeto, e os processos orientados ao produto desejado, que objetivam especificar e criar o produto do projeto.

Os processos que descrevem e organizam o trabalho do projeto são divididos em cinco grupos:

- **Início:** processos que reconhecem que o projeto ou determinada fase deve iniciar.
- **Planejamento:** processos que dividem o projeto em etapas ou fases, representando componentes menores que vão ao encontro dos objetivos do projeto.
- **Execução:** processos que coordenam pessoas e recursos empregados, de acordo com planos previamente estabelecidos.
- **Controle:** processos que visam garantir que os objetivos do projeto serão atingidos, pelo uso de técnicas de monitoramento, medição de progresso e tomada de ações corretivas, se for necessário.
- **Finalização:** processos que formalizam a aceitação do projeto ou de determinada fase.

Já os processos de gerenciamento de projetos se dividem em subprocessos relacionados a áreas de conhecimento em gestão de projetos, que interagem ao longo de todo o projeto. Como exemplo de processos consagrados e muito utilizados temos o PMBOK (*Project Management Body of Knowledge*) e o SWEBOK (*Software Engineering Body of Knowledge*), ambos explorados no capítulo 4. A seguir resumiremos alguns tipos de processos que podem ser utilizados para gerenciar um projeto:

- **Gerenciamento de tempo:** processos necessários para garantir que o projeto seja concluído no prazo.
- **Gerenciamento de custo:** processos necessários para garantir que o projeto seja concluído dentro do orçamento.
- **Gerenciamento de escopo:** processos necessários para garantir que o projeto inclua todo e apenas o trabalho necessário para sua conclusão ser satisfatória.

- **Gerenciamento de risco:** processos necessários para identificação, análise e respostas a riscos do projeto.
- **Gerenciamento de qualidade:** processos necessários para garantir que o projeto irá satisfazer as necessidades para as quais ele foi empreendido.
- **Gerenciamento da integração:** processos necessários para garantir que os diversos elementos que compõem um projeto sejam devidamente coordenados. O gerenciamento da integração normalmente é retratado por meio do desenvolvimento de um plano de projeto e de um controle de mudanças.
- **Gerenciamento da comunicação:** processos necessários para garantir a geração, organização, armazenamento, disseminação das informações necessárias à condução do projeto.
- **Gerenciamento de recursos humanos:** processos necessários para a obtenção do uso mais efetivo das pessoas envolvidas com o projeto.
- **Gerenciamento de contratos ou aquisições:** processos necessários para a aquisição de bens e serviços provenientes de fora da organização.

2.3 Gerenciamento de projetos

Existem diversas instituições que se preocupam em estudar, formalizar e divulgar boas práticas de gerenciamento de projetos. Como exemplo, a seguir vamos citar duas delas.

2.3.1 Project Management Institute (PMI)

O PMI, sediado na Pensilvânia, Estados Unidos, é uma associação sem fins lucrativos de profissionais da área de gerenciamento de projetos. O PMI visa manter e ampliar o conhecimento existente sobre gerenciamento de projetos, assim como melhorar o desempenho de profissionais e organizações nessa área. Desde sua fundação, em 1969, o PMI cresceu para ser a "organização dos profissionais de gerência de projetos". Com cerca de 90.000 membros em todo o mundo, o PMI é hoje a organização mais importante da área de gerenciamento de projetos, estabelecendo padrões, provendo seminários, programas educacionais e certificação profissional que cada vez mais as organizações exigem dos seus líderes de projeto.

Duas das principais iniciativas do PMI na difusão do conhecimento em gerenciamento de projetos estão nas certificações profissionais em gerência de projetos (*Project Management Professional* - PMP e *Cerfified Associate in Project Management* - CAPM) e na publicação de padrões globais de gerenciamento de projetos, sendo o mais popular o PMBOK, editado sob a forma de um livro e traduzido oficialmente para diversos idiomas, inclusive o português do Brasil.

2.3.2 Institute of Electrical and Electronic Engineers (IEEE)

Criado em 1984, nos EUA, é uma sociedade técnico-profissional internacional, dedicada ao avanço da teoria e prática da engenharia nos campos da eletricidade, eletrônica e computação. O IEEE congrega mais de 312.000 associados, entre engenheiros, cientistas, pesquisadores e outros

profissionais, em cerca de 150 países, promovendo a engenharia da criação, desenvolvimento, integração, compartilhamento e o conhecimento aplicado no que se refere à ciência e às tecnologias da eletricidade e da informação.

Das 300 seções do IEEE existentes no mundo, cinco delas estão no Brasil e, juntas, formam o Conselho Brasil. Durante o período de 1981-1985, a IEEE realizou uma série de workshops sobre aplicações de padrões de engenharia de software, onde os envolvidos puderam compartilhar suas experiências com as normas existentes na época. Esses workshops também tinham por objetivo realizar sessões de planejamento envolvendo métricas para produtos de software e processos. Esse esforço resultou no planejamento do padrão IEEE 1002 (taxonomia de padrões de engenharia de software), que fornecia uma visão nova e holística da engenharia de software.

Em 1990, foi iniciado um planejamento para a adoção de um padrão internacional, focado na conciliação de pontos de vista de processos de software já existentes na IEEE. Essa norma internacional foi concluída em 1995 com a designação de ISO/IEC 12207 (norma para o processo de ciclo de vida de software). Em seguida, uma comissão mista, que também contava com a participação da ACM (*Association for Computing Machinery*), aprovou uma moção para a criação de uma comissão mista para estabelecer os conjuntos apropriados de normas e critérios para o exercício profissional da engenharia de software, para certificação profissional, além de programas educacionais. Essas boas práticas para a engenharia de software foram concluídas em 1998, culminando na publicação conhecida por SWEBOK (*Software Engineering Body of Knowledge*), aprovada tanto pela IEEE quanto pela ACM e passando a ser adotada por inúmeras organizações. Vamos apresentar informações complementares sobre o SWEBOK ainda no capítulo 4.

2.3.3 Conceitos básicos de gestão de projetos

A gestão de projetos consiste da combinação única de recursos - físicos, humanos, materiais e financeiros - sistemas e técnicas gerenciais para atender aos objetivos estabelecidos para o projeto. Os gerentes de projetos dependem basicamente de quatro competências, sendo Conhecimento, Destreza, Habilidade e Motivação, além de desenvolver habilidades relacionadas à Administração de Conflitos.

- **Conhecimento:** refere-se à capacidade de compreender a teoria, os conceitos e as práticas da gerência de projetos.
- **Destreza:** ter a capacidade de usar as técnicas e os recursos da profissão para obter resultados adequados.
- **Habilidade:** capacidade de integrar e usar de modo eficaz o conhecimento e as aptidões.
- **Motivação:** desenvolver e manter continuamente valores, atitudes e aspirações adequados, que ajudam os *stakeholders* (principais interessados no projeto) a trabalharem em conjunto para o aperfeiçoamento do projeto.
- **Administração de conflitos:** adquirir ou melhorar os seus conhecimentos acerca do processo de negociação, contribuindo para o desenvolvimento da capacidade de buscar um ponto de consenso no qual as partes envolvidas, numa negociação ou situação de conflito, cheguem a resultados positivos.

Segundo o PMBOK, gerenciamento de projetos "*é a aplicação de conhecimentos, habilidades, ferramentas e técnicas em projetos com o objetivo de atingir ou até mesmo exceder às necessidades e expectativas dos clientes e demais partes interessadas do projeto*".

É importante percebermos que os projetos envolvem decisões, escopo, tempo, custo, qualidade, diferentes necessidades e expectativas dos clientes e partes interessadas, requisitos identificados (necessidades) e não identificados (expectativas) etc. A Figura 2.1 apresenta uma visão de três pilares básicos que sustentam um projeto (escopo, custo e prazo), sustentados pela qualidade.

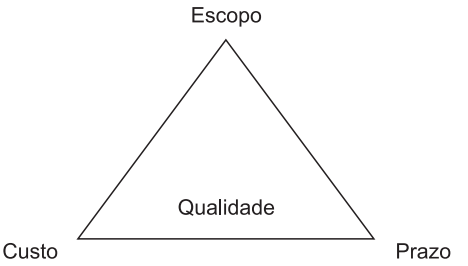


Figura 2.1 – Três pilares de um projeto: escopo, custo e prazo.

2.3.4 Quando é importante gerenciar projetos?

- Quando temos de desenvolver um produto complexo.
- Quando é necessário controlar prazos e custos.
- Quando é necessário compartilhar recursos.

Projetos: problemas típicos

Também podemos entender os esforços empregados na gestão de projetos a partir do momento em que pensamos nos problemas típicos com que temos de conviver durante a execução de um projeto, como os exemplificados a seguir:

- Atrasos nos cronogramas;
- Produtos que não funcionam;
- Falta de recursos de pessoal;
- Mudanças de requisitos e especificações;
- Qualidade abaixo da esperada;
- Complexidade acima da capacidade;
- Produtos mal projetados;
- Custo acima do previsto;
- Projetos que são cancelados.

2.3.5 Fases e ciclo de vida de projetos

Projetos possuem características únicas e distintas e estão associados a certo grau de incerteza. São normalmente divididos em várias fases de projeto, com o objetivo de melhorar o seu controle e gerenciamento. O conjunto das fases de um projeto é conhecido como ciclo de vida de um projeto. Normalmente nas fases iniciais as incertezas são maiores e as possibilidades de modificações no projeto são altas. À medida que o projeto avança, diminuem as incertezas e as possibilidades de modificações no projeto.

Cada fase é marcada pela conclusão de um ou mais produtos. Produtos e fases possuem uma sequência lógica de atividades para sua execução, e em cada fase devemos estabelecer um ponto de revisão, que deve determinar se o projeto deve prosseguir para a próxima fase, além de detectar e corrigir erros. As fases do projeto e seus produtos geralmente são identificados pelo próprio nome dos itens produzidos e das atividades realizadas, como, por exemplo, fase de requisitos, fase de projeto, fase de construção, fase de integração, fase de implantação etc. Em algumas áreas de aplicação, estes nomes são definidos de acordo com o modelo de ciclo de vida utilizado no projeto.

Observe que, em projetos complexos, as fases podem ser estruturadas em pacotes. Por exemplo, durante o desenvolvimento de um módulo de contas a pagar (que pertence a uma fase do projeto) podemos observar o desenvolvimento das atividades de análise, projeto, construção, integração, implantação e homologação. Em um segundo momento, perceberemos a repetição dessas atividades para o contas a receber e assim por diante.

Um bom começo quando queremos definir o ciclo de vida de um projeto é pensarmos nas características que o próprio ciclo de vida deve ter, descritas a seguir:

- O ciclo de vida deve definir o início e o fim do projeto.
- Deve determinar quais ações de transição deveriam estar ou não incluídas ao final das fases do projeto ou entre os projetos.
- Deve definir o trabalho ou esforço técnico que deveria ser feito em cada fase do projeto.
- Deve determinar quais pessoas devem estar envolvidas em cada fase do projeto.

2.4 Exercícios

1) Como podemos definir um projeto?

2) Qual é a diferença entre processos e projetos?

3) Quais são os cinco grupos básicos que descrevem e organizam o trabalho de qualquer projeto? Resuma cada um deles.

4) Cite exemplos de tipos de processos que podem ser utilizados para gerenciar um projeto. Comente os exemplos citados.

5) Descreva, de forma resumida, as quatro competências básicas que os gerentes de projeto devem ter.

6) Cite exemplos de problemas típicos que podemos encontrar durante a gestão de projetos.

Anotações

3

Engenharia de Software: Conceitos Básicos

Objetivos

- *Apresentar conceitos básicos utilizados na engenharia de software.*
- *Caracterizar os sistemas de informação.*
- *Enfatizar a importância da definição adequada do escopo do projeto.*
- *Apresentar as principais preocupações com requisitos de software.*
- *Conceituar a fase de análise.*
- *Apresentar os conceitos de abstração, representação e comentar a denominação das principais ferramentas de gestão utilizadas.*

3.1 Considerações iniciais

Genericamente podemos definir engenharia como a ciência de adquirir e de aplicar determinados conhecimentos abrangentes para realizar uma função ou objetivo. No caso da engenharia de software, observamos que é uma área de conhecimento da computação, cujos conhecimentos se relacionam basicamente com a especificação, o desenvolvimento e a manutenção de sistemas de software, com o objetivo de organizar o projeto e aumentar sua produtividade. Além destes objetivos, chamamos a atenção para outro objetivo importante, obtido pela aplicação de práticas de engenharia para o desenvolvimento de software: fornecer uma estrutura para a construção de software com alta qualidade.

Podemos encontrar na literatura ou mesmo em pesquisas na Internet diversas definições para engenharia de software. Objetivando mostrar formalmente uma delas, escolhemos a definição proposta por Fritz Bauer, apresentada em uma conferência pioneira sobre este assunto: "Engenharia de software é a criação e utilização de sólidos princípios de engenharia, a fim de obter software de maneira econômica, que seja confiável e que trabalhe eficientemente em máquinas reais". Como esta definição foi dita para servir de motivação para discutir melhor este tema, podemos ressaltar a tentação que o leitor pode ter em procurar ampliá-la, considerando alguns detalhes passíveis de serem observados. Por exemplo, esta definição:

- não inclui diretamente a necessidade de satisfação do cliente ou aspectos relacionados à pontualidade;
- diz pouco sobre aspectos técnicos referentes à qualidade de software;
- não fala sobre a importância de utilizarmos medidas e unidades; ou
- não fala sobre a importância de adotarmos um processo de desenvolvimento amadurecido.

De fato, a definição de Bauer desperta quem a interpreta para questões complementares. Por exemplo, ela é omissa sobre quais "sólidos princípios de engenharia" poderiam ser aplicados ao desenvolvimento de software para computador. Também não menciona como podemos construir software "economicamente" de modo que ele seja "confiável". Independente de a resposta a estas questões não estar explícita, certamente elas continuam a desafiar os engenheiros de software. Talvez a dificuldade encontrada quando tentamos reproduzir em uma única sentença toda a responsabilidade atribuída à engenharia de software venha do fato de ser uma tecnologia em camadas. Como citado anteriormente, hoje sabemos que qualquer abordagem de engenharia, incluindo a engenharia de software, certamente deve se apoiar em um compromisso com a qualidade. Por isso o foco na qualidade é a camada base na sustentação dessa disciplina, seguida, nesta ordem, do uso de processos, uso de métodos e de ferramentas adequadas.

3.1.1 Aplicação de técnicas de engenharia ao desenvolvimento de software

Para entendermos de que maneira técnicas de engenharia podem ser aplicadas ao desenvolvimento de software, é interessante examinarmos que características tornam o software diferente de outras coisas que os seres humanos podem produzir. Para entender essas diferenças, um bom começo seria fazermos uma analogia com processos de construção de hardware. Neste caso, torna-se claro observarmos quais são os processos humanos criativos envolvidos (exemplo: análise, projeto, construção e testes), pois, em última análise, o hardware será traduzido sob uma forma física ou tangível. Já o software, por se tratar de um elemento de um sistema lógico e não de um sistema físico, certamente possui características bem diferentes daquelas que observamos no hardware, tornando as duas atividades fundamentalmente diferentes.

Outro aspecto interessante é que as duas atividades são dependentes de pessoas, mas a relação entre as pessoas envolvidas e o trabalho realizado é inteiramente diferente, pois o software desenvolvido passa por um processo de engenharia não manufaturado no sentido clássico. Outro detalhe é que os custos do desenvolvimento de um software se concentram basicamente na engenharia, o que significa que projetos de software não podem ser geridos como se fossem projetos de fabricação.

Estudos indicam que as taxas de falha que podemos encontrar no hardware são relativamente altas logo após começar sua vida útil, falhas essas muitas vezes atribuídas a defeitos de projetos ou de fabricação. Neste caso, após os defeitos terem sido constatados e corrigidos, normalmente a taxa de falhas cai para um nível constante, que idealmente deve ser bastante baixo, por um período de tempo determinado. Com o passar do tempo, pode ser que as taxas de falha aumentem novamente, considerando que os componentes de hardware sofrem efeitos acumulativos relacionados com vibrações, maus-tratos, variações de temperatura, acúmulos de poeira etc. Em outras palavras, o hardware começa a se desgastar.

Ao tentarmos transportar essas possibilidades para produtos de software, percebemos que ele não é suscetível a males ambientais causadores de desgaste, como os observados anteriormente no hardware. No caso do software, defeitos não detectados podem causar altas taxas de erros no início de sua utilização. Na medida em que esses erros são corrigidos, idealmente sem a introdução de novos erros, a tendência é que o software se estabilize.

É curioso também observar que, no caso do hardware, quando uma peça se desgasta, ela pode ser totalmente substituída por outra peça, possibilidade que não se aplica a um software. Quando estudamos as causas de falha em sistemas de software, observamos que geralmente são decorrentes de erros de projeto ou no processo utilizado para seu desenvolvimento. Por isso, manter um sistema de software envolve uma complexidade muito maior quando comparada à manutenção de hardware.

3.1.2 O uso da componentização

Atualmente percebemos que existe uma tendência da indústria de software em utilizar a chamada "componentização" ou montagem baseada em componentes. Isso tem ocorrido devido à evolução da engenharia, que permitiu que coleções de componentes fossem criadas, principalmente com a popularização das linguagens orientadas a objetos e da UML. Mesmo existindo uma variedade muito grande de componentes e classes à disposição, ainda observamos que a maior parte dos softwares utilizados continua sendo desenvolvida sob encomenda.

A padronização de componentes de hardware tem sido muito evidente desde o advento da Revolução Industrial, como, por exemplo, ao observarmos padrões de parafusos ou de circuitos integrados. É importante observar que os padrões acabam sendo criados para serem usados e reutilizados por engenheiros quando projetam novos produtos, de forma que não precisem se preocupar com o que já existe e possam se concentrar nos elementos inovadores de seu projeto (partes que apresentam algo de novo no mundo do hardware ou software).

Hoje em dia esse conceito tem sido amplamente difundido na engenharia de software, pois existe uma preocupação constante em projetar um novo componente de modo que ele possa ser utilizado novamente em programas diferentes, sem termos de "reinventar a roda". Esta já era uma preocupação existente na década de 1960, quando pudemos observar o uso de bibliotecas de sub-rotinas científicas que podiam ser reutilizadas em um amplo conjunto de aplicações. Apesar de essas sub-rotinas e bibliotecas utilizarem algoritmos muito bem definidos, tinham um domínio de aplicação limitado.

Atualmente a visão de reúso utilizada não se restringe apenas aos algoritmos, tendo sido ampliada de modo que também comporte o uso de estruturas de dados. Portanto, componentes de software modernos e reutilizáveis conseguem encapsular tanto dados como o processamento que aplicamos a eles, conforme podemos observar nos paradigmas orientados a objeto. Essa importante característica vem permitindo que engenheiros de software criem aplicações a partir de partes reutilizáveis, estruturadas em várias camadas, física e geograficamente distribuídas.

3.2 Características dos sistemas de informação

Conceito de sistema

Antes de estudarmos as características dos sistemas de informação, vamos refletir um pouco sobre um conceito mais abrangente, que é o conceito de sistema. Podemos dizer, basicamente, que sistema é um conjunto de partes que interagem para funcionarem como um todo. Usamos essa ideia quando dizemos "sistema nervoso", "sistema jurídico" ou "sistema de refrigeração". Um radiador, um ventilador, uma bomba de água, um termostato, diversas mangueiras e braçadeiras, tudo isso organizado de uma maneira correta forma o sistema de refrigeração de um automóvel. Contudo, separadamente são inúteis para proteger o motor do carro.

Portanto, podemos observar que quando temos um conjunto de partes e não importa a maneira como estão organizadas, é possível dizer que temos uma "pilha" ou um "monte", mas não um sistema. A questão que pode surgir neste momento é: se as peças de um sistema agem juntas como uma unidade única, será que esse sistema pode ser "parte de um sistema maior"? Intuitivamente sabemos que sim. Como exemplo podemos citar que um sistema de refrigeração de um motor é um subsistema desse motor que, por sua vez, é subsistema do veículo.

Ludwig Von Bertalanffy

Quando apresentamos noções conceituais de sistemas, não podemos deixar de mencionar a importante contribuição de Karl Ludwig Von Bertalanffy, um biólogo alemão que se dedicou a elaborar a chamada "teoria geral dos sistemas", entre 1950 e 1968. É reconhecido pelo mundo inteiro como pioneiro em defender a visão organística na biologia e o papel da simbologia na interpretação da experiência humana. Sobretudo, é aceito como um dos fundadores da teoria geral dos sistemas. No seu trabalho, expõe o significado do conceito da teoria geral dos sistemas e suas principais aplicações, consideradas uma reorientação na ciência em geral e em toda a escala de disciplinas que vão da física e da biologia às ciências sociais, comportamentais e à própria filosofia. Ele sugere que, em vez de pensarmos em dividir o mundo em áreas de conhecimento (como física, química, biologia etc.), devemos estudar os diversos tipos de sistemas existentes globalmente, envolvendo todas as suas interdependências. Sua justificativa baseava-se no fato de que cada um dos elementos, ao serem reunidos para constituírem uma unidade funcional maior, desenvolve características que não se encontram em seus componentes isolados.

Definição de sistema

Basicamente, podemos definir sistema como um conjunto de elementos inter-relacionados com um objetivo comum. Se utilizarmos uma definição mais elaborada, objetivando complementar melhor este conceito, sistema seria um conjunto de elementos interconectados harmonicamente, de modo a formar um todo organizado. É curioso perceber que esta é uma definição que pode ser aplicada para várias disciplinas, como biologia, medicina, informática, administração. Remetendo a origens gregas, observamos que o termo "sistema" significa "combinar", "ajustar", "formar um conjunto". Podemos observar que todas as áreas de conhecimento possuem sistemas e que tais sistemas possuem características e leis independentes da área em que se encontram, como, por exemplo, sistema solar, sistema monetário, sistema nervoso, sistema elétrico, sistema social etc.

Todo sistema possui um objetivo, embora às vezes seja difícil identificá-lo - por exemplo, quando não conseguimos visualizar o meio ambiente em que está inserido. Um sistema consiste de componentes, entidades, partes ou elementos - embora também possam ser vistos como subsistemas - e as relações entre eles. A integração de tais componentes pode se dar por fluxo de informações, matéria, energia. Desta forma, um sistema fica inteiramente definido quando se conhecem:

- Seu objetivo (sua finalidade e relacionamento externo);
- Seus limites ou fronteiras (seu campo de influência e as entidades que o influenciam);
- Os subsistemas (sua estrutura);
- As funções e o inter-relacionamento de seus subsistemas (funcionamento interno).

Considerando o exposto anteriormente, podemos então concluir que um sistema é um conjunto organizado, uma combinação ou montagem de entidades, de partes, de processos ou de elementos interdependentes que formam um complexo unitário, podendo comportar diversas dimensões. Baseado nesta definição, o termo "sistema" cobre um largo espectro, onde, por exemplo, o Universo pode ser visto como uma sequência hierarquizada de sistemas naturais (sistema solar, um planeta, uma bacia fluvial, um bosque, uma árvore, um pássaro, um verme etc.). Outros sistemas foram construídos por pessoas, com fins específicos, chamados sistemas artificiais, como máquinas (um automóvel, um computador, um liquidificador etc.), sistemas de transporte (estradas, aeroportos, veículos etc.), sistemas sociais (clubes, empresas, sindicatos etc.), ramos do conhecimento (administração etc.).

Sob um ponto de vista mais amplo dos sistemas, é usual incluir seu relacionamento com o ambiente em que está contido. Os sistemas biológicos foram os primeiros a serem estudados, servindo como ponto de partida para posteriores evoluções para outros campos do conhecimento. Exemplo: o termo anatomia, referente às partes construtivas do sistema de fisiologia, sendo o estudo do funcionamento dessas partes. Um sistema artificial é intencionalmente criado como um conjunto de partes, elementos ou componentes inter-relacionados (os subsistemas) que visa a realização de determinado objetivo, no ambiente em que está inserido. Para alcançar os objetivos do sistema, cada subsistema tem seu objetivo próprio e se acha conectado a outro subsistema, do qual recebe os insumos ou entradas.

Considerando a percepção que passamos a ter do que vem a ser um sistema, vamos agora iniciar a caracterização dos sistemas de informação com outra reflexão: por que estudá-los?

Primeira razão: como sociedade, estamos envolvidos em uma competição econômica global por recursos, mercados, negócios com outras nações etc.

Para Adam Smith, economista escocês do século XVIII que iniciou o estudo da economia moderna com seu livro "*A Riqueza das Nações*", a riqueza de uma nação dependia de como a sociedade organizava a produção em suas fábricas nacionais.

Hoje está claro que nossa sociedade terá de organizar mercados globais, corporações internacionais e forças de trabalho multinacionais, se quiser manter e melhorar nosso padrão de vida. Portanto, precisamos de um sistema de informação para fazer isso com eficiência e sucesso.

Segunda razão: para atingir níveis mais altos de eficácia e produtividade em nossas fábricas e escritórios, precisamos de uma ampla compreensão sobre sistemas de informação. Será simplesmente

impossível operar com eficiência mesmo uma pequena empresa sem investimentos significativos em sistemas. Os desafios colocados por novos clientes, concorrência, tecnologia, condições econômicas, regulamentações governamentais e aspectos sindicais pedem muitos tipos de mudança, tais como técnicas mais aperfeiçoadas de produção, novos produtos e serviços, novos sistemas administrativos e novas habilidades dos empregados. Você deve saber como identificar problemas e oportunidades e como usar os sistemas de informação para aumentar a capacidade de reação da organização.

Terceira razão: sua eficácia como profissional ou empresário - na realidade, sua carreira e sua renda - dependerá de como você se dedica à tarefa de compreender os sistemas de informação. Se você deseja ser um artista gráfico, um músico profissional, um advogado, um administrador de empresas ou dono de um pequeno negócio, você vai trabalhar com e por sistemas de informação.

Antes de explorar o crescente papel dos sistemas de informação ao longo deste estudo, devemos primeiro definir um sistema de informação e seus componentes básicos. Antes disso, a título de pré-requisito para o entendimento dos conceitos básicos, vamos estudar o que vem a ser uma "informação".

Conceito de informação

A informação representa um conceito central de uma organização. Para ser um gerente eficiente, independente da área de negócios, é fundamental entender que a informação é um dos recursos mais valiosos e importantes de uma organização. Contudo, este termo é frequentemente confundido com o termo dados. Os dados consistem em fatos não trabalhados. Exemplos: nome de um empregado, número de peças num estoque, pedido de vendas etc. A Tabela 3.1 mostra vários tipos de dados que podem ser usados para representar esses fatos.

Dados	Representados por
Dados alfanuméricos	Números, letras e outros caracteres
Dados de imagem	Imagens gráficas ou fotos
Dados de áudio	Sons, ruídos, tons
Dados de vídeo	Imagem em movimento

Tabela 3.1 - Dados x Fatos.

Já a informação consiste em uma coleção de fatos organizados de modo que adquirem um valor adicional além do valor dos próprios fatos. Exemplo: um gerente em particular poderia entender que o total de vendas mensais está mais adequado a seu objetivo, ou seja, é mais valioso para ele do que o número de vendas de cada representante individual.

Dados representam as coisas do mundo real, tendo pouco valor além do valor de sua existência. Por exemplo, podemos equiparar os dados a pedaços de madeira. Verificamos que, nesse estado, a madeira possui um valor inerente a um simples objeto. Contudo, se alguma conexão for definida entre os pedaços de madeira, agrega-se valor a eles. Se empilharmos, por exemplo, os pedaços de madeira de determinada forma, eles poderão, por exemplo, ser utilizados como degraus de assento.

A transformação de dados em informação é um processo, ou seja, pertence a um conjunto de tarefas logicamente relacionadas e executadas para atingir a um resultado definido. O processo de definição de relacionamentos entre dados exige conhecimento. O conhecimento, por sua vez, representa a percepção e a compreensão de um conjunto de informações e de como essas informações podem ser úteis para uma tarefa específica. O ato de escolher ou rejeitar fatos, com base em sua relevância, para executar determinadas tarefas, está fundamentado num tipo de conhecimento pertinente ao processo de conversão de dados em informação. Portanto, considera-se a informação como dados que se tornaram mais úteis quando da aplicação do conhecimento. O conjunto de dados, regras, procedimentos e relacionamentos que precisam ser seguidos para agregar valor ou alcançar resultados adequados constitui a base de conhecimento.

Sistemas de Informação Baseados em Computadores (SIBC)

São sistemas formais, montados com a finalidade de resolver problemas importantes que surgem nas organizações. Um SIBC usa a tecnologia de computação para executar parte das funções de processamento de um sistema de informação e também algumas das funções de entrada e saída. Contudo, seria um erro descrever um sistema de informação apenas em termos de computadores. Um sistema de informação é uma parte integrante de uma organização, sendo um produto que envolve três componentes: tecnologia, organizações e pessoas, conforme mostra a Figura 3.1.

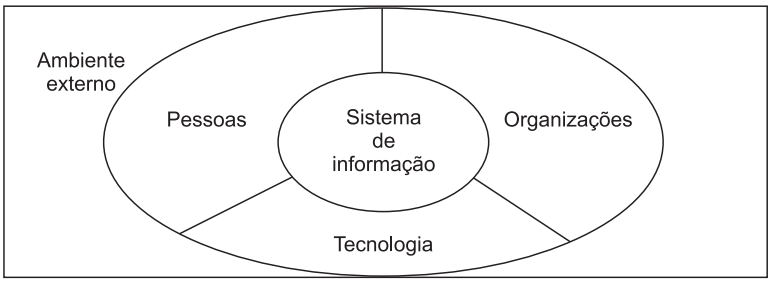


Figura 3.1 - Componentes dos sistemas de informação.

3.3 A importância do escopo

O escopo de um projeto de desenvolvimento de software possui várias definições. Sob o ponto de vista da análise de requisitos, podemos dizer que ele representa uma lista de funcionalidades que devem estar presentes no sistema de software. Quando dividimos o desenvolvimento do software em várias fases, e as fases em iterações, podemos denominar o escopo de desenvolvimento das iterações como uma *baseline* ou linha de base dos requisitos. Todo escopo, independente de representar o projeto como um todo ou as diversas iterações de desenvolvimento que podem pertencer a ele, deve possuir determinadas características, tais como:

- Funções ou recursos: que são úteis e novas para uma determinada versão do software;
- Considerar que a implementação é factível dentro do prazo estipulado para a versão do software;
- Ser aceito pelos interessados (*stakeholders*).

Considerando a definição apresentada anteriormente, o primeiro passo para a especificação do escopo do sistema de software é elaborar uma lista das funções que devem fazer parte dele, também conhecida como lista de requisitos, que representam as necessidades ou "desejos" do cliente. Após a lista ter sido feita, devemos definir alguns atributos para cada funcionalidade descrita, tais como prioridade, complexidade, risco, esforço etc. Esses atributos são importantes por diversas razões, apresentadas a seguir:

Prioridade

Esse atributo indica as funcionalidades mais relevantes para o usuário ou aquelas que servirão de base para outros recursos existentes no sistema. Podemos determinar categorias de prioridades que serão atribuídas a cada funcionalidade do sistema, tais como:

- **Prioridade crítica:** significa que a funcionalidade deve ser implementada na próxima versão do software, pois estamos considerando que o sistema não pode evoluir ou atender seus objetivos sem ela.
- **Prioridade importante:** atribuída a funcionalidades que devem ser incluídas na próxima versão, mas que podem ser postergadas caso exista um grande número de funcionalidades críticas pendentes ou por outras razões técnicas.
- **Prioridade útil:** atribuída quando observamos que determinada funcionalidade é desejada pelo usuário, mas sua utilização não vai afetar substancialmente a produtividade do sistema ou sua falta não será impedimento para o desenvolvimento do software em curso.

3.3.1 Medida de complexidade

É uma medida sobre a dificuldade de implementação relacionada a determinada funcionalidade desejada para o sistema de software. Na prática, esse atributo é utilizado como base para avaliar o esforço necessário ao desenvolvimento das funcionalidades que pertencem ao escopo do projeto. Podemos medir a complexidade das funcionalidades por diversas técnicas e métodos existentes. Uma das técnicas utilizadas é chamada de *Use Case Points* ou Pontos por Caso de Uso, que avalia a complexidade dos casos de uso com base, principalmente, no número de fluxos alternativos presentes. Independente da técnica utilizada, podemos sugerir alguns níveis de categorias para definir o grau de complexidade das funcionalidades que serão trabalhadas, conforme descrito a seguir:

- **Nível baixo:** normalmente atribuído a funcionalidades relacionadas à inclusão e à consulta de dados. É atribuído principalmente quando nos deparamos com telas de cadastro que possuem funções básicas de inserção, alteração, eliminação e consulta de registros.
- **Nível médio:** é atribuído quando, em determinada funcionalidade, predominam processos de transformação, transporte ou processamento de dados. Está relacionado com cálculos matemáticos, estatísticos ou interações com outros sistemas.
- **Nível alto:** atribuído quando percebemos que uma funcionalidade acumula as características existentes no nível de complexidade médio, acrescida de características consideradas críticas, como, por exemplo, o processamento de grandes volumes de dados, exigências relacionadas à segurança dos dados e ao alto desempenho etc.

- **Nível muito alto:** atribuído quando a funcionalidade acumular as características exigentes no nível de complexidade alto e também exigir um conhecimento altamente especializado para desenvolver essa funcionalidade.

3.3.2 Fatores técnicos e ambientais

A complexidade e a dificuldade em obter a tão almejada qualidade de software estão relacionadas diretamente com um conjunto de fatores técnicos e ambientais, presentes em todas as etapas do desenvolvimento de um software. Os fatores técnicos estão relacionados aos requisitos funcionais do sistema. Os fatores ambientais referem-se aos requisitos não funcionais associados ao processo de desenvolvimento. A relação entre esses dois fatores e o processo de desenvolvimento de software utilizado é intrínseca, uma vez que a qualidade do sistema está diretamente relacionada com a qualidade do processo de desenvolvimento escolhido.

Risco

A preocupação com os riscos que podem ocorrer durante o desenvolvimento do software é muito importante, mas muitas vezes negligenciada. Atribuir um risco a uma determinada funcionalidade representa uma medida de incerteza sobre a prioridade, complexidade e esforço necessários ao desenvolvimento de cada funcionalidade desejada para o sistema. Definir os riscos que podem ocorrer ajuda a identificar as funcionalidades que podem trazer problemas para o projeto e que, portanto, merecem atenção especial. O risco também contribui para a definição do escopo do projeto ou de uma iteração, uma vez que a equipe pode decidir primeiro pelo desenvolvimento das funcionalidades mais críticas ou mesmo tentar removê-las do escopo do projeto.

O risco que envolve o desenvolvimento de um software está relacionado com aspectos operacionais, contratuais e organizacionais. A título de exemplo podemos citar a incapacidade de fornecedores de responderem às exigências do projeto ou mesmo a falta de apoio para o projeto por parte da organização. Existem várias preocupações importantes relacionadas aos riscos em projetos de software, e a maior delas talvez seja riscos relacionados com obtenção de recursos orçamentários. Existem também riscos relacionados aos processos, que incluem tanto procedimentos técnicos quanto gerenciais. De uma maneira bem resumida, podemos dizer que qualquer coisa que ameace o bom andamento de um projeto pode ser considerado um risco. O gerenciamento de risco de software consiste em avaliar e controlar os riscos que podem afetar o projeto, o processo ou o produto de software. Podem ser gerenciados pelas seguintes atividades: identificação, análise, planejamento, acompanhamento e resolução dos riscos.

Esforço/tempo

O cálculo do esforço/tempo serve como base para o orçamento do sistema de software. Normalmente é medido em unidades que relacionam integrantes do projeto com o tempo, como, por exemplo, homens/mês (HM) ou homens/hora (HH). Em um projeto de software esse cálculo está intimamente ligado ao conceito de métricas, e uma das formas para mapear o tempo/esforço é a utilização de uma base histórica de projetos, que deve conter:

- o tamanho do projeto (estimado em pontos por função, por exemplo);
- a quantidade de horas utilizada na execução do projeto;
- o número de pessoas envolvidas com o projeto;
- e, principalmente, os artefatos que especificam a arquitetura do software.

Geralmente, quanto mais estivermos nas fases iniciais do desenvolvimento, mais tende-se a estimar de forma pouco precisa. Para estimarmos o tempo de desenvolvimento, com o objetivo de seguir com o cálculo de custo associados, podemos fazer uso de métodos como o COCOMO (*COnstructive COst MOdel*). Trata-se de um modelo de estimativa do tempo de desenvolvimento de um produto, criado por Barry Boehm.

3.4 Requisitos

Como vimos, a engenharia de software corresponde à análise, ao projeto, à construção, à verificação e à gestão de elementos técnicos ou sociais. É importante observar que independente do elemento a ser tratado, não podemos nos esquecer de responder às seguintes questões que devem ser levantadas inicialmente e respondidas:

- Qual é o problema que estamos querendo resolver?
- Que características do software serão usadas para resolver o problema?
- Como o software vai ser construído?
- Que abordagem será usada para descobrir erros que foram cometidos no projeto e na construção do software?
- Como o software será mantido a longo prazo, quando correções, adaptações e aperfeiçoamentos forem solicitados pelos usuários?

Uma metodologia também muito empregada é o ciclo PDCA (em inglês, Plan, Do, Check e Action). Corresponde a um ciclo de análise de melhoria, criado por Walter Shewhart, em meados da década de 1920, que acabou sendo disseminado pelo mundo por Deming. Trata-se de uma ferramenta gerencial de tomada de decisões com o objetivo de garantir o alcance das metas, também sendo de fundamental importância para a análise e melhoria dos processos organizacionais e para a eficiência do trabalho em equipe. Em cada ciclo dessa metodologia podemos usar adicionalmente um método conhecido como 5W2H, que representa as palavras em inglês: What (O quê), Who (Quem), Why (Por quê), When (Quando), Where (Onde), How (Como) e How Much (Quanto custa). Correspondem a perguntas básicas sobre questões importantes relacionadas ao levantamento de requisitos. A seguir resumiremos o que devemos refletir quando pensamos em cada uma dessas questões:

What: quais são as entradas do processo? Quais são as saídas? Quais são os indicadores? Quais são as metas? Quais são os recursos? Quais são os problemas? Quais são os métodos e tecnologias empregadas?

Who: quem é o cliente/usuário/beneficiário? Quem executa? Quem gerencia? Quem fornece? Quem participa das decisões?

Why: por quê/para que esse processo existe?

When: quando é planejado o processo? Quando é executado? Quando é avaliado?

Where: onde é planejado o processo? Onde é executado? Onde é avaliado?

How: como é planejado o processo? Como é executado? Como é avaliado? Como as informações são registradas e disseminadas? Como é avaliada a satisfação do cliente? Como está o desempenho do processo?

How much: quanto irá custar?

Portanto, a fase de levantamento de requisitos é composta por processos por meio dos quais devemos estabelecer o entendimento dos reais problemas dos patrocinadores do projeto (*stakeholders*) e necessidades dos usuários, objetivando a busca de soluções para estes problemas. A análise de requisitos é composta de algumas etapas importantes, descritas a seguir:

- Identificar quem são os patrocinadores do sistema de software;
- Buscar um acordo sobre a definição do problema a ser resolvido;
- Entender as raízes do problema;
- Determinar as fronteiras do sistema;
- Compreender as restrições que foram impostas à solução.

Stakeholders (patrocinadores do sistema)

O termo de origem inglesa "*stakeholders*" faz menção a uma pessoa, um grupo ou mesmo uma entidade que possui legítimos interesses nas ações e no desempenho de uma organização ou projeto. Em português também são conhecidos como patrocinadores, partes interessadas ou intervenientes. É um termo utilizado em diversas áreas, como administração e, como estamos observando, na engenharia de software. Representam, portanto, as partes interessadas que devem estar de acordo com as práticas de governança executadas pela empresa que representam ou que contratam.

Este termo foi utilizado pela primeira vez pelo filósofo Robert Edward Freeman. Segundo ele, os *stakeholders* representam elementos essenciais ao planejamento estratégico dos negócios. De uma maneira mais ampla, compreende todos os envolvidos em um processo, o que pode ser de caráter temporário (como um projeto) ou duradouro (como o negócio de uma empresa). A seguir observe alguns exemplos de possíveis *stakeholders*:

- Donos;
- Acionistas;
- Empregados;
- Investidores;
- Fornecedores;
- Governos Federal, Estadual e Municipal;

- Sindicatos;
- Associações;
- Grupos normativos;
- ONGs;
- etc.

Considerando o desenvolvimento de software, *stakeholders* serão aqueles que irão apoiar e viabilizar o desenvolvimento da solução proposta para o problema. Observe que, em diversas situações, os *stakeholders* podem ser representados pelos usuários do sistema. Já em outras, pode haver diversos *stakeholders*, cada qual com um objetivo e razões próprias. Temos que ter bastante atenção aos anseios dos *stakeholders*, pois disputas políticas entre diferentes *stakeholders* com diferentes visões sobre o que está sendo desenvolvido pode atrapalhar o desenvolvimento do software. O analista de sistemas deve ser capaz de lidar com as relações políticas entre os diversos *stakeholders*, objetivando administrar possíveis conflitos para buscar o sucesso do projeto. Devemos tomar cuidado ao refletir sobre os *stakeholders* envolvidos para não esquecermos nenhum. Por exemplo, se estivermos desenvolvendo um sistema de faturamento para determinada organização, mesmo o governo desconhecendo tal sistema, será um *stakeholder* indireto, pois o recolhimento de impostos sobre uma nota fiscal lhe interessa.

Como identificar stakeholders?

A identificação dos *stakeholders* é um passo importante, que tem por objetivo gerar uma lista de pessoas que irão influenciar no sucesso ou no fracasso do sistema. É importante mencionarmos nessa lista, além da identificação do *stakeholder*, os papéis por ele cumpridos. A seguir apresentamos algumas questões que nos inspiram a identificar tais elementos:

- Quem será o cliente do sistema?
- Quem serão os usuários do sistema?
- Quem será afetado pelas saídas oferecidas pelo sistema?
- Quem irá manter o sistema?
- Quem avaliará ou aprovará o sistema quando for concluído e entregue?

Como atingir um acordo sobre a definição do sistema?

É muito importante que todos os envolvidos no desenvolvimento do software (equipe e *stakeholders*) entrem em acordo sobre a definição do sistema. Para isso, basicamente temos que nos concentrar na definição do problema, descrevendo-o de forma sucinta. Inicialmente, a partir de uma descrição escrita, os *stakeholders* devem chegar a um acordo de que o problema existe e que é realmente relevante. Estabelecido este acordo, devemos agora buscar uma visão comum sobre a abordagem que poderia ser utilizada na solução pretendida. Normalmente, para formalizar essa visão comum, redigimos um documento que deixa clara a visão sobre o problema em conjunto com a solução proposta. Em algumas metodologias esse documento é denominado "documento de visão".

Como exemplo, podemos criar uma ficha de definição de requisitos, com o objetivo de auxiliar na definição do problema. Essa ficha deve conter pelo menos os seguintes itens:

- Descrição do problema;
- Identificação dos *stakeholders* afetados;
- Impacto do problema sobre os *stakeholders*;
- Solução proposta e seus principais benefícios.

Segue um exemplo de como essa ficha poderia ser preenchida:

Descrição: Pedidos de compra incorretos

Identificação dos *stakeholders* afetados: Departamento de vendas, Clientes, Departamento de entregas, SAC.

Impacto do problema sobre os *stakeholders*: Tem gerado muitas queixas, Aumenta o custo para reposição de mercadorias, Aumentou o índice de insatisfação dos clientes, Ocasiona diminuição da renda.

Solução proposta e seus principais benefícios: Desenvolver um novo sistema para resolver este problema, que deve incluir: Verificação dos pedidos no momento de sua realização; Relatórios de vendas para a gerência; Proporcionar aumento dos rendimentos.

Observar que o documento de visão pode ser elaborado tanto de uma forma mais técnica e complexa (como sugerido pelo Processo Unificado, por exemplo), como também pode ser elaborado de forma mais simples e, consequentemente, mais clara para o cliente, como ilustramos anteriormente. O documento de visão normalmente também é usado como parte do contrato de desenvolvimento, uma vez que seu conteúdo indica o que será feito, e o que não faz parte desse documento está fora do escopo do projeto.

Como entender as raízes do problema?

Essa etapa consiste em identificar as causas do problema em questão, uma vez que um sistema de software deve atacar as causas do problema para tentar resolvê-lo ou amenizá-lo. Para isso podemos utilizar, por exemplo, uma técnica bem conhecida, normalmente utilizada no controle de qualidade para a identificação das causas de um determinado problema. Ela se chama "Diagrama de Causa e Efeito", ou "Diagrama Espinha de Peixe", de Ishikawa.

Para utilizar esse diagrama, basicamente apresentamos o problema em uma linha, enquanto suas possíveis causas são desenhadas por setas. Observe que causas para as setas também podem ser desenhadas por outras setas, que apontam para as originais. O uso desse diagrama é indicado por ser muito simples e de fácil entendimento para os *stakeholders*, pois oferece uma visão clara e sistemática para a identificação dos problemas. A Figura 3.2 apresenta um exemplo de uso desse diagrama.

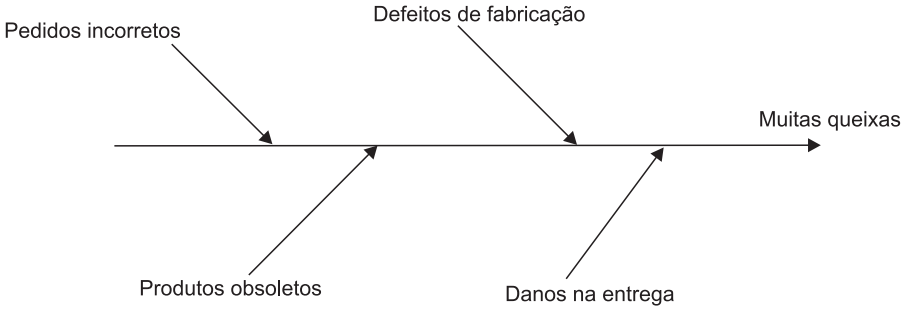


Figura 3.2 – Exemplo de diagrama de causa e efeito.

Como definir as fronteiras do sistema?

Devemos estabelecer as fronteiras do sistema a ser desenvolvido, pela identificação dos limites de atuação do sistema, considerando as operações manuais e outros sistemas existentes na empresa. É importante perceber que a definição dos limites do sistema apoia a identificação de potenciais integrações a serem desenvolvidas no projeto. O uso de diagramas simplificados pode ajudar na visualização desses limites, como ilustra a Figura 3.3, na qual a representação de "homens-palito" utilizada refere-se aos *stakeholders* envolvidos.

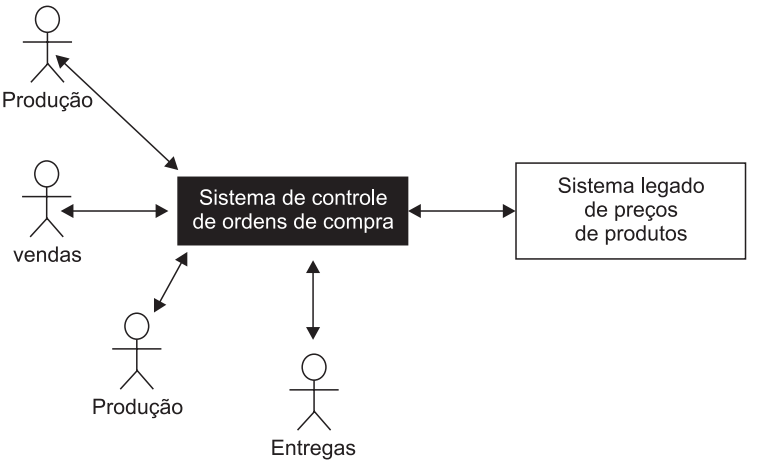


Figura 3.3 – Diagrama para ajudar na visualização das fronteiras do sistema.

Também podemos utilizar o próprio diagrama de causa e efeito para fazer um rascunho considerando os principais tipos de cortes das fronteiras do sistema, conforme ilustra a Figura 3.4.

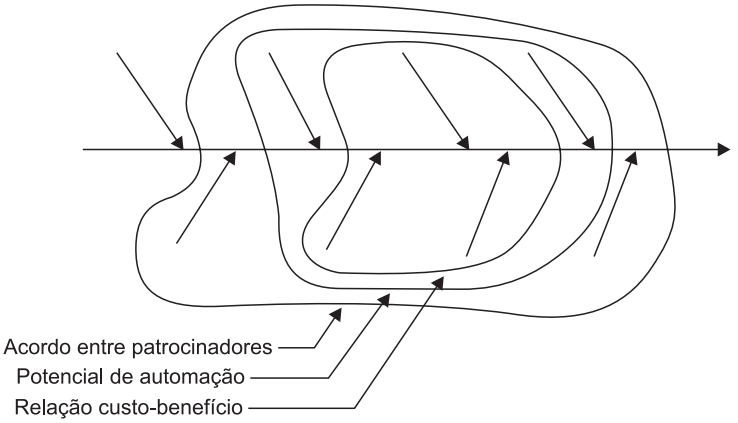


Figura 3.4 – Exemplo de tipos de cortes das fronteiras do sistema.

Identificação de restrições

A identificação das restrições consiste em listar as restrições organizacionais e técnicas que serão impostas para a solução. Consideramos como tipos de restrições, por exemplo, as econômicas, políticas, ambientais, de recursos, de tempo (cronograma), técnicas etc. Observe que restrições técnicas, neste caso, ainda não dependem de particularidades do projeto da solução proposta, mas sim de restrições de plataforma, tecnologia utilizada, sistemas de apoio, condições para homologação do sistema, entre outras exigências do cliente. Devemos tomar o cuidado para registrar o raciocínio utilizado que serviu de base para as restrições, objetivando evitar que elas sejam rediscutidas frequentemente, lembrando que o documento de visão também pode conter essas informações. A seguir apresentamos alguns exemplos de identificação de restrições, considerando o raciocínio (ou justificativa) para cada uma delas:

Restrição operacional

Descrição: Manter uma cópia exata da ordem de compra na base de dados legada.

Raciocínio: Durante o período de implantação do sistema o risco de perda de dados é grande, devendo ser necessário rodar sistemas em paralelo por um período de um ano.

Restrição tecnológica

Descrição: Um novo processo de desenvolvimento deve ser utilizado.

Raciocínio: Esta medida poderá trazer ganhos em termos de produtividade.

Restrição sistêmica

Descrição: O espaço no servidor utilizado para a aplicação deverá ser de 30 GB.

Raciocínio: Não dispomos de muito espaço no servidor.

3.4.1 Tipos de requisitos

Os requisitos devem indicar O QUE o sistema deve fazer em vez de COMO isso deve ser feito. Considerando essa preocupação e por residirem na fronteira entre os usuários e a equipe de desenvolvimento do sistema, os requisitos podem e devem ser classificados de acordo com dois pontos de vista:

- **Classificação sob o ponto de vista do usuário (ou seja, para o usuário):** que representa as necessidades do negócio, as necessidades do usuário (ou features) e os requisitos do sistema propriamente dito.
- **Classificação sob o ponto de vista dos desenvolvedores (ou seja, classificação técnica):** divide-se nos chamados requisitos funcionais e não funcionais.

Classificação sob o ponto de vista dos usuários

Requisitos de negócio: representam a descrição das necessidades que existem no negócio executado pelos usuários. Consideram, portanto, os objetivos, processos, papéis e outras características presentes na organização onde o sistema será utilizado. Pela leitura dos requisitos de negócio podemos observar, em um nível mais alto que os requisitos de usuário, quais serão as funções do sistema, uma vez que ele deve atender às necessidades do negócio. É importante que não consideremos para os requisitos de negócio aspectos técnicos, pois, como mencionamos, nosso foco neste momento é entender o que o sistema deve fazer e não como será feito.

Requisitos do usuário: são declarações de alto nível sobre as funções que o sistema deve oferecer, considerando também as restrições sobre as quais deve operar. Quando pensamos nos requisitos do usuário, devemos ter em mente quais serviços deverão ser fornecidos pelo sistema, que devem atender a uma ou mais necessidades dos *stakeholders*. Os requisitos dos usuários servem também como base para negociação entre cliente e fornecedor de software. Assim como recomendado para os requisitos de negócio, os requisitos do usuário também não devem se preocupar em como será feito e sim com o que deve ser feito.

Requisitos do sistema: oferecem uma especificação detalhada das funções descritas nos requisitos do usuário, tendo como objetivo principal guiar o processo de desenvolvimento do sistema. Oferecem, portanto, uma descrição mais detalhada dos requisitos do usuário, incluindo informações sobre que dados serão armazenados ou recuperados em cada um dos requisitos.

Classificação sob o ponto de vista dos desenvolvedores

Como mencionamos anteriormente, os requisitos são divididos em duas classificações técnicas, que ajudam os desenvolvedores a entender melhor o que deve ser implementado.

Requisitos funcionais: são descrições das funções que o sistema deve prover para o usuário, ligados ao domínio do problema. Aqui encontramos declarações sobre como o sistema recebe entradas, como gera saídas, como ele deve se comportar ao longo do tempo etc. Observe que existem requisitos evidentes, em que o usuário está ciente de que a função está sendo feita. Contudo, não podemos nos esquecer de requisitos "omitidos", ou seja, embora a função tenha de ser feita, ela é invisível aos olhos do usuário. Os requisitos funcionais representam as regras de negócio, ou seja, contêm as funcionalidades que o sistema deve atender.

Requisitos não funcionais: são compostos por restrições impostas sobre as funções oferecidas pelo sistema, ligadas ao domínio da solução. Os requisitos não funcionais podem ser encontrados quando refletimos sobre restrições que podem estar presentes em diferentes contextos, como exemplificado a seguir:

Requisitos de produto: são requisitos não funcionais que especificam o comportamento do sistema no que se refere a aspectos computacionais, como desempenho, confiabilidade, segurança etc.

Requisitos organizacionais: podem ser provenientes de políticas ou procedimentos adotados pela organização, como normas de atendimento ou qualquer outro padrão organizacional adotado.

Requisitos externos: são encontrados quando observamos o ambiente onde o sistema será executado, ou seja, fora do sistema e do processo de desenvolvimento.

3.4.2 Primeira pergunta: qual é o seu problema?

No item 3.4 discutimos aspectos básicos da necessidade de observarmos e identificarmos os diversos tipos de requisitos e do papel dessa importante fase do desenvolvimento de software. Neste tópico apresentamos considerações complementares relacionadas principalmente com técnicas que podemos utilizar para entendermos qual será, de fato, o problema que estamos tentando resolver ao estudarmos as necessidades dos clientes.

Antes convém salientar aspectos relacionados a quem estaria responsável por essa tarefa. Em uma equipe de desenvolvimento de software tradicional, normalmente temos o papel do analista de requisitos, que é o responsável pela identificação e definição do escopo do projeto ou de suas alterações. Contudo, atualmente muitas organizações contam com a colaboração do chamado analista de negócios, que tem sido um profissional cada vez mais demandado por colaborar com aspectos estratégicos da empresa. O analista de negócios trabalha na busca de melhores oportunidades de negócio, analisando tendências, criando produtos, estando sempre preocupado em encontrar novas oportunidades para a organização. Esse profissional atualmente é visto como uma função complementar a outros profissionais, como os analistas de processos ou analistas de sistemas, sendo que todos exercem funções complementares.

Já percebemos que podemos encontrar na literatura diversas definições sobre requisitos de software. Independentemente de como tais definições são expressas, todas elas convergem para o fato de os requisitos serem a base de comunicação conceitual para o entendimento do problema a ser resolvido. Também todas são unânimes em afirmar que os requisitos, evidentemente, proporcionam subsídios para a estruturação do software a ser desenvolvido. Também concordam com a importância de se realizar um levantamento de requisitos adequado, pois um problema mal analisado certamente proporcionará um resultado insatisfatório sob os olhos do cliente. Mas quais seriam, então, os procedimentos práticos para melhorar o entendimento do problema a ser resolvido?

Segundo Kendall (1992), quando levantamos requisitos, geralmente nos deparamos com duas importantes questões:

- Quais objetos de investigação devem ser identificados entre os diversos formulários, relatórios ou documentos observados, que foram gerados pelos membros da organização?

- Considerando o grande número de pessoas que podem ser afetadas pelo sistema a ser desenvolvido, quais devem ser observadas ou entrevistadas?

Considerando todas as possibilidades de uso de técnicas para o levantamento de requisitos, sempre temos de decidir sobre esses dois aspectos importantes relacionados a "o que" observar e a "quem" questionar. É importante também perceber por que as empresas desenvolvem ou compram sistemas de informação. Podemos resumir essa necessidade da seguinte forma:

- Empresas possuem processos para a realização de operações continuadas (lembrar que um projeto é temporário, relacionado à entrega de um produto ou serviço; a operação continuada pode estar relacionada, por exemplo, a um processo de fabricação).
- Empresas adquirem sistemas de informação, pois auxiliam na execução de atividades que compõem os processos utilizados.
- Empresas têm expectativa de que os sistemas de informação proporcionem uma execução mais rápida ou com menor consumo de recurso das atividades relacionadas aos processos.

Entender o problema a ser resolvido, além de passar pela percepção objetiva do que de fato está motivando seu cliente, não pode deixar de incluir um entendimento da estrutura geral da organização. Uma organização empresarial é complexa e formal, cujo objetivo é gerar produtos ou serviços com fins lucrativos. É claro que também existem organizações sem fins lucrativos (como órgãos do governo e organizações não governamentais). Para ajudar a entender as necessidades de uma organização, podemos fazer um exercício e imaginar que decisões se pode tomar ao decidirmos abrir o próprio negócio. Vamos observar então algumas possíveis preocupações:

- **Qual produto ou serviço você forneceria?** Essa decisão é chamada de escolha estratégica, porque ela determina o tipo de empregado de que você precisará, os métodos de produção, os temas para marketing e um conjunto de outros fatores.
- **De que tipo de organização você precisaria?** Inicialmente você teria de desenvolver alguma espécie de divisão de produção - um arranjo de pessoas, máquinas e procedimentos a serem seguidos para gerar o produto (que poderia ser um serviço).
- **Criação de um grupo ou divisão de vendas e marketing.** A função principal seria vender o produto ou serviço a um preço rentável.

Você também precisaria ter um grupo de finanças e contabilidade. Essas pessoas procurariam fontes de crédito ou verbas e controlariam as transações financeiras em andamento, tais como pedidos, compras, desembolsos e folhas de pagamento. Finalmente, precisaria de um grupo de pessoas para se dedicar a seleção, contratação, treinamento e motivação do pessoal que trabalharia para você. Em outras palavras, precisaria de um grupo de recursos humanos. A reflexão sobre como a empresa deve se organizar e agir encontra amparo no chamado *Business Plan* ou Plano de Negócio, documento imprescindível àqueles que desejam ser empresários ou como ferramenta de gestão. Trata-se de um documento vivo, pois deve ser constantemente atualizado. O plano de negócio também é utilizado como instrumento de marketing e comunicação para os investidores.

Empresas são organizações formais e consistem em unidades especializadas com uma divisão nítida de mão de obra e especialistas empregados e treinados para diferentes funções profissionais, como vendas, produção, recursos humanos, finanças etc. Além disso, as organizações são hierárquicas e estruturadas. Os empregados em uma firma são dispostos em níveis crescentes de

autoridades nos quais cada pessoa deve responder a alguém acima dela. Procedimentos formais ou regras para o cumprimento das tarefas coordenam grupos especializados na firma, de forma que eles completem seu trabalho de maneira aceitável. Por outro lado, diferentes níveis e especialidades em uma organização criam interesses e pontos de vista diferentes, que frequentemente conflitam entre si. Desses conflitos, políticas e eventuais compromissos vêm os sistemas de informação. As organizações precisam construir esses sistemas para resolver problemas criados por esses fatores internos e também por fatores externos, tais como mudanças em regulamentações governamentais ou condições de mercado.

Outro aspecto que não pode ser ignorado diz respeito aos colaboradores. As pessoas usam informações vindas de sistemas baseados em computadores em seus trabalhos, integrando-as ao ambiente de trabalho. Elas são solicitadas a introduzir dados no sistema para que o computador possa interpretá-los. Portanto, a interface com o usuário ou aquelas partes de um sistema de informação com as quais as pessoas devem interagir, tais como relatórios ou terminais de vídeo, também têm grande influência na eficiência e produtividade dos empregados.

Com base nessas necessidades podemos encontrar inspiração para caracterizar o problema do cliente e encontrar a melhor solução para resolvê-lo. Devemos considerar também que a maioria das organizações passa por diversos estágios. Primeiramente o trabalho é dividido em um grande número de empregados, o que permite a especialização. Na especialização, cada empregado se concentra em uma tarefa específica e se torna perito nela. Depois, uma hierarquia de subordinação e de relações de autoridade é desenvolvida para assegurar que o trabalho seja feito. Com o tempo, uma estrutura informal emerge em uma organização formal à medida que as pessoas se conhecem: festas de aniversários, nascimentos, novas contratações e aposentadorias são ocasiões para desenvolver relações informais e para fazer aparecer uma cultura da empresa. Observe que uma organização precisa tanto das relações informais quanto da cultura para ajudar a coordenar o trabalho e dar um sentido a ele. Portanto, saber em que estágio a organização se encontra também é necessário.

3.5 Análise

Como vimos, a análise do domínio de aplicação estudado fica mais fácil a partir do entendimento do problema a ser resolvido. Neste tópico, apresentamos algumas sugestões de como prestar atenção a aspectos funcionais que devem ser observados quando iniciamos o estudo da análise do domínio de aplicação.

Um dos casos mais típicos refere-se à manutenção de cadastros. Independentemente de qual seja esse cadastro, normalmente envolve as operações de inclusão, alteração, consulta e remoção de um determinado item existente no mundo real. Para identificar elementos que podem ser mantidos em cadastros, os analistas de sistema devem identificar elementos estáticos, que são aqueles cujas informações não costumam mudar ao longo do tempo. Esses elementos são comuns e existem para complementar os serviços que o sistema disponibilizará. Por falar nos serviços que devem ser prestados pelo sistema, devemos estar atentos a tudo que representa um serviço que existe no mundo real. Os serviços do sistema representam uma parte mais dinâmica. Portanto, para identificá-los, devemos observar as atividades realizadas no dia a dia do sistema real e os processos ou elementos afetados por essas atividades.

Outro motivo de preocupação diz respeito aos relatórios que devem ser gerados. Normalmente os clientes fornecem informações importantes sobre o conteúdo deles, mas é sempre bom estar atento a outras possibilidades que podem ser úteis. Para gerar relatórios, começamos com a identificação das saídas de informações consideradas relevantes. Podemos começar com os cadastros do sistema, seguindo pelo estudo dos serviços, não nos esquecendo de determinar os interessados por cada tipo de relatório definido. Como vimos anteriormente, as funcionalidades do sistema normalmente são registradas por meio dos casos de uso, por isso devemos tomar alguns cuidados ao analisar o domínio de aplicação e registrar o que observamos nos casos de uso.

No caso dos cadastros, por exemplo, normalmente assumimos que cada cadastro será composto de um caso de uso contendo quatro subfluxos (inclusão, alteração, exclusão e consulta). Se estamos observando serviços, transportamo-los para os casos de uso assumindo que cada serviço terá seu próprio caso de uso. Devemos observar também se existem operações complementares relacionadas aos serviços. Por exemplo, para cada serviço o analista de sistemas deve procurar operações que possam ser realizadas após a inclusão de um novo serviço. Podemos identificar essas operações traçando uma "linha do tempo" para as diversas operações do sistema. Os relatórios observados devem ser agrupados em poucos casos de uso. Devemos observar as afinidades entre eles ou agrupá-los por ator interessado. Quando analisamos um domínio de aplicação, independente da técnica de registro utilizada é preciso prestar atenção aos seguintes procedimentos:

- **Observar e inferir:** observar o comportamento dos usuários e inferir suas necessidades.
- **Discutir e formular:** discutir com os usuários as necessidades e, juntamente com eles, formular um entendimento comum dos requisitos.
- **Negociar:** começar com um conjunto padrão e negociar quais desses requisitos serão incluídos.
- **Estudar e identificar problemas:** identificar os requisitos que podem melhorar o produto.
- **Supor:** quando não existe usuário, é preciso usar a intuição.

Por fim, devemos refletir sobre todos os dados registrados e nos concentrar na consolidação das ideias. Para ajudar nessa etapa, a seguir encontram-se alguns exemplos de procedimentos a serem seguidos:

- As ideias devem ser discutidas, revisadas, organizadas e avaliadas.
- Os participantes podem concordar que uma ou mais ideias não são interessantes e descartá-las.
- Ideias remanescentes devem ser discutidas e classificadas segundo uma ordem de prioridade.

Frequentemente é necessário identificar:

- Requisitos absolutamente essenciais;
- Aqueles que são bons, mas não essenciais;
- Aqueles que seriam apropriados para uma versão subsequente do software.

3.6 Abstração e representação

De nada adianta utilizar procedimentos adequados para entender como devemos desenvolver um sistema de software se não registrarmos esse entendimento de uma maneira fácil de ser compreendida tanto para desenvolvedores quanto para os *stakeholders* (termo que referencia os principais interessados no projeto). Essa necessidade é absolutamente natural, uma vez que o mundo se apresenta extremamente complexo. Por isso, uma das principais formas que o ser humano encontrou para lidar com a complexidade é por meio do uso de abstrações.

As abstrações têm sido muito usadas, uma vez que constatamos que as pessoas normalmente tentam entender o mundo pela construção de modelos mentais de partes dele. A vantagem desses modelos é que representam uma visão simplificada de algo que está sendo observado, e a característica mais importante é que apenas elementos relevantes são considerados. Portanto, modelos mentais se mostram mais simples do que os complexos sistemas que eles modelam. Então, independente da maneira como a implementação de um sistema seja feita, depois de analisarmos o domínio de aplicação (que se refere ao cenário sob o qual o sistema de software estará inserido), devemos proceder à modelagem das entidades e fenômenos desse domínio, considerados relevantes para a aplicação. Essa tarefa é conhecida como modelagem conceitual, que basicamente é condicionada por dois aspectos:

- **Abstração:** diz respeito às operações mentais que executamos para observar um domínio e capturar sua estrutura em um modelo conceitual.
- **Representação:** refere-se às convenções de representação que adotamos para um modelo conceitual.

A Figura 3.5 ilustra esses passos.

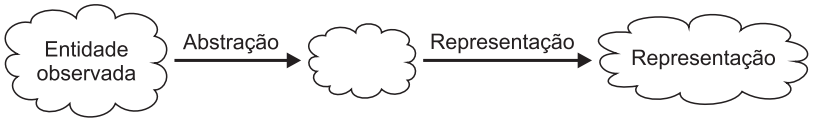


Figura 3.5 - Atividades básicas existentes no modelo conceitual.

A abstração pode ser utilizada em qualquer contexto, sempre que nos preocupamos em fragmentar e organizar cenários para melhor compreensão do todo. Com a abstração, o analista de sistemas consegue observar a realidade e dela abstrair uma série de elementos que servirão de base para o desenvolvimento do software, tais como entidades, ações etc. Note que os elementos abstraídos devem ser considerados essenciais para uma aplicação.

Ao aplicarmos o processo de abstração, devemos tomar o cuidado para não consideramos a implementação (codificação) do software, pois ainda não é o momento para pensar nas estratégias de codificação que serão usadas. No contexto do desenvolvimento de um software, existem duas formas de abstração tradicionais de grande importância e interesse dos desenvolvedores: a abstração de dados e a abstração de procedimentos. Para exemplificar, vamos analisar duas frases com base em um cenário acadêmico:

- Estudante é uma pessoa que está matriculada em uma universidade.
- Um curso é oferecido pela universidade.

No caso dos exemplos anteriores, em um primeiro momento devemos apenas nos concentrar em entender os principais elementos existentes e de interesse da aplicação, além da interface entre eles (no caso, estudante, universidade e curso). Em seguida, observamos a interface entre esses elementos abstraídos. Para isso, podemos nos basear nos verbos encontrados, pois podem indicar relacionamentos entre os elementos. No caso do exemplo, temos:

- Estudante MATRICULADO Universidade
- Curso OFERECIDO Universidade

Observe que os estudantes, a universidade e o curso foram abstraídos, pois além de se mostrarem importantes para o contexto do domínio de aplicação estudado, também possuem características próprias, conhecidas como atributos. No caso do estudante, certamente será necessário armazenar seu nome, RG, filiação etc. O curso também terá um nome, uma carga horária, assim como a universidade terá endereço, CNPJ etc. A preocupação com o modelamento do conteúdo a ser preservado referente aos elementos considerados importantes para o sistema chama-se abstração de dados.

Inicialmente vamos conhecer melhor as características da abstração de dados, a qual se refere ao processo mental por meio do qual nos concentramos em aspectos relevantes de um conjunto de objetos, desconsiderando as suas diferenças. Em outras palavras, a abstração de dados consiste em definir um tipo de dado com base nas operações que podem ser aplicadas aos principais elementos identificados.

Como exemplo, vamos pensar em uma serralheria. Considerando a matéria-prima utilizada e as operações necessárias, podemos observar que um tipo de dado "madeira" pode ser definido pelas operações "empilhar", colocar um elemento "madeira" sobre outro e "desempilhar", ou seja, retirar um elemento "madeira" que está no topo da pilha. Portanto, um objeto do tipo "madeira" só pode ser modificado por essas duas operações.

A abstração de procedimentos parte do princípio de que uma operação com efeito bem definido pode ser tratada pelos usuários do sistema como uma entidade única.

Para entender melhor, vamos agora imaginar um cenário corporativo, no qual podemos identificar um objeto chamado "funcionário" e uma operação que desejamos realizar com ele, denominada "calculo_do_salário_liquido". Neste caso, para os usuários do sistema, essa operação é considerada uma entidade única, mesmo que na realidade ela seja constituída de uma sequência de operações, tais como "calcular_IR", "calcular_ISS", "calcular_quinquenio" etc.

3.7 Métodos

Os métodos utilizados pela engenharia de software mostram como construir determinado software, o que normalmente inclui um amplo conjunto de tarefas que abrangem, por exemplo, a análise de requisitos, o projeto propriamente dito, a construção de programas, testes e manutenção. Podemos dizer que os métodos repousam em um conjunto de princípios básicos, responsáveis por reger cada área da tecnologia utilizada, incluindo também atividades de modelagem e outras técnicas descritivas.

3.8 Ferramentas

As ferramentas utilizadas pela engenharia de software têm o objetivo de fornecer apoio automatizado ou semiautomatizado para os processos e métodos. Chamamos de engenharia de software apoiada por computador (ou ferramenta CASE, do inglês Computer-Aided Software Engineering) o uso de ferramentas integradas, de modo que a informação criada por uma ferramenta possa ser usada por outra, caracterizando assim um sistema de apoio para o desenvolvimento do software. As ferramentas CASE combinam software, hardware e uma base de dados que apoia a engenharia de software. Observe que essa base de dados corresponde a um depósito que contém importantes informações sobre a análise preliminar, sobre o projeto, sobre a construção do software e informações sobre testes realizados.

Não há um padrão definido para a categorização das ferramentas CASE, contudo existem algumas terminologias utilizadas que ajudam a identificá-las, tais como:

- **Front End ou Upper Case:** são ferramentas que apoiam as etapas iniciais do desenvolvimento de sistemas (fases de planejamento, análise, projeto da aplicação etc.).
- **Back End ou Lower Case:** ferramentas que dão apoio à parte física do sistema, ou seja, codificação, testes, manutenção etc.
- **I-Case ou Integrated Case:** ferramentas que cobrem todo o ciclo de vida do software, desde os requisitos do sistema até o controle final da qualidade.

3.9 Exercícios

1) Por que a montagem baseada em componentes tem sido utilizada na indústria de software?

2) Como podemos definir um sistema? Essa denominação se aplica apenas à área de tecnologia da informação?

3) Qual a diferença entre dados e informação?

4) Todo escopo, independente de representar o projeto como um todo ou as diversas iterações de desenvolvimento que podem pertencer a ele, deve possuir determinadas características. Que características são essas?

5) Qual é o primeiro passo para a especificação do escopo do sistema de software? O que devemos fazer em seguida?

6) O que representa a medida de complexidade?

7) Qual é a diferença entre fatores técnicos e ambientais?

8) Por que devemos refletir sobre possíveis riscos relacionados à determinada funcionalidade que deve ser desenvolvida?

9) Como é medido o cálculo do esforço/tempo para sistemas de software?

10) Como podemos definir o termo *stakeholder*?

11) Como podemos definir as fronteiras do sistema a ser desenvolvido?

12) Cite exemplos de identificação de restrições, considerando o raciocínio (ou justificativa) para cada uma delas.

13) Qual é a diferença entre requisitos funcionais e não funcionais?

14) Quando analisamos um domínio de aplicação, independente da técnica de registro utilizada, é preciso prestar atenção a alguns procedimentos. Quais?

15) A modelagem conceitual de sistemas basicamente é condicionada por dois aspectos. Quais são e o que representam?

4

Metodologias de Desenvolvimento Tradicionais

Objetivos

- *Apresentar as características das metodologias de desenvolvimento tradicionais: modelo sequencial linear, modelo de prototipagem, modelo clássico, desenvolvimento incremental, desenvolvimento em espiral, IBM Rational Unified Process.*
- *Apresentar uma visão geral das metodologias PMBOK, SWEBOK e abordagem RAD.*
- *Paradigmas de desenvolvimento de software.*

Este capítulo apresenta uma visão geral das metodologias de desenvolvimento tradicionais, ou seja, aquelas que não estão relacionadas com paradigmas ágeis. O propósito desta abordagem é proporcionar ao leitor um parâmetro de comparação para que possa perceber posteriormente o diferencial proporcionado pelos paradigmas ágeis. Não é a pretensão esgotar todas as características que fazem parte do vasto universo das metodologias tradicionais. Apenas chamaremos a atenção para alguns pontos básicos e importantes. Vamos iniciar, portanto, com uma visão geral dos principais modelos utilizados, considerados tradicionais.

4.1 Modelo sequencial linear

Algumas vezes chamado de ciclo de vida clássico ou modelo em cascata, o modelo sequencial linear sugere uma abordagem sistemática sequencial para o desenvolvimento de software, que começa no nível de sistema e progride através da análise, projeto, codificação, teste e manutenção. A Figura 4.1 ilustra o modelo sequencial linear para a engenharia de software.

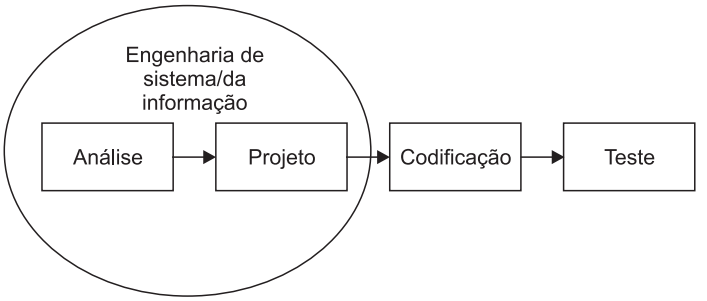


Figura 4.1 – Modelo sequencial linear para engenharia de software, segundo Pressman (2002).

Modelado segundo o ciclo convencional de engenharia, o modelo sequencial linear abrange as seguintes atividades:

4.1.1 Modelagem e engenharia do sistema

Como o software sempre faz parte de um sistema (ou negócio) maior, o trabalho começa pelo estabelecimento de requisitos para todos os elementos do sistema e depois pela alocação de algum subconjunto desses requisitos para o software. Essa visão de sistema é essencial quando o software precisa interagir com outros elementos, tais como hardware, pessoas e bases de dados. A engenharia e a análise de sistemas incluem um conjunto de necessidades em nível de sistema com um pouco de projeto e análise de alto nível. A engenharia de informação inclui um conjunto de necessidades estratégico e em âmbito da área de negócios.

4.1.2 Análise de requisitos de software

O processo de definição de requisitos é intensificado e focalizado especificamente no software. Para atender a natureza do(s) programa(s) a ser(em) construído(s), o engenheiro de software ("analista") deve conhecer o domínio da informação do software tanto quanto a função necessária, o comportamento, o desempenho e a interface. Os requisitos do sistema e do software são documentados e revistos com o cliente.

4.1.3 Projeto

O projeto de software é, na verdade, um processo de múltiplos passos que enfoca quatro atributos distintos do programa:

- Estrutura de dados
- Arquitetura do software
- Representações da interface
- Detalhes procedimentais

O processo de projeto traduz os requisitos para uma representação do software, que pode ser avaliada quanto à qualidade, antes que a codificação tenha início. À semelhança dos requisitos, o projeto é documentado e torna-se parte da configuração do software.

4.1.4 Geração de código

O projeto deve ser traduzido para linguagem de máquina. O passo de geração de código executa essa tarefa. Se o projeto é realizado de maneira detalhada, a geração de código pode ser realizada mecanicamente.

4.1.5 Teste

Uma vez gerado o código, o teste do programa tem início. O processo de teste focaliza os aspectos lógicos internos do software, garantindo que todos os comandos sejam testados, e os aspectos externos funcionais, isto é, conduz testes para descobrir erros e garantir que entradas definidas produzirão resultados reais, que concordam com os resultados exigidos.

4.1.6 Manutenção

O software inevitavelmente vai sofrer modificações depois de ser liberado para o cliente (uma possível exceção é o software embutido). Uma modificação acontece quando erros são encontrados, quando o software precisa ser adaptado para acomodar mudanças no seu ambiente externo (por exemplo, uma modificação necessária por causa de um novo sistema operacional ou dispositivo periférico), ou quando o cliente deseja melhoramentos funcionais ou de desempenho.

O suporte/manutenção do software reaplica cada uma das fases precedentes a um programa existente em vez de um novo programa. O modelo sequencial linear é o mais antigo e é o paradigma para engenharia de software mais amplamente usado. Todavia, a crítica do paradigma levou até mesmo seus atuais adeptos a questionar sua eficácia.

4.1.7 Problemas relacionados ao modelo sequencial linear

Existem problemas que são algumas vezes encontrados quando o modelo sequencial linear é aplicado. Por exemplo, projetos reais raramente seguem o fluxo sequencial que o modelo propõe. Apesar de o modelo linear poder acomodar interação, o faz indiretamente. Como resultado, modificações podem causar confusão à medida que a equipe de projeto prossegue. Outro problema encontrado é que, em geral, é difícil para o cliente estabelecer todos os requisitos explicitamente. O modelo sequencial linear exige isso e tem dificuldade de acomodar a incerteza natural que exige no começo de vários projetos.

Também podemos observar como problema do modelo sequencial linear o fato de que o cliente precisa ter paciência. Uma versão executável do(s) programa(s) não vai ficar disponível até o projeto terminar. Um erro grosseiro pode ser desastroso, se não for detectado até que o programa executável seja revisto.

4.2 Modelo de prototipagem

O cliente frequentemente define um conjunto de objetivos gerais para o software, mas não identifica detalhadamente requisitos de entrada, processamento ou saída. Em outros casos, o desenvolvedor pode estar inseguro da eficiência de um algoritmo, da adaptabilidade de um sistema operacional ou da forma que a interação homem/máquina deve assumir. Nessas e em muitas outras situações, um paradigma de prototipagem pode oferecer a melhor abordagem. O paradigma de prototipagem, esquematizado na Figura 4.2, começa com a definição de requisitos.

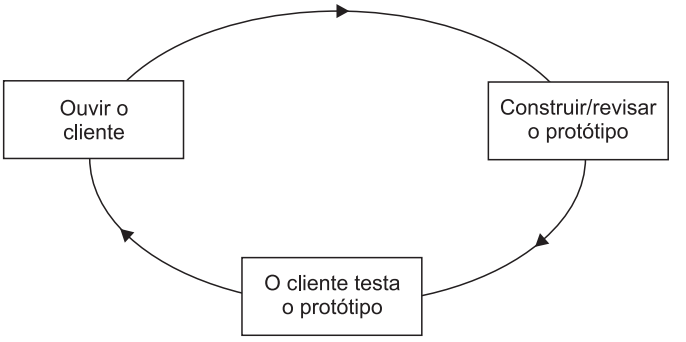


Figura 4.2 - Visão geral do paradigma de prototipagem, adaptado de Pressman (2002).

O desenvolvedor e o cliente encontram-se e definem os objetivos gerais do software, identificam as necessidades conhecidas e delineiam áreas que necessitam de mais definições. Um "projeto rápido" é então realizado. Esse projeto concentra-se na representação daqueles aspectos do software que vão ficar visíveis ao cliente/usuário (por exemplo, abordagens de entrada e formatos de saída). O projeto rápido parte de um protótipo. O protótipo é avaliado pelo cliente/usuário e usado para refinar os requisitos do software que será desenvolvido. Interações ocorrem à medida que o protótipo é ajustado para satisfazer as necessidades do cliente, enquanto, ao mesmo tempo, permitem ao desenvolvedor entender melhor o que precisa ser feito.

Idealmente, o protótipo serve como um mecanismo para a identificação dos requisitos do software. Se um protótipo executável é elaborado, o desenvolvedor tenta usar partes de programas existentes ou aplica ferramentas (por exemplo, geradores de relatórios) que possibilitam que programas executáveis sejam gerados rapidamente. O protótipo pode servir como "o primeiro sistema". Aquele que Brooks recomenda que se descarte. Mas esta pode ser uma visão idealizada.

É verdade que tanto clientes quanto desenvolvedores gostam do paradigma da prototipagem. Os usuários têm o sabor de um sistema real e os desenvolvedores conseguem construir algo imediatamente, todavia a prototipagem pode ser problemática pelas seguintes razões:

a) O cliente vê o que parece ser uma versão executável do software, ignorando que o protótipo apenas consegue funcionar precariamente ("mantido em pé com goma de mascar e arame"), sem saber que na pressa de fazê-lo rodar ninguém considerou a sua qualidade global ou manutenibilidade a longo prazo. Quando informado de que o produto deve ser refeito de modo que altos níveis de qualidade possam ser atingidos, o cliente reclama e exige alguns "consertos", para transformar o protótipo num produto executável. Em geral, a gerência de desenvolvimento de software concorda.

b) O desenvolvedor frequentemente faz concessões na implementação a fim de conseguir rapidamente um protótipo executável. Um sistema operacional ou uma linguagem de programação inapropriada podem ser usados simplesmente por estarem disponíveis e serem conhecidos; um algoritmo ineficiente pode ser implementado simplesmente para demonstrar uma possibilidade. Passado um certo tempo, o desenvolvedor pode ficar familiarizado com essas escolhas e esquecer todas as razões por que elas eram inadequadas. A escolha muito aquém da ideal tornou-se agora parte integral do sistema.

Apesar de problemas poderem ocorrer, a prototipagem pode ser um paradigma efetivo para a engenharia de software. O importante é definir as regras do jogo no início, isto é, o cliente e o desenvolvedor devem estar de acordo que o protótipo seja construído para servir como um mecanismo para definição dos requisitos. E depois ele é descartado (pelo menos em parte), e o software real é submetido à engenharia com o objetivo de buscar qualidade e a manutenibilidade.

Observe que existem pontos de vista divergentes no que diz respeito ao fato de a prototipagem ser reconhecida como uma metodologia. Isso é sustentado, pois a prototipagem, cuja evolução é conhecida popularmente como "wireframe", não deveria ser considerada uma metodologia, mas um recurso que pode ser usado em qualquer metodologia, para confirmação do entendimento e da proposta do sistema com o cliente por meio de uma ferramenta visual.

4.3 Modelo clássico

As metodologias tradicionais, também conhecidas como "pesadas" ou orientadas a documentação, tiveram sua origem em um cenário de desenvolvimento de software muito diferente do atual, onde imperava o uso de *mainframes* e dos chamados "terminais burros". Naquela época, não existia, por exemplo, o apoio de ferramentas de desenvolvimento de software que possuem depuradores ou analisadores de código, o que fazia com que alterações ou correções necessárias tivessem um custo muito alto. Foi devido a isso que existia toda uma grande preocupação com o planejamento e a documentação do projeto antes da sua implementação. A principal metodologia utilizada nessa época era o chamado Modelo Clássico ou Sequencial, apesar de ter suas origens há mais de quarenta anos.

O modelo clássico foi o primeiro processo publicado de desenvolvimento de software, composto de um modelo de fácil entendimento, pois se baseia em uma sequência de etapas, e cada etapa tem associada ao seu término uma documentação padrão que deve ser aprovada para que se inicie a etapa imediatamente posterior. A Figura 4.3 apresenta uma visão das etapas que compõem o modelo clássico. O formato dessa famosa figura também originou uma outra denominação para esse modelo, conhecido por muitos como "Modelo em Cascata".

O modelo clássico tem sido um dos paradigmas mais amplamente utilizados na engenharia de software, contudo sua aplicabilidade tem sido questionada devido a alguns problemas que surgem quando utilizamos esse modelo. Uma dos problemas observados é que, na verdade, os projetos raramente seguem o fluxo sequencial proposto pelo modelo. A interação é sempre necessária e, quando presente, cria problemas na aplicação do modelo. Outra questão é que é naturalmente difícil para o cliente especificar os requisitos completamente, acarretando uma incerteza natural no início do projeto. Outra questão é que o cliente deve ser paciente, pois segundo o modelo clássico, ele só verá uma versão funcional no final do desenvolvimento. Desta forma, qualquer erro ou mal-entendido, se não for detectado durante o desenvolvimento do software, pode ser desastroso.

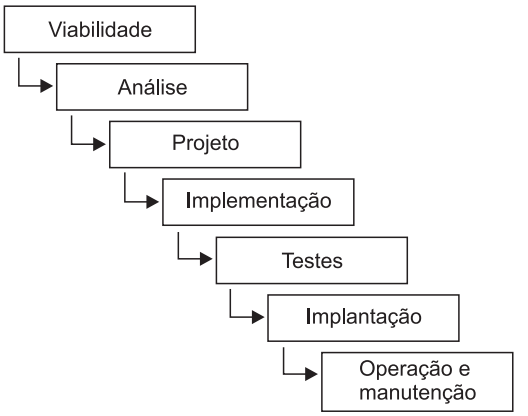


Figura 4.3 - Modelo clássico.

4.4 Desenvolvimento incremental

O desenvolvimento incremental combina elementos do modelo sequencial linear com elementos da filosofia interativa da prototipagem. A ideia básica de um modelo incremental é a aplicação de sequências lineares ao longo do tempo de desenvolvimento. Cada uma dessas sequências lineares deve produzir um incremento do projeto que está sendo desenvolvido, ou seja, dividimos todas as funcionalidades pretendidas e as agrupamos em sequências de atividades. Ao final de cada sequência linear, devemos sempre ter um produto pronto, que é parte integrante do projeto como um todo. Observe que durante o desenvolvimento de uma sequência linear incorporamos as estratégias utilizadas pelo paradigma de prototipagem.

As iterações representam, portanto, uma parte do projeto que está sendo desenvolvido. Mas por que iterar? Como pudemos perceber no modelo clássico, os projetos tradicionalmente são organizados para percorrer cada disciplina em sequência apenas uma vez. O ciclo de vida iterativo tem sido visto como uma maneira mais flexível de desenvolvimento, uma vez que é possível percorrer várias vezes as diversas disciplinas de desenvolvimento, construindo assim um melhor entendimento dos requisitos, contribuindo para o planejamento de uma arquitetura robusta, entre outros benefícios. Sob o ponto de vista do desenvolvimento, esse ciclo de vida é uma sucessão de iterações, por meio das quais o software se desenvolve de maneira incremental.

Quando o primeiro incremento é concluído, uma análise desse núcleo do produto deve ser feita, objetivando perceber se os requisitos básicos foram satisfeitos. Pode ser que nesse momento características funcionais suplementares (algumas já conhecidas, outras não) tenham de ser incorporadas ao projeto. Então um novo plano de desenvolvimento para o próximo incremento deve ser elaborado, como resultado do uso ou avaliação realizada. Esse novo planejamento tem por objetivo alterar o núcleo do produto para melhor satisfazer às necessidades do cliente e a elaboração de características e funcionalidades adicionais. Esse processo deve ser repetido após a realização de cada incremento, até que o produto completo seja produzido. A Figura 4.4, adaptada de Pressman (2002), ilustra as etapas e os processos descritos.

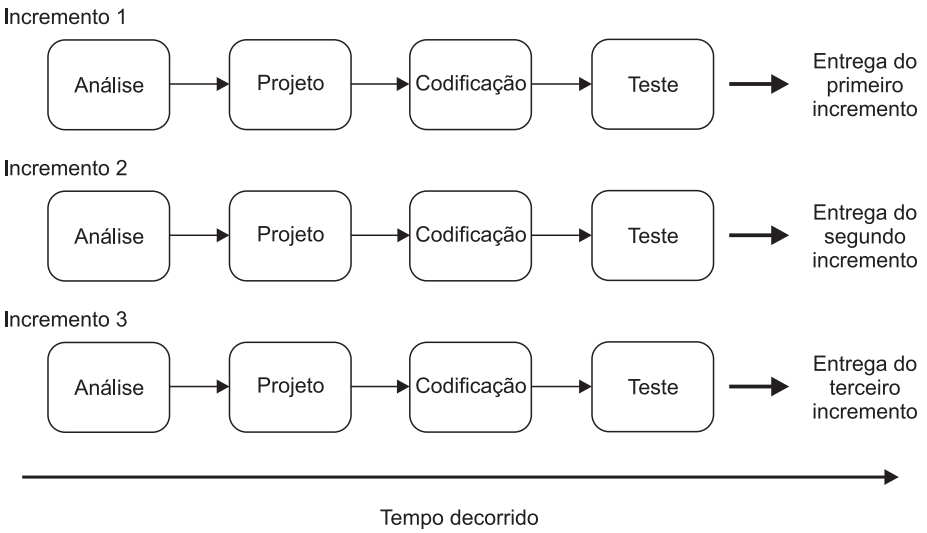


Figura 4.4 - Etapas e processos do modelo incremental.

Como podemos observar, o modelo incremental é interativo por natureza, mas ao contrário da prototipagem, ele objetiva a elaboração de um produto operacional a cada incremento. Também podemos considerar que ao final de cada incremento estamos usando um protótipo parcial. Podemos perceber que os primeiros incrementos são versões simplificadas do produto final, mas oferecem capacidades que servem aos usuários.

Desta forma, o desenvolvimento incremental é particularmente útil quando não dispomos de mão de obra para uma implementação completa, dentro do prazo de entrega estabelecido para o projeto. Os primeiros incrementos podem então ser implementados com menos recursos humanos, adicionando recursos à medida que os incrementos forem desenvolvidos e aceitos pelo usuário final. Outra vantagem dos incrementos é que eles podem ser planejados para gerir melhor os riscos técnicos.

4.5 Desenvolvimento em espiral

O modelo espiral foi inicialmente proposto por Boehm, em 1986, sendo um processo de desenvolvimento de software evolucionário que combina a natureza iterativa da prototipagem com aspectos controlados e sistemáticos do modelo sequencial linear.

Esse modelo fornece potencial para o desenvolvimento rápido de versões incrementais de software. Pelo uso do desenvolvimento espiral, o software pode ser desenvolvido numa série de versões incrementais. Durante as primeiras iterações, podemos produzir, por exemplo, protótipos, até que nas últimas iterações possamos produzir versões cada vez mais completas do sistema. O modelo espiral é desenvolvido em atividades que formam uma estrutura, também chamada de regiões de tarefa.

Segundo Pressman (2001), geralmente há de três a seis regiões de tarefa, conforme descrito a seguir:

- **Comunicação com o cliente:** tarefas necessárias para estabelecer efetiva comunicação entre o desenvolvedor e o cliente.
- **Planejamento:** tarefas necessárias para definir recursos, prazos e outras informações relacionadas ao projeto.
- **Análise de risco:** tarefas necessárias para avaliar os riscos, tanto técnicos quanto gerenciais.
- **Engenharia:** tarefas necessárias para construir uma ou mais representações da aplicação.
- **Construção e liberação:** tarefas necessárias para construir, testar, instalar e fornecer apoio ao usuário (por exemplo, documentação e treinamento).
- **Avaliação pelo cliente:** tarefas necessárias para obter realimentação do cliente, com base na avaliação das representações do software criadas durante o estágio de engenharia e implementadas durante o estágio de instalação.

Cada uma das regiões de tarefa mencionadas é preenchida por um conjunto de tarefas, que devem ser adaptadas às características do projeto a ser desenvolvido. Para projetos de menor porte, a quantidade de tarefas deve ser reduzida, diminuindo a formalidade. Já em projetos maiores, cada região de tarefas deve conter mais tarefas de trabalho, definidas com o objetivo de alcançar um alto nível de formalidade.

À medida que esse processo evolucionário tem início, todos os envolvidos com o projeto, ou a equipe de engenharia de software, devem ser mover em volta da espiral, sempre no sentido horário, a partir do seu centro. Desta forma, o primeiro circuito em torno da espiral poderia, por exemplo, resultar no desenvolvimento da especificação do produto. Dando continuidade a esse processo, passagens subsequentes em torno da espiral podem ser usadas para o desenvolvimento de um protótipo e, em seguida, de maneira progressiva, versões mais sofisticadas e complementares do software podem ser desenvolvidas.

Observe que cada passagem pela região de planejamento deve resultar em ajustes no plano de desenvolvimento do projeto. Com base na realimentação derivada de avaliações realizadas pelo cliente, também devemos ir ajustando o custo e o cronograma do projeto, o que pode levar a ajustes na quantidade de iterações necessárias.

De maneira diferente dos modelos clássicos de processo, que terminam apenas quando o software é entregue, o modelo espiral pode ser adaptado ao longo do ciclo de desenvolvimento do software. Por ter essas características, esse modelo é uma abordagem realística para o desenvolvimento de sistemas de software de grande porte, pois tanto os desenvolvedores quanto o cliente podem entender melhor e reagir aos riscos em cada nível evolucionário, à medida que o software evolui. É interessante notar que o modelo espiral usa a prototipagem como mecanismo de redução de riscos, mas o mais importante é que permite ao desenvolvedor aplicar a prototipagem em qualquer estágio de evolução do produto. Também permite manter a abordagem sistemática passo a passo, sugerida pelo ciclo de vida clássico, mas incorporada a uma estrutura iterativa, possibilitando refletir de maneira mais realística o mundo real.

Como ocorre com outros paradigmas, às vezes pode ser difícil convencer os clientes (principalmente em situações envolvendo contratos) de que a abordagem evolucionária é controlável. Para tentar minimizar essa impressão, é necessário apresentar uma competência considerável na avaliação de riscos, pois se um risco importante não for descoberto e gerenciado, problemas graves podem ocorrer. A Figura 4.5 ilustra a proposta de desenvolvimento em espiral.

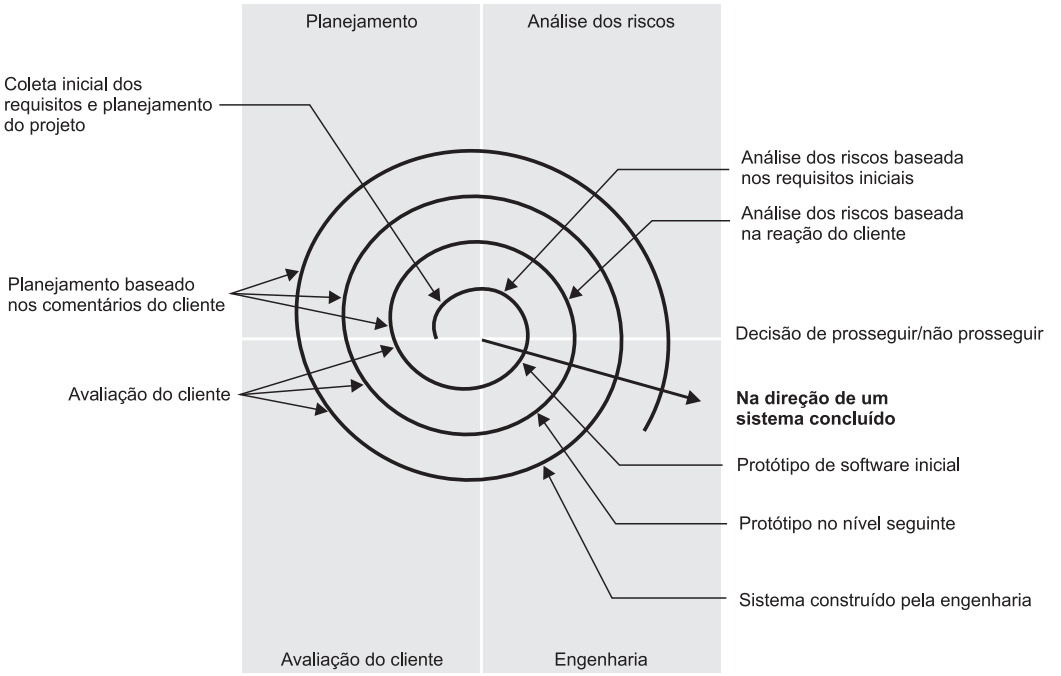


Figura 4.5 - Modelo espiral.

Observe que o modelo em espiral combina a iteratividade da prototipagem com os aspectos controlados e sistemáticos do modelo sequencial linear. O modelo espiral pode utilizar incrementos (iterações), uma vez que a cada ciclo, versões progressivamente mais completas do software são construídas.

4.6 IBM Rational Unified Process®

O IBM Rational Unified Process®, ou RUP®, é uma plataforma de processo de desenvolvimento de software configurável que oferece melhores práticas comprovadas e uma arquitetura configurável. O RUP® foi inicialmente criado pela Rational Corporation, que foi adquirida pela IBM®. Inicialmente conhecido como Processo Unificado, ou UP (do inglês, *Unified Process*), é um processo de engenharia de software que prega a prática de uma disciplina em que tarefas e responsabilidades são atribuídas em organizações que desenvolvem produtos de software. Seu objetivo, assim como ocorre com outros paradigmas da engenharia de software, é garantir que a produção de software tenha alta qualidade e que esteja de acordo com as necessidades dos usuários finais. O IBM® RUP possui duas dimensões:

Eixo vertical: representa o núcleo do fluxo de trabalho (workflow) do processo, contendo grupos de atividades logicamente distribuídas.

Eixo horizontal: representa o tempo e apresenta os vários aspectos do ciclo de vida do projeto.

A primeira dimensão (eixo vertical) contém os aspectos estáticos do processo:

- Descrição dos componentes do processo
- Fluxos de trabalho
- Artefatos
- Papéis

Os workflows previstos no eixo vertical são: Modelamento de Negócio, Requisitos, Análise e Design, Implementação, Teste, Entrega, Configuração e Gerenciamento de Mudanças, Ambientes, portanto o eixo vertical apresenta aspectos estáticos, como a descrição dos componentes do processo, fluxos de trabalho, artefatos utilizados, papéis (responsabilidades) etc.

A segunda dimensão (eixo horizontal) contém os aspectos dinâmicos do processo, ou seja, a maneira como são ordenados e expressos em termos de fases, repetições e marcos. Esse eixo representa o tempo e indica os vários aspectos do ciclo de vida do projeto. O gerenciamento do ciclo de vida do software no RUP se divide no tempo em quatro fases sequenciais, cada uma concluída por um marco. Cada fase, portanto, é essencialmente o tempo entre dois marcos. Em cada fase-extremidade uma avaliação é executada (atividade: revisão de marco do ciclo de vida) para determinar se os objetivos da fase anterior foram encontrados. Uma avaliação satisfatória permite que o projeto mova-se para a fase seguinte.

O IBM® RUP é hoje um dos mais consagrados processos concebidos para o desenvolvimento e manutenção de software. Trata-se de um processo de engenharia de software que prega a prática de uma disciplina em que tarefas e responsabilidades são atribuídas a organizações que desenvolvem produtos de software. O seu objetivo é garantir que a produção de software tenha alta qualidade e esteja de acordo com as necessidades dos usuários finais.

Os pontos fortes de sua abordagem são:

- Modelagem visual de artefatos e requisitos, análise e projeto;
- Decomposição funcional a partir de casos de uso;
- Especificação de testes funcionais e testes não funcionais em vários estágios (unidade, integração, sistemas e aceitação);
- Processo iterativo, levando a uma antecipação da análise e prototipagem de riscos arquiteturais e à reavaliação contínua do projeto;
- Amplo suporte de ferramentas integradas.

A abordagem iterativa utilizada pelo RUP® prevê a possibilidade de dividir uma fase em mais de uma iteração. Assim, para cada fase podemos ter um critério específico para determinar o número de iterações. As fases do RUP® estão baseadas no ciclo de vida em espiral, proposto por Boehm. A fase de concepção está localizada no centro da espiral, que cresce em direção à sua borda, onde se localiza a fase de transição. Assim, para cada iteração em cada uma das fases são realizados todos os estágios presentes no tradicional ciclo de vida em cascata, que compreende, basicamente, análise de requisitos, análise do sistema, projeto, implementação e testes.

4.6.1 Fase de iniciação

Nessa fase é importante medir o esforço do desenvolvimento, os riscos significativos do negócio e as exigências que devem ser cumpridas antes que o projeto possa prosseguir.

Principais objetivos

- Estabelecimento das condições, espaço e limite do projeto, incluindo uma visão operacional, critérios de aceitação e o escopo do projeto (visão).
- Discriminar os exemplos críticos do uso do sistema e os cenários preliminares da operação (recomenda-se definir pelo menos 20% dos casos de uso nesse momento).
- Exibir, e talvez demonstrar, ao menos uma arquitetura do projeto relacionada a alguns dos cenários preliminares que estimam o custo total e a programação para o projeto inteiro (protótipo).

Atividades essenciais

- Formular o escopo do projeto.
- Planejar alternativas de evolução para gerenciamento de riscos, equipes, planos de projeto, custos, cronograma etc.
- **Sintetizar uma arquitetura candidata:** observar que o esforço para preparar um protótipo nessa fase deve ser limitado. A solução deve ser elaborada durante as fases de elaboração e construção.
- **Preparar o desenvolvimento para o projeto:** selecionar ferramentas para cada uma das partes do projeto.

O foco da fase de iniciação está em como proporcionar o caminho para gerenciar os requisitos e como gerenciar o projeto.

4.6.2 Fase de elaboração

Principais objetivos

- Definir a arquitetura do sistema, provendo uma base estável para execução, desenho e implementação na fase de construção.
- Garantir que a arquitetura, requisitos e planos estejam estáveis o suficiente.
- Produzir um protótipo evolucionário dos componentes.
- Demonstrar que as plataformas arquiteturais vão suportar os requisitos do sistema, com custo e tempo razoável.

Atividades essenciais

- Definição e validação da arquitetura.
- Refinamento da visão, com base em novas informações obtidas durante essa fase.
- Criação de planos de iteração para a fase de construção.
- Refinar a arquitetura e solucionar componentes.

No fim da fase de elaboração, pretende-se que todos os processos e ferramentas escolhidos estejam em seu lugar. A parte mais crítica dessa fase é como realizar a configuração e o gerenciamento de alterações, uma vez que na fase de "construção", o trabalho será realizado por equipes atuando em paralelo.

4.6.3 Fase de construção (ou implementação)

Principais objetivos

- Clarear os requisitos remanescentes, completando o desenvolvimento do sistema.
- Minimizar o custo do desenvolvimento, otimizando recursos para evitar o sucateamento desnecessário e o retrabalho.
- Finalizar a análise, o design, o desenvolvimento e os testes de todos os requisitos funcionais.
- Observar se os softwares necessários, os ambientes de trabalho e todos os envolvidos estão prontos para o desenvolvimento.
- Definir degraus de paralelismo no trabalho de desenvolvimento dos times. Como em pequenos projetos, há componentes típicos que podem ser desenvolvidos independentemente de outros.

Atividades essenciais

- Gerenciamento de recursos, controle e otimização de processos.
- Desenvolvimento completo de componentes e testes de acordo com critérios de evolução definidos.
- Controle dos *releases* (versões) dos produtos com base nos critérios para aceitação da versão.

Na fase de construção, nenhum novo processo ou ferramenta é introduzido. O principal foco dessa fase é o desenvolvimento do produto, portanto o ambiente de desenvolvimento deve ser estável. Durante essa fase ainda há muito trabalho de design, principalmente nos primeiros ciclos da construção. Esse trabalho vai diminuir em interações de construção posteriores, uma vez que se relacionam com solicitações de mudanças. A descoberta e o refinamento de requisitos são mostrados como concluídos nesse estágio, e o esforço restante refere-se inteiramente ao gerenciamento de mudança. Portanto, observamos que o cliente ou demais *stakeholders* ainda têm uma participação importante durante essa fase.

4.6.4 Fase de transição (ou implantação)

Principais objetivos

- Garantir que o software esteja disponível para os usuários finais.
- Nessa fase podemos ter várias iterações, incluindo testes do produto e preparativos para liberação.
- No final dessa fase, os objetivos do projeto devem ter sido alcançados, sendo ele considerado fechado.
- A fase de transição começa quando a plataforma está madura o suficiente para ser entregue ao domínio do usuário final.
- Essa fase também deve produzir um beta teste (produto concluído, mas apenas para teste) para validar o novo sistema de acordo com as expectativas do usuário.
- Realizar operações paralelas relativas a sistemas legados (sistemas existentes na organização, anteriores ao que está sendo desenvolvido) que estão sendo substituídos.
- Realizar a conversão de bases de dados operacionais.
- Interagir com marketing, distribuição e força de vendas.
- Realizar atividades para registrar *bugs* (erros no sistema), estudos sobre performance e usabilidade.

É importante ressaltar que o beta teste e as atividades de registro de bugs, mencionados anteriormente, devem também ser feitos antes da fase de transição, justamente para garantir que o produto seja testado no cliente com o menor número de erros possíveis. Os testes realizados na fase de transição são complementares e necessários, pois muitas vezes não conseguimos reproduzir todo o ambiente do cliente na empresa que está desenvolvendo o software.

Atividades essenciais

- Finalizar material de suporte aos usuários finais.
- Testar os produtos entregues no *site* do cliente.
- Criar um *release* do produto.
- Obter um *feedback* do usuário final.
- Fazer o refinamento do produto com base no *feedback* recebido.
- Garantir que o usuário final receba o produto.

Na fase de transição, nenhum novo processo ou ferramenta é introduzido. As experiências acumuladas servem de subsídio para proporcionar processos e ferramentas para desenvolvimento da próxima evolução do produto.

4.6.5 Papéis, atividades e artefatos

Vamos agora fazer algumas considerações sobre os diversos papéis presentes no IBM® RUP, bem como as atividades que eles exercem e os artefatos que produzem. Antes disso, vamos a alguns conceitos básicos:

Atividades: são relatadas em artefatos. Uma atividade é uma unidade de trabalho de um indivíduo que responde por um determinado papel.

Artefatos: proporcionam uma entrada e saída para as atividades e o mecanismo pelo qual as informações se comunicam entre as atividades. Atividades possuem entrada e saída de artefatos. Um artefato é um produto do trabalho do processo. Artefatos são atividades sob a responsabilidade de um papel específico. Dão a ideia de que qualquer informação no processo, por mais insignificante que pareça, deve ser de responsabilidade de uma pessoa específica.

Papéis: são tipicamente realizados por um indivíduo ou um conjunto de indivíduos trabalhando em equipe. Um papel é uma definição abstrata de um conjunto de atividades realizadas.

Observação: pessoas que não estão presentes na organização podem estar desempenhando um importante papel, como, por exemplo, os *stakeholders*.

4.7 PMBOK (Project Management Body of Knowledge)

O "corpo de conhecimento de gerência de projetos", traduzindo do termo em inglês, é um guia de conhecimento e de melhores práticas para a profissão de gerência de projetos. Ele reúne o conhecimento comprovado internacionalmente relacionado a práticas tradicionais de gestão de projetos, incluindo também outras práticas inovadoras e mais avançadas.

É um guia genérico que serve para todas as áreas de projeto, independente de ser uma obra de construção civil, um processo de fabricação industrial ou a produção de software. Outro objetivo do PMBOK é a padronização de termos utilizados na área de gerência de projetos. O PMBOK é mantido pelo PMI (*Project Management Institute*), sediado na Pensilvânia, Estados Unidos, uma associação sem fins lucrativos composta por profissionais da área de gerenciamento de projetos. O PMI visa manter e ampliar o conhecimento sobre gerenciamento de projetos, assim como melhorar o desempenho de profissionais e organizações nessa área. Ele estabelece padrões, provê seminários, programas educacionais e certificações profissionais. De acordo com o PMBOK, os processos de gerência de projetos se resumem nos seguintes processos:

1) **Processos de iniciação:** reconhecem que um projeto ou fase deve começar e se comprometer com a sua execução.
2) **Processos de planejamento:** planejar e manter um esquema de trabalho viável para atingir aqueles objetivos de negócio que determinam a existência do projeto.
3) **Processos de execução:** coordenar pessoas e outros recursos para realizar o que foi planejado.
4) **Processos de controle:** assegurar, pela monitoração e avaliação do seu progresso, que os objetivos do projeto estão sendo atingidos, tomando ações corretivas quando for necessário.

5) **Processos de finalização:** finalizar a aceitação do projeto ou fase e realizar o seu encerramento de forma organizada.

O PMBOK é organizado em áreas de conhecimento, sendo cada uma descrita por processos. Cada área de conhecimento se refere a um aspecto a ser considerado dentro da gerência de projetos. A não execução de processos de uma área afeta negativamente o projeto, pois o projeto é um esforço integrado. As áreas que compõem o modelo proposto pelo PMBOK são:

- Qualidade
- Recursos humanos
- Escopo
- Aquisições
- Integração
- Comunicações
- Custo
- Riscos
- Tempo

Gerência da qualidade do projeto: são processos para assegurar que as necessidades que originaram o projeto sejam atendidas. Pode ser dividida em:

- **Planejamento da qualidade:** identifica os padrões de qualidade relevantes para o projeto, além de determinar a forma de satisfazê-los.
- **Garantia de qualidade:** avalia periodicamente o desempenho geral do projeto, buscando assegurar a satisfação dos padrões relevantes de qualidade.
- **Controle da qualidade:** monitora os resultados do projeto para determinar se estão de acordo com os padrões de qualidade, eliminando também as causas de desempenho insatisfatório.

Gerência dos recursos humanos do projeto: são processos necessários para proporcionar a melhor utilização das pessoas envolvidas no projeto. Divide-se em:

- **Planejamento organizacional:** identifica, documenta e designa funções, responsabilidades e relacionamentos do projeto.
- **Montagem da equipe:** objetiva conseguir que os recursos humanos necessários sejam designados e alocados ao projeto.
- **Desenvolvimento da equipe:** desenvolve habilidades individuais e do grupo para aumentar o desempenho do projeto.

Gerência do escopo do projeto: são processos necessários para assegurar que o projeto contemple todo o trabalho requerido, e nada mais que o trabalho requerido, para completar o projeto com sucesso. Divide-se em:

- **Iniciação:** autoriza o início do projeto ou de uma nova fase.
- **Planejamento do escopo:** desenvolve uma declaração escrita do escopo como base para decisões futuras do projeto.
- **Definição do escopo:** subdivide o projeto em componentes menores, tornando-os mais manejáveis.
- **Verificação do escopo:** formaliza e aprova o escopo do projeto.
- **Controle de mudanças de escopo:** controla as mudanças de escopo, envolvendo escopo do produto e escopo do projeto.

Gerência das aquisições do projeto: são processos necessários para a aquisição de mercadorias e serviços fora da organização que desenvolve o projeto. Esses processos estão organizados nas seguintes áreas:

- **Planejamento das aquisições:** determina o que e quando contratar.
- **Planejamento das solicitações:** documenta as necessidades de produtos ou serviços e identifica possíveis fornecedores.
- **Obtenção de propostas:** obtém propostas de fornecimentos conforme cada caso (cotações de preço, cartas-convite, licitações etc.).
- **Seleção de fornecedores:** escolhe os possíveis fornecedores.
- **Administração de contratos:** gerencia o relacionamento com os fornecedores.
- **Encerramento de contratos:** completa e liquida o contrato, incluindo a resolução de qualquer item pendente.

Gerência da integração do projeto: são processos necessários para assegurar que os diversos elementos do projeto sejam adequadamente coordenados. É composta por:

- **Desenvolvimento do plano do projeto:** agrega os resultados dos outros processos de planejamento, construindo um documento coerente e consistente.
- **Execução do plano do projeto:** conduz o projeto através das atividades nele incluídas.
- **Controle geral de mudanças:** coordena as mudanças por todo o projeto.

Gerência das comunicações do projeto: são processos necessários para assegurar que a geração, captura, distribuição, armazenamento e apresentação das informações do projeto sejam feitos de forma adequada e no tempo certo. A gerência das comunicações é dividida em:

- **Planejamento das comunicações:** determina que informações e comunicações são necessárias para as partes interessadas: quem, qual, quando e como será fornecida.
- **Distribuição das informações:** disponibilizar as informações necessárias para as partes interessadas de uma maneira conveniente.
- **Relato de desempenho:** coletar e disseminar as informações de desempenho, incluindo relatórios de situação, medição de progresso e previsões.
- **Encerramento administrativo:** gera, reúne e dissemina informações para formalizar a conclusão de uma fase ou de todo o projeto.

Gerência do custo do projeto: são processos necessários para assegurar que o projeto termine dentro do orçamento aprovado. Divide-se em:

- **Planejamento dos recursos:** determina quais recursos e quantidades de cada um devem ser usados para executar as atividades do projeto.
- **Estimativa dos custos:** desenvolve uma estimativa dos custos dos recursos necessários à implementação das atividades do projeto.
- **Orçamento dos custos:** aloca as estimativas dos custos do projeto aos itens individuais de trabalho.
- **Controle dos custos:** controla as mudanças no orçamento do projeto.

Gerência dos riscos do projeto: são processos necessários para a identificação, análise e resposta a riscos do projeto, divididos nas seguintes áreas:

- **Planejamento da gerência dos riscos:** decide como abordar e planejar as atividades de gerência dos riscos do projeto.
- **Identificação dos riscos:** determina quais riscos podem afetar o projeto e documenta suas características.
- **Análise qualitativa dos riscos:** analisa qualitativamente os riscos e as condições para priorizar seus efeitos nos objetivos do projeto.
- **Análise quantitativa dos riscos:** mede a probabilidade e as consequências dos riscos, além de estimar as implicações nos objetivos do projeto.
- **Planejamento das respostas aos riscos:** desenvolve processos e técnicas para aumentar as oportunidades e reduzir as ameaças dos riscos.
- **Controle e monitoração dos riscos:** monitora riscos residuais, identifica novos, reduz e avalia a efetividade por todo o projeto.

Gerência do tempo do projeto: são processos necessários para assegurar que o projeto termine dentro do prazo previsto. Divide-se em:

- **Definição das atividades:** identifica as atividades específicas que devem ser realizadas para produzir os diversos subprodutos do projeto.
- **Sequenciamento das atividades:** identifica e documenta as relações de dependência entre as atividades.
- **Estimativa de duração das atividades:** estima a quantidade de períodos de trabalho que serão necessários para a implementação de cada atividade.
- **Desenvolvimento do cronograma:** analisa a sequência e a duração das atividades, bem como os requisitos de recursos para criar o cronograma do projeto.
- **Controle do cronograma:** controla as mudanças no cronograma do projeto.

4.8 SWEBOK (Software Engineering Body of Knowledge)

Conforme mencionado anteriormente, o SWEBOK é um documento desenvolvido sob o patrocínio da IEEE com o objetivo de ser uma referência em assuntos relacionados à engenharia de software. A engenharia de software é dividida no SWEBOK em dez áreas, resumidas a seguir:

Gerência de engenharia de software: apresenta boas práticas para a gerência de projetos de desenvolvimento de software.

Gerência de configuração de software: identifica a configuração do sistema de modo a controlar sistematicamente suas mudanças e manter sua integridade e rastreabilidade durante o ciclo de vida do sistema.

Processo de engenharia de software: define, implementa, mede, gerencia, modifica e aperfeiçoa os processos de desenvolvimento de software.

Ferramentas e métodos: estimulam o uso de ferramentas de software que automatizam o processo de engenharia de software.

Qualidade de software: é um conjunto de atividades relacionadas com a garantia de qualidade de software (verificação e validação).

Requisitos de software: responsável por determinar boas práticas para a aquisição, análise, especificação e gestão de requisitos de software.

Design de software: utilizado para transformar os requisitos de software (estabelecidos em termos relevantes ao domínio do problema) em uma descrição que explica como solucionar os aspectos do problema relacionados com o software.

Construção de software: construção de programas funcionais e coerentes por meio da codificação, autovalidação e teste unitário.

Teste de software: recomenda verificações dinâmicas do comportamento do programa pelo uso de um conjunto finito de casos de teste contra o comportamento esperado destes.

Manutenção de software: atividades de suporte custo-efetivo a um sistema de software, que pode ocorrer antes e após a entrega do software.

Como sabemos, a computação não existe como uma entidade separada da matemática, por exemplo. Assim como acontece em toda a ciência, cada uma dessas divisões trabalha em conjunto com as demais. Normalmente utilizamos classificações para que possamos organizar o conhecimento e direcionar esforços. Todas as subáreas da engenharia de software proporcionam, de alguma forma, qualidade a programas de computador. Observando as áreas do SWEBOK apresentadas anteriormente, podemos perceber que ele inclui qualidade como uma área de conhecimento específica. Devemos tomar cuidado para não considerar essa área de qualidade estanque do restante do guia. Cada área do SWEBOK é subdividida em até dois níveis, formando uma estrutura hierárquica para catalogar os assuntos.

4.9 A abordagem RAD (Rapid Application Development)

O desenvolvimento rápido de aplicação *(Rapid Application Development, RAD)* é um modelo de processo de desenvolvimento de software incremental que enfatiza um ciclo de desenvolvimento extremamente curto. O modelo RAD é uma adaptação "de alta velocidade" do modelo sequencial linear, no qual o desenvolvimento rápido é conseguido pelo uso de construção baseada em componentes. Se os requisitos são bem compreendidos e o objetivo do projeto é restrito, o processo RAD permite a uma equipe de desenvolvimento criar um "sistema plenamente funcional", dentro de períodos muito curtos. A abordagem RAD, usada principalmente para aplicações de sistemas de informação, abrange as seguintes fases:

- Modelagem de negócio
- Modelagem de dados
- Modelagem de processos
- Geração da aplicação
- Teste e entrega

4.9.1 Modelagem de negócio

O fluxo de informações entre as funções do negócio é modelado de forma a responder às seguintes questões:

- Que informação dirige o processo de negócio?
- Que informação é gerada?
- Quem a gera?
- Para onde vai a informação?
- Quem a processa?

4.9.2 Modelagem de dados

O fluxo de informações, definido como parte da fase de modelagem, é refinado num conjunto de objetos de dados, que são necessários para dar suporte ao negócio. As características de cada objeto (chamadas atributos) são identificadas e as relações entre esses objetos são definidas.

4.9.3 Modelagem de processo

Os objetos de dados definidos na fase de modelagem dos dados são transformados para conseguir o fluxo de informação necessário para implementar uma função do negócio. Descrições do processamento são criadas para adicionar, modificar, descartar ou recuperar um objeto de dados.

4.9.4 Geração da aplicação

Em vez de criar software usando linguagens de programação convencionais, o processo RAD trabalha para reusar componentes de programas existentes (quando possível) ou criar componentes reusáveis (quando necessário).

4.9.5 Teste de entrega

Como o processo RAD enfatiza o reúso, muitos dos componentes de programa já devem ter sido testados. Isso reduz o tempo total de teste. Todavia, os componentes novos devem ser testados e todas as interfaces devem ser exaustivamente exercitadas. Obviamente, as restrições de tempo impostas num projeto RAD exigem "âmbito mensurável". Se uma aplicação comercial pode ser modularizada de modo a permitir que cada função principal seja completada em menos de três meses, é uma candidata ao RAD. Cada função principal pode ser tratada por uma equipe RAD distinta e depois integrada para formar um todo. Como todos os modelos de processos, a abordagem RAD tem desvantagens:

- Para projetos grandes, mas mensuráveis, o RAD exige recursos humanos suficientes para criar um número adequado de equipes RAD.

O RAD exige desenvolvedores e clientes compromissados com atividades continuamente rápidas, necessárias para obter um sistema completo num período muito curto. Se não houver esse compromisso por qualquer das partes, os projetos RAD falharão.

- Se for necessário um desempenho superior e esse desempenho tiver de ser conseguido ajustando as interfaces dos componentes do sistema, a abordagem RAD pode não funcionar.
- Quando riscos técnicos forem elevados, o RAD não é adequado. Isso ocorre quando uma nova aplicação faz uso intenso de uma nova tecnologia ou quando o novo software exige um alto grau de interoperabilidade com programas de computadores existentes.

4.10 Paradigmas de desenvolvimento de software

Independente da metodologia que será usada para o desenvolvimento de um sistema de software e da estrutura de hardware e software que a suportará, o primeiro grande desafio é a análise do domínio de aplicação e a subsequente modelagem das entidades e fenômenos relevantes que podemos observar. Essa tarefa é conhecida como modelagem conceitual e está condicionada basicamente a dois aspectos: a abstração e a representação.'

A abstração diz respeito às operações mentais que executamos para observar um domínio e capturar sua estrutura em um modelo conceitual. Já a representação refere-se às convenções que adotamos para um modelo conceitual. Como a abstração é uma etapa importante do processo de modelagem, não podemos deixar de abordar aspectos relacionados às técnicas envolvidas. É importante observar que a maneira pela qual decidimos abstrair informações do chamado "mundo real" aliada à técnica escolhida para representá-las em um modelo conceitual serve de base para determinar o paradigma (modelo) de desenvolvimento de software a ser utilizado.

4.10.1 O que é um paradigma?

O termo paradigma foi introduzido pela primeira vez pelo historiador Thomas Kuhn, em seu trabalho *A estrutura das revoluções científicas*, em 1962. Nesse trabalho, Kuhn se preocupava com as discrepâncias e os desentendimentos entre os cientistas no tocante à aplicação dos métodos e das técnicas utilizadas pela ciência. A partir daí ele propôs e desenvolveu a ideia de estruturas de conhecimento evolutivas e competitivas, as quais chamou de paradigmas. Como consequência, Kuhn observou que quando um paradigma é aceito pela maioria da comunidade científica, ele se torna um modo obrigatório de abordagem de problemas. Portanto, os paradigmas não têm apenas uma função explicativa, mas também uma função normativa, pois tendem a definir o que é e o que não é possível.

4.10.2 Como os paradigmas surgem?

Kuhn também observou que, de tempos em tempos, novas descobertas aparecem e não se ajustam às considerações prevalecentes. Os pesquisadores então começam a procurar uma forma alternativa para explicar ou compreender a realidade dos casos que eles observam. Eventualmente, uma nova maneira de conceituar o cenário estudado é apresentada como um modo melhor de compreender as coisas: surge então um novo paradigma.

4.10.3 Competência paradigmática

Definição: é a crença na existência de múltiplos paradigmas e na capacidade humana de entender e incorporar essa noção às práticas científicas e profissionais. Exemplo de mudança de paradigma: Copérnico revelou que nosso sistema planetário girava em torno do Sol, substituindo as ideias anteriores centradas na Terra.

Observação importante:

Os administradores que possuem competência paradigmática, ou seja, possuem o conhecimento de uma variedade de estruturas conceituais, serão capazes de julgar a aplicabilidade e o valor de diferentes formas de ver as coisas, de abordar e de analisar problemas. Ter competência paradigmática permite não ser prisioneiro de um único paradigma.

4.10.4 A importância dos paradigmas

É sempre importante refletirmos sobre a "veracidade" de nosso conhecimento em relação ao que percebemos como "realidade", conforme ilustra a Figura 4.6.

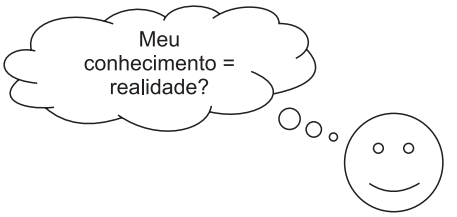


Figura 4.6 - Conhecimento X Realidade.

O conhecimento que não está em conformidade com a realidade é tido como de uso limitado e pode levar a graves erros.

Grécia Antiga: Epistema = Verdadeiro Conhecimento

Segundo os gregos, só podia ser adquirido por meio da observação de fenômenos.

4.10.5 A contribuição de Kant para a importância dos paradigmas

No seu livro "A crítica da razão pura" (1787), o filósofo alemão Emanuel Kant investigou o que a mente conhece ou pode conhecer, sem qualquer auxílio dos cinco sentidos. A preocupação de Kant era: existe algum conhecimento na mente humana que não se origina no mundo externo? Kant citou o raciocínio matemático como um primeiro exemplo. Ele demonstrou que a lógica matemática existe sem qualquer referência a sensações externas.

Paradigmas dominantes: são os modelos mais aceitos e utilizados como referência. Os paradigmas dominantes que aprendemos na nossa cultura estão prejulgando e predeterminando toda a nossa percepção do que está acontecendo. Por isto o estudo dos paradigmas é importante: para que não sermos prisioneiros dos paradigmas culturais e, portanto, cegos a maneiras alternativas de ver as coisas; precisamos trazer à tona para revisão as chamadas **suposições não examinadas**.

4.10.6 Aristóteles e as "leis da natureza"

As ideias universais iniciais que caracterizam as visões ocidentais do mundo natural foram criadas por Aristóteles há mais de 2.000 anos. As ideias de Aristóteles foram estudadas e aprimoradas por filósofos da Idade Média e pregavam um universo finito, mecanicista e propositadamente ordenado, no qual tudo tem o seu lugar próprio. Qual era a ideia básica? Conhecer algo é compreender sua essência, e conhecer sua essência depende da compreensão de sua causa necessária, isto é, que evento ou força causou ser esse objeto o que ele é. Segundo essa visão, o Universo era regido por leis e ordenado mecanicistamente. Achavam que nada acontecia por acaso.

4.10.7 A escola Milesiana na Jônia

Na cultura grega, ciência, filosofia e religião não eram separadas. Os sábios da escola Milesiana na Jônia não se preocupavam com tais distinções, porque o objetivo era descobrir a natureza essencial, ou a constituição real das coisas (chamada "*physis*"). O termo "*physis*" tem origem grega e significava, inicialmente, o empenho de ver a natureza essencial das coisas.

4.10.8 O grande desentendimento filosófico

Há 2.500 anos ocorreu um desentendimento entre dois filósofos gregos, que teve grandes implicações para o desenvolvimento posterior da ciência ocidental. Heráclito de Éfeso não acreditava no chamado "ser estático", ou seja, o ser humano é como é, independente dos outros elementos que o cercam. Heróclito ensinava que as constantes mudanças no mundo eram responsáveis pelas características dos seres e da natureza. Por outro lado, Parmênides de Eléa acreditava que as mudanças que nós percebíamos no mundo eram mera ilusão. Ele acreditava que o verdadeiro conhecimento dependia da compreensão inicial da essência imutável dos objetos no universo físico. Assim, o destino da ciência e da civilização ocidental ficou determinado quando a filosofia grega acabou preferindo a visão de mundo de Parmênides à de Heráclito.

René Descartes

Nos séculos XII e XIII, importantes avanços começaram a ser feitos em astronomia e em outras ciências físicas engajadas em exames mais detalhados do mundo físico propriamente dito. O trabalho do famoso matemático e filósofo René Descartes parece marcar o início de uma importante mudança no pensamento da comunidade científica da Europa no que diz respeito à relação entre homens, objetos e conhecimento. Descartes queria criar um sistema de linguagem científico universal baseado na matemática, o que permitiria ao homem investigar e, ao mesmo tempo, comunicar um conhecimento crescente sobre esse cosmos ordenado.

Augusto Comte

Filósofo francês, estimulado pelo sucesso das ciências físicas na metade do século XVIII, ensinava que desde o início da civilização ocidental, cada novo sistema de conhecimento passava por três estágios de progresso:

Estágio místico ou religioso: o homem compreendia os fenômenos observados com a ajuda de conceitos místicos ou religiosos.

Estágio metafísico: alguns aspectos que antes eram puramente místicos começam a ser compreendidos junto com fenômenos físicos observados.

Estágio da ciência positivista: não existe necessidade de invocar fatores místicos para explicar fenômenos físicos.

4.10.9 Surge a metodologia científica

A metodologia científica teve sua origem no pensamento de Descartes, posteriormente foi aperfeiçoada por Comte e finalmente desenvolvida empiricamente (ou seja, na prática) pelo físico inglês Isaac Newton. Surgiu então uma metodologia científica nova, dividindo o problema em partes, focalizando um objeto de estudo e utilizando o método do laboratório. A partir daí as evidências e as provas eram deduzidas das observações dos resultados (fenômenos) observados nos laboratórios.

4.10.10 A psicologia Lockeana

John Locke foi um filósofo inglês que, após estudos, afirmava que todas as pessoas começam por não saber absolutamente nada e que aprendem pela experiência, por tentativa e erro. Essa visão do funcionamento da mente proposta por Locke parecia confirmar as visões emergentes de **verdade**, **evidência**, **significado** etc. A preocupação contemporânea com fatos "concretos", dados empíricos, "prova positiva" etc. tem suas raízes no paradigma da ciência positivista e na explicação de Locke sobre o funcionamento da mente. Segundo Locke, nossa mente é um gravador passivo de impressões sensórias originadas em um objeto ou no mundo físico e transmitidas à mente através de nossos sentidos.

4.10.11 A necessidade de paradigmas alternativos

É fácil observar que as pessoas são relutantes em abandonar uma forma preferida de ver as coisas, especialmente quando esse ponto de vista é bem-sucedido. Como vimos anteriormente, o que se defende atualmente é a competência paradigmática, na qual o administrador aprende como mudar sua perspectiva conceitual e passar a ver o problema de várias perspectivas, sendo capaz de inventar soluções criativas.

4.10.12 Paradigmas de desenvolvimento de software

Apesar de nos depararmos com diversos paradigmas consagrados, existe uma necessidade de darmos abertura a paradigmas alternativos. Como exemplo, podemos citar um gerente que tenta encaixar todo e qualquer problema em um único modelo conceitual. A princípio ele pode ser bem-sucedido, considerando uma ordem de problemas ligados à estrutura com a qual está acostumado ou na qual está inserido. Contudo, este gerente poderá falhar se for obrigado a enfrentar problemas de um tipo diferente.

Atualmente é importante entendermos as diferenças entre as linguagens orientadas a objetos e as demais. Mas antes convém conhecer um pouco do histórico das linguagens de programação. As que fazem parte da primeira geração eram linguagens desenvolvidas por codificações binárias decimais, diretamente sobre o hardware, conhecidas como linguagens de máquina. As codificações binárias decimais, também conhecidas por BDC (*Binary-Coded Decimal*), são representadas por conjuntos de 4 bits, que permitem a indicação de números decimais de 0 a 9, conforme tabela específica. Esse padrão de codificação normalmente é utilizado em muitos circuitos integrados. Eram escritas de acordo com as características específicas de cada processador e executadas diretamente pelo hardware, sem traduções.

As linguagens de programação de segunda geração usavam abreviações simples para as instruções do programa e ficaram conhecidas popularmente como linguagens de baixo nível. Um exemplo clássico é a linguagem de montagem Assembly, cujo código-fonte era traduzido para linguagem de máquina por meio de um montador. A partir daí, a evolução do hardware e a necessidade de desenvolver novas técnicas de programação estimularam o aparecimento das linguagens de programação pertencentes à terceira geração, que ficaram conhecidas como linguagens de alto nível. Estas últimas são também conhecidas como estruturadas ou procedurais, que caracterizam grande parte das linguagens utilizadas atualmente, como, por exemplo, C, C++, C#, ASP, Java, Python etc.

A análise estruturada de sistemas foi concebida entre as décadas de 1960 e 1970, época em que era utilizada em grandes projetos, com ambientes e computadores de grande porte. Nesse período ainda era possível observar pouca maturidade da programação por parte dos técnicos que a utilizavam. A análise estruturada surgiu na tentativa de organizar os projetos de software ao impor alguma ordem ao desenvolvimento dos sistemas, produzindo, desta forma, um modelo rigorosamente sequencial. Nessa época a literatura tratava a análise, o projeto e a implementação de sistemas como atividades diferentes, sendo até, em certo ponto, desconexas. Isso se devia ao fato de cada uma delas possuir métodos, notações e objetivos diferentes. Exemplos de linguagens utilizadas na época são LISP, FORTRAN e COBOL.

Da análise estruturada passamos agora a estudar as características da programação estruturada, também conhecida como programação procedimental. O que caracteriza esse modelo é que o programa principal separa os dados dos seus procedimentos. Foi durante o uso mais intensivo das linguagens de terceira geração, na década de 1970, que surgiu o conceito de programação modular. Essa ideia foi uma consequência natural, pois baseia-se justamente no fato de a compilação ser feita por módulos separados. Desta forma, podemos dizer que os princípios da programação modular giravam em torno do ocultamento de informações pelo conceito de encapsulamento. Também existia uma ênfase ao reaproveitamento de código, que ficou conhecido como reusabilidade. Um exemplo de linguagem utilizada nessa época é a Modula-2.

Apesar de terem sido consideradas uma evolução significativa, as linguagens de programação de terceira geração (modulares) ainda não apresentavam soluções para diversos problemas; por exemplo, não possuíam suporte adequado para abstrações de dados. Essa deficiência fazia os erros serem detectados somente durante a fase de execução do programa. A solução para esse problema só ocorreu após o advento da programação orientada a objetos, a qual integrou as chamadas linguagens de quarta geração. Ela proporcionou mecanismos de acoplamento dos dados e dos procedimentos com esses dados, usando conceitos como encapsulamento, polimorfismo e herança. Outra característica interessante é que, ao contrário das linguagens estruturadas e modulares, que executavam suas instruções de maneira sequencial, as linguagens de quarta geração passaram a ter um processamento feito pela comunicação entre os objetos, conhecida como mensagem.

Ao longo da história da computação, observamos então que as quatro gerações de linguagens de programação mencionadas foram sendo categorizadas em três paradigmas básicos, sendo o paradigma funcional, o paradigma orientado a dados e, posteriormente, o paradigma orientado a objetos.

O paradigma funcional usa algoritmos e procedimentos para descrever o problema, por meio de diferentes notações ao longo do ciclo de vida de desenvolvimento. Para isso aplica métodos de representação de requisitos que são orientados a fluxo de dados. É um modelo claramente influenciado pelos sistemas computacionais e linguagens de programação convencionais existentes principalmente nas décadas de 1960 e 1970, os quais tinham como principal característica ações sequenciais.

As linguagens de programação que usavam esse paradigma baseavam-se na programação procedural, em que não havia o que chamamos hoje de encapsulamento, uma vez que os dados eram abertos e de uso global. As entidades criadas são dissociadas entre si, havendo uma distinção clara entre procedimentos e dados. Nessa época, a metodologia mais utilizada era a estruturada, em que a decomposição funcional do domínio do problema era mapeada para uma hierarquia de funções e subfunções. Observe que essa decomposição funcional baseava-se em dois componentes princi-

pais, sendo uma representação hierárquica, que mostra como uma função pode se dividir em várias funções, e uma representação de entrada-processo-saída, que mostra cada função na hierarquia em termos de suas entradas e saídas.

Portanto, os resultados apresentados pelo desenvolvedor ao usar a decomposição funcional estão relacionados aos níveis de sistema, subsistema, função e subfunção. Estudando este enfoque, observamos problemas que dizem respeito à instabilidade da funcionalidade do sistema e também à dificuldade de se obter alta coesão na descrição da composição dos componentes do sistema e das interfaces entre esses componentes. Uma dificuldade comum entre muitos desenvolvedores gira em torno da identificação de funções que seriam capazes de suportar os novos requisitos do sistema com mínimas alterações na análise e organização da especificação.

Já o paradigma orientado a dados utiliza elementos da informação e os seus relacionamentos, além de métodos de modelagem da informação específicos para estabelecer como a informação será apresentada. Para isso utiliza tecnologia de estruturação da informação, como, por exemplo, o modelo relacional, que é usado para determinar como a informação será armazenada. A estratégia de programação modular é frequentemente usada pelo paradigma orientado a dados, em que a organização do programa é feita pela criação de módulos, e a representação dos dados é feita por procedimentos. O que mais caracteriza o paradigma orientado a objetos é o fato de trabalhar com o conceito de encapsulamento de dados. Essa possibilidade apresentou um importante diferencial, pois com o encapsulamento podemos proporcionar a tão almejada reutilização de código, por meio da componentização. Essa estrutura oferece maior precisão para capturar regras do negócio quando comparada, por exemplo, a modelagens centradas nos dados. Assim, com a orientação a objetos podemos analisar dois aspectos importantes que sempre devem ser considerados no desenvolvimento de um software, sendo a informação e comportamento.

O paradigma orientado a objetos surgiu na tentativa de solucionar problemas existentes no desenvolvimento de softwares, que são complexos, observando a necessidade de manter um baixo custo de desenvolvimento e manutenção. Como vimos, o chamado "mundo real" é formado por objetos que interagem. Desta forma, representar esses objetos em um software seria muito mais natural e permanente do que representar a sua funcionalidade (decomposição funcional).

É importante reconhecer nesse momento o que separa e caracteriza o "mundo real" do "mundo computacional". O "mundo real" é o espaço de problemas, enquanto o "mundo computacional" é o espaço para soluções. Desta forma, torna-se necessário realizar um mapeamento (processo de identificação de abstrações) entre os objetos e operações que podemos observar no "mundo real" para objetos e operações abstratas e dados de saída, que correspondem à interpretação humana dos resultados necessários observados.

Mais recentemente, as aplicações voltadas para a Internet precisaram de novas arquiteturas para se tornarem eficientes. Um dos exemplos que tem sido amplamente utilizado é o SOA (*Service-Oriented Architecture*) ou arquitetura orientada a serviços, que é um tipo de arquitetura de software cujo princípio fundamental prega que as funcionalidades implementadas pelas aplicações devem ser disponibilizadas na forma de serviços. Esses serviços seriam conectados através de um barramento de serviços, também conhecido como *Enterprise Service Bus*. O objetivo desse barramento é disponibilizar interfaces (ou contratos) acessíveis através de web services ou outra forma de comunicação entre aplicações. Portanto, a arquitetura SOA baseia-se nos princípios da chamada computação distribuída.

4.11 Exercícios

1) Quais os principais problemas que podemos encontrar no uso do modelo sequencial linear?

2) Desenvolvedores e clientes gostam de usar o paradigma da prototipagem? Existem problemas relacionados ao uso desse paradigma?

3) Apesar de o modelo clássico (em cascata) ter sido um dos paradigmas mais amplamente utilizados na engenharia de software, sua aplicabilidade tem sido questionada. Por quê?

4) O desenvolvimento incremental sofreu influência ou utiliza elementos de outros paradigmas de desenvolvimento? Qual é a ideia básica desse modelo?

5) A abordagem evolucionária do desenvolvimento incremental às vezes não convence os clientes de que pode ser controlável, principalmente em situações envolvendo contratos. Qual é a estratégia usada para minimizar essa impressão?

6) Quais são os pontos fortes do IBM Rational Unified Process?

7) Comente as características e os propósitos dos artefatos do IBM® Rational Unified Process.

8) Qual é o propósito do PMBOK e como ele é organizado?

9) Como a engenharia de software é dividida no SWEBOK? Resuma cada uma das áreas.

10) O que a abordagem RAD tem a ver com os processos de desenvolvimento incremental e linear?

11) Qual a definição para o termo "competência paradigmática"? Comente este tema.

Anotações

5

Introdução às Metodologias Ágeis

Objetivos

- *Apresentar um histórico das metodologias ágeis.*
- *Apresentar e comentar o conteúdo do manifesto ágil.*
- *Expor as principais motivações que podem levar ao uso de metodologias ágeis.*

5.1 Histórico

Os conceitos existentes hoje relacionados ao desenvolvimento ágil de software nasceram em meados de 1990, motivados por uma reação adversa aos chamados "métodos pesados" de desenvolvimento de software, caracterizados por um formalismo muito grande nas documentações e regulamentações, sendo, na sua maioria, microgerenciados pelo tradicional e burocrático modelo em cascata. A partir desta reflexão, novos frameworks para processos de desenvolvimento de software começaram a surgir, sendo conhecidos inicialmente pela denominação "métodos leves" ou *Lightweight Methods*.

Em 2001, uma importante reunião ocorrida em uma estação de esqui, nas montanhas nevadas do estado norte-americano de Utah, marcou definitivamente o surgimento e a propagação de paradigmas de desenvolvimento de software ágeis. Estavam presentes nessa reunião dezessete profissionais de software que já trabalhavam com os chamados métodos leves utilizados na época, a saber: Kent Beck, Mike Beedle, Arie Van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland e Dave Thomas. Estas pessoas representavam metodologias já utilizadas, como SCRUM, *Extreme Programming*, DSDM, *Adaptative Software Development*, *Crystal*, *Feature Driven Development*, entre outras.

O objetivo do encontro era trocar ideias sobre o que estavam fazendo e discutir formas de melhorar o desempenho de seus projetos. Cada participante trouxe suas experiências, teorias e práticas de como obter sucesso no desenvolvimento de projetos de software, pois trabalhavam em organizações diferentes, com necessidades diferentes, portanto a aplicação de uma metodologia de desenvolvimento acabava sendo diferente das outras empresas.

Durante a reunião, eles perceberam que apesar de cada um utilizar uma prática diferente ou adaptada à sua necessidade, bem como haver teorias bem pessoais sobre como um projeto de software deva ser conduzido, todos acabaram concordando que existia um conjunto de princípios básicos que parecia ter sido respeitado por todos quando os projetos eram bem-sucedidos.

A partir daí, decidiram redigir um documento cujo nome foi sugerido pelo único inglês da turma, Martin Fowler, que utilizou o termo "*agile*" para descrever o que estavam discutindo. Desta forma, o documento foi chamado de "manifesto ágil", assinado por todos os presentes.

5.2 Manifesto ágil

O manifesto ágil foi criado considerando a velocidade demandada para novos sistemas de software, com o objetivo de abandonar métodos antigos que se mostravam ultrapassados devido ao uso de hardwares mais avançados, linguagens de programação, ambientes de desenvolvimento e necessidades organizacionais. Trata-se de um documento que encoraja o uso de melhores métodos de desenvolver sistemas de software, contendo um conjunto de princípios que definem critérios para os processos de desenvolvimento ágil de sistemas. O manifesto ágil é composto de doze princípios sob os quais qualquer metodologia ágil deve se ajustar, sendo descritos e comentados a seguir:

a) "A prioridade é satisfazer ao cliente por meio de entregas contínuas e frequentes de software de valor."

b) Esta visão evidencia que o cliente é o principal foco das equipes de desenvolvimento ágil, devendo entregá-lo de forma rápida e eficiente.

c) "Mudanças de requisitos são bem-vindas, mesmo em uma fase avançada do projeto. Processos ágeis esperam que a mudança traga uma vantagem competitiva ao cliente."

d) Processos ágeis não se incomodam caso projetos mudem ao longo de seu desenvolvimento, contudo as mudanças devem ser aceitas somente com o intuito de agregar valor e vantagens competitivas ao negócio do cliente. Mudanças que não colaboram com a evolução do projeto devem ser descartadas.

e) "Entregas com frequência de software funcional, sempre na menor escala de tempo, de algumas semanas a alguns meses, preferindo sempre um período curto."

Tempo de desenvolvimento é algo que deve se flexibilizar considerando as necessidades do cliente e da equipe de desenvolvimento. Sempre se deve buscar entregar o software funcional, com qualidade, o mais breve possível.

f) "As equipes de negócio e de desenvolvimento devem trabalhar juntas diariamente durante o projeto."

Deve existir uma comunicação frequente entre a equipe de desenvolvimento e os principais interessados no projeto (*stakeholders*). O cliente deve sempre estar presente durante o acompanhamento do projeto.

g) "Construa projetos objetivando manter uma equipe motivada, fornecendo ambiente, apoio e confiança necessários para realizar o trabalho."

A equipe de desenvolvimento, os líderes e gerentes de projetos devem transmitir confiança um ao outro. Equipes ágeis devem ter características autogerenciáveis em um ambiente de trabalho organizado e estimulante, prezando pelo crescimento de seus integrantes e pela manutenção da qualidade do trabalho.

h) "A maneira mais eficiente de a informação circular entre a equipe de desenvolvimento é por uma conversa cara a cara."

O cliente sempre deve estar presente ao se discutirem aspectos relacionados ao sistema sendo desenvolvido. Metodologias ágeis estimulam a realização de reuniões mais frequentes e rápidas, que tornam mais fácil o entendimento das necessidades do cliente e do negócio, evitando inadequações.

i) "Ter um software funcionando é a medida primária de progresso."

Esta preocupação remete ao entendimento de que entregar um software que não funcione é o mesmo que não entregá-lo. Espera-se que na medida em que as funcionalidades sejam validadas, sejam implementadas e entregues ao cliente.

j) "Processos ágeis promovem o desenvolvimento sustentável. Patrocinadores, desenvolvedores e usuários devem ser capazes de manter um ritmo constante."

O ritmo de entregas de novas funcionalidades consegue ser mantido, incluindo melhorias contínuas, se um ambiente organizado for proporcionado, estimulando a comunicação e com foco nas prioridades do projeto.

k) "Atenção contínua à excelência técnica e a um bom projeto aumenta a agilidade."

Seguir padrões de projeto que prezam pela qualidade e design é importante para manter a excelência técnica da equipe de desenvolvimento. Seguindo esses princípios, é possível entregar software com qualidade rapidamente.

l) "Simplicidade é essencial."

Estimula-se a busca por soluções simples para os requisitos do cliente, que possam eventualmente ser aprimoradas posteriormente. Arquiteturas complexas podem não agregar valor ao cliente e contribuir com atraso nas entregas.

m) "As melhores arquiteturas, requisitos e projetos provêm de equipes organizadas."

As metodologias ágeis estimulam a formação de equipes auto-organizáveis, pois conseguem se adaptar mais facilmente às mudanças de direção do negócio proposto pelos patrocinadores do projeto. Equipes auto-organizáveis reinventam e reestruturam o negócio com criatividade e conforme a demanda dos clientes.

n) "Em intervalos regulares, a equipe deve refletir sobre como se tornar mais eficaz, então sintoniza e ajusta seu comportamento."

Reflexões periódicas sobre o desempenho da equipe são importantes para identificar processos falhos e desnecessários, bem como traçar planos de melhoria futura, que devem ser propostos pela própria equipe.

Com base nos princípios elencados, cada um dos métodos ágeis existentes oferece um conjunto de atividades que podem ser adotadas durante um processo de desenvolvimento de software. A ideia principal é redefinir continuamente as prioridades em um projeto de software. Apesar de evidenciar certo grau de informalidade, é possível perceber que nas metodologias ágeis não deixaram de optar pelo uso de processos, ferramentas, negociações de contrato, documentos, porém de forma mais reduzida e objetiva.

Com base no manifesto ágil, que representa uma publicação que agia como ponto de partida para aqueles que compartilhavam as mesmas ideias básicas, foi criada a "Aliança Ágil", um dos frutos desse esforço, que representa uma entidade mais permanente. Trata-se de uma organização sem fins lucrativos que procura promover o conhecimento e discussões sobre todas as metodologias ágeis; muitos dos líderes do movimento ágil mencionados anteriormente também são membros dessa aliança. Informações complementares sobre a Aliança Ágil podem ser obtidas em http://www.agilealliance.org/.

5.3 Principais motivações

O desenvolvimento de software sempre foi considerado uma atividade complexa e, por que não dizer, caótica, principalmente se o software é desenvolvido sem um plano definido aliado a decisões de curto prazo que são tomadas ao longo do seu desenvolvimento. A ausência de um planejamento formal pode até dar certo quando o software a ser desenvolvido é de baixa complexidade. Mas mesmo neste caso, se considerarmos que novas funcionalidades podem ser agregadas, acaba se tornando mais difícil gerenciar a inclusão desses novos requisitos sem um planejamento adequado. Neste cenário, normalmente percebemos que comportamentos indesejados se tornam frequentes e cada vez mais difíceis de serem eliminados. Podemos constatar facilmente que um determinado sistema de software passou por essa situação quando observamos que ele experimentou uma longa fase de testes, que fatalmente contribuiu para atrasar o cronograma, associado ao fato de testes serem atividades cujo tempo de execução é difícil de ser estimado.

Portanto, não é exagero afirmar que o uso de metodologias, independente de quais sejam, é um consenso hoje em dia, pois impõe o uso de processos disciplinados durante o desenvolvimento de um software. A principal motivação no uso de processos disciplinados é que eles têm o objetivo de tornar o desenvolvimento mais previsível e eficiente. Conforme abordado anteriormente, as metodologias ágeis surgiram em resposta a metodologias tradicionais mais "engessadas", proporcionando mudanças de ênfase significativas e estimulando pequenos grupos de desenvolvedores a adotar um processo disciplinado para projetos de qualquer natureza.

Vamos utilizar como argumentos, para ilustrar as principais motivações no uso de metodologias ágeis, pontos de vista encontrados em artigos publicados a partir de junho de 2000 por Martin Fowler, um dos signatários do manifesto ágil.

Segundo Fowler, talvez uma das grandes motivações, que tem atraído cada vez mais profissionais da área de tecnologia da informação e também de outras áreas, deve-se ao fato de as metodologias ágeis serem menos centradas em documentações, sendo mais voltadas ao código-fonte do sistema de software. Contudo, não é conveniente associarmos apenas essa característica quando pensamos em metodologias ágeis, pois existem outras diferenças tão impactantes quanto esta. Uma delas é o fato de que as **metodologias ágeis são adaptativas em vez de predeterminantes**. Isso se deve ao fato de que as metodologias propostas pela engenharia de software tradicional tendem a planejar grande parte do processo de desenvolvimento por um longo período de tempo, prática que funciona bem até que haja a necessidade de mudar o projeto. Neste caso, a tendência é resistir a essas mudanças. Já as metodologias ágeis aceitam mudanças ao longo do desenvolvimento de maneira natural, adaptando seus processos as mudanças exigidas.

Outro aspecto muito importante que diferencia as metodologias ágeis das tradicionais é que os **métodos ágeis utilizados são orientados a pessoas e não a processos**. Na engenharia de software tradicional, o objetivo dos métodos adotados é definir processos que possam funcionar bem independentemente de quem os estiver usando. Os métodos ágeis pregam que nenhum processo pode ser equivalente à habilidade dos integrantes da equipe de desenvolvimento. Assim, os processos existentes nos métodos ágeis têm o papel de dar suporte à equipe de desenvolvimento. A seguir apresentamos diversos aspectos que envolvem conceitos relacionados às metodologias ágeis, utilizados como subsídio para argumentar motivações que levam ao seu uso.

5.3.1 Por que optar por uma metodologia adaptativa?

Quem trabalha com desenvolvimento de software, ou quem pretende um dia fazer isso, certamente vai perceber que grande parte dos projetos possui requisitos que são alterados ao longo de seu ciclo de desenvolvimento. Atualmente não deveríamos mais nos espantar com essas mudanças, uma vez que até entendemos suas origens. Mas o que muitas equipes de desenvolvimento não costumam debater com frequência é que, se essas mudanças fazem parte do modelo de negócio atual, o que deve ser feito para que possamos adaptar nossos processos a essa realidade?

Tradicionalmente, a ideia por trás da chamada engenharia de requisitos é obter uma visão clara de todos os requisitos antes de iniciar o desenvolvimento do software. Adicionalmente a essa visão, tentamos obter o aceite do cliente sobre os requisitos levantados e, finalmente, definir procedimentos que limitem a alteração destes.

Esse "congelamento" de requisitos é desejável, pois com ele aumentamos as chances de cumprir prazos para entregar o projeto ou marcos de desenvolvimento, além de mantermos os custos previstos para os requisitos iniciais levantados, mas infelizmente requisitos imutáveis não fazem mais parte da realidade dos desenvolvedores de software. E isso se deve a vários fatores, entre eles a dificuldade de definir o valor de uma determinada funcionalidade do software que está sendo desenvolvido até que ele seja materializado.

O cliente vai começar a entender o valor de uma funcionalidade apenas quando tiver acesso a uma versão preliminar. Esta é uma das razões que tornam os requisitos mutáveis, fazendo com que, consequentemente, os desenvolvedores gastem muito tempo para "consertar" os requisitos dos clientes. Outro motivo que torna requisitos de software mutáveis é o cenário - macro e microeconômico - em que as empresas estão inseridas, sendo alvo constante de alterações de modelos de negócio e tendências. A economia atual pode fazer com que o valor de um requisito de software

mude rapidamente, e um bom requisito identificado hoje pode não ser mais importante ou prioritário daqui a alguns meses. Então, mesmo que façamos um grande esforço para fixar requisitos de determinado cliente, contando inclusive com seu apoio nesse sentido, sabemos que o mundo dos negócios não irá parar e aguardar que o software seja concluído. Portanto, temos de conviver com o fato de que, se não podemos obter requisitos estáveis, consequentemente não podemos utilizar um plano de projeto estável.

Não estamos querendo dizer com isso que é impossível alcançar um alto grau de previsibilidade em projetos de software. Certamente existem casos em que não ocorrem muitas alterações de requisitos ao longo do desenvolvimento do software. Mas não é o que observamos na maioria dos sistemas desenvolvidos hoje em dia. Para estes casos é necessário utilizar processos diferentes dos tradicionais, o que se torna um desafio, porque é difícil para a maioria que trabalha com metodologias de desenvolvimento identificar se a metodologia que vem sendo utilizada é ou não apropriada.

Normalmente esse questionamento nem é feito, pois se subentende que metodologias existem para serem utilizadas por todo e qualquer projeto. Essa premissa acaba levando ao uso de determinada metodologia de desenvolvimento de software em circunstâncias erradas, pois, como já mencionamos, não é adequado utilizar uma metodologia previsível em uma situação imprevisível. Este é o principal recado e fator motivador para optarmos por uma metodologia adaptativa: **se estamos em um contexto não previsível, não podemos usar uma metodologia previsível**. Portanto, muitos modelos utilizados para controlar projetos e o relacionamento que devemos manter com clientes não se aplicam mais ou precisam ser revistos. O uso de metodologias adaptativas, que pregam o desapego à previsibilidade, não significa que tenhamos de voltar ao caos incontrolável. Na verdade, elas proporcionam um controle da imprevisibilidade, sendo este o ponto mais relevante de uma metodologia adaptativa.

5.3.2 Design e construção de software usando uma metodologia adaptativa

As tradicionais áreas de engenharia, que inspiraram as metodologias de desenvolvimento de software, sempre enfatizaram que o planejamento deve ser feito antes da construção. Assim, muitas decisões relacionadas com o design (desenho) são tomadas, como, por exemplo, a forma como uma ponte deve ser construída para aguentar sua carga, à medida que os desenhos vão sendo desenvolvidos. Normalmente tais desenhos são entregues a outro grupo de profissionais para serem construídos, muitas vezes pertencentes à outra empresa, e a partir daí assume-se que o processo utilizado para a construção deverá seguir os desenhos.

Esse processo tem funcionado nas áreas de engenharia civil, mecânica, entre outras, que consideram os desenhos para especificar os componentes do projeto e a maneira pela qual se encaixam, agindo como uma verdadeira fundação para um posterior plano de construção detalhado. A partir desse plano é que são derivadas as atividades que precisam ser feitas, bem como as dependências existentes entre elas, permitindo, consequentemente, a definição de um cronograma, bem como o levantamento de um orçamento relativamente previsível para o projeto. Ainda considerando o desenho produzido, podemos detalhar como os integrantes do projeto deverão proceder com relação aos seus trabalhos, contribuindo também para conceber certo grau de qualidade nos processos adotados.

Considerando as duas atividades descritas, o design e a construção, tradicionalmente utilizados pelas áreas clássicas de engenharia, podemos notar a semelhança existente com abordagens relacionadas à engenharia de software, uma vez que neste caso também desejamos construir um cronograma previsível baseado no desenho apresentado, sendo também necessário separarmos as áreas de design da construção do software. O desafio está então relacionado à escolha de como fazer o design do software de forma que a construção não seja composta de tarefas complicadas.

É neste ponto que começam possíveis resistências ao uso de paradigmas ágeis ou, caso a resistência não exista, dificuldades para considerar seu uso prático. O que ocorre é que, para muitos, o design de um software deve ser feito por meio de notações de modelagem, tal como a UML (um estudo completo sobre a UML pode ser encontrado no livro UML 2.3 - Teoria e Prática, de José Henrique Teixeira de Carvalho Sbrocco, também editado pela Editora Érica). Isso se sustenta porque a UML servirá de base para atividades estritamente de construção, realizadas pelos programadores.

Martin Fowler questiona se, ao criarmos um design de software, seremos capazes de transformar a programação em uma atividade de construção previsível, a exemplo do que constatamos em outras áreas da engenharia, como, por exemplo, a engenharia civil. Ele ainda vai mais adiante, questionando também se o custo do design de software é suficientemente baixo para que esta seja uma abordagem vantajosa. Sabemos que a aplicação de diagramas da UML é útil em diversas situações, principalmente quando trabalhamos com soluções complexas. Por essa razão não vamos questionar sua pertinência ou não. O que estamos ressaltando é que adeptos das metodologias ágeis questionam a natureza do design em software comparado com seu papel em outros ramos da engenharia. Esses questionamentos levaram Jack Reeves a sugerir que o código-fonte é, por si só, um documento de design, e a fase de construção seria representada apenas pelo uso do compilador e do *linker*. Podemos concluir então que a metodologia ágil prega que devemos ter muito cuidado com a metáfora tradicional de engenharia para construção de software, pois esta última requer um tipo diferente de atividade, precisando, consequentemente, de um processo diferente.

5.3.3 Desenvolvimento iterativo X controle de processos imprevisíveis

Antes de adotarmos processos que podem ser adaptados, temos de exercitar a habilidade de perceber em que situação estamos periodicamente. Para isso podemos remeter ao conceito de desenvolvimento iterativo, que já vem sendo usado há muito tempo, sendo conhecido por várias denominações, tais como desenvolvimento evolucionário, incremental, em espiral, em estágios, entre outros.

Segundo Martin Fowler, a chave para o desenvolvimento iterativo é frequentemente produzir versões do sistema final que funcionem, sendo que cada versão deve ser totalmente testada e integrada como se fosse a entrega do produto final. O teste e a integração de partes do sistema ao longo de seu desenvolvimento concebem uma dose de realidade a qualquer projeto. Limitar-se a analisar documentos ou não testar códigos-fonte pode esconder falhas, ao contrário da percepção proporcionada quando se senta em frente a um monitor e trabalha com o software ou parte do sistema que foi implementada e testada. Neste caso, as falhas se tornam aparentes, tanto em termos de defeito como relacionadas a requisitos mal interpretados.

Observe que o desenvolvimento iterativo faz sentido em processos previsíveis, mas seu uso é particularmente importante em processos adaptativos, pois estes precisam ser capazes de lidar com mudanças nas funcionalidades requisitadas. Assim, percebemos que o uso de processos iterativos em um processo adaptativo proporciona um planejamento de longo prazo bastante fluído, sendo que os únicos planos estáveis são os de curto prazo (representados por uma iteração). Assim, o desenvolvimento iterativo dá uma fundação sólida para cada iteração, que posteriormente pode ser usada como *feedback* de um futuro replanejamento. Quanto mais curta a iteração seja, mais frequente será o *feedback* proporcionado.

5.3.4 Processos adaptativos X relacionamento com o cliente

Processos adaptativos requerem diferentes tipos de relacionamento com os clientes. Essa diversidade de posturas é necessária porque normalmente a maioria dos clientes prefere um contrato de custo fixo, ou seja, o cliente apresenta o problema, junta as propostas recebidas e, uma vez escolhida a empresa, passa automaticamente todo o ônus do desenvolvimento para ela. O problema é que contratos de custo fixo requerem requisitos estáveis. Como já vimos que esse cenário não tem feito mais parte de nossa realidade atual, consequentemente não podemos mais trabalhar com a habitual noção de custo fixo. Também não podemos tentar simplesmente encaixar um modelo de custo fixo em um processo adaptativo, pois tanto os clientes como os desenvolvedores se sentiriam desconfortáveis com os perigos existentes pelo fato de ambas as partes assinarem um contrato de custo fixo em um cenário onde um processo predeterminista não possa ser utilizado.

Não estamos querendo dizer que não seja possível fixar um orçamento para o desenvolvimento de software antes de desenvolvê-lo. O que não podemos fazer é fixar tempo, preço e escopo, pois abordagens ágeis costumam apenas fixar tempo e preço, deixando o escopo variar de uma maneira controlada. Desta forma, com o uso de um processo adaptativo podemos proporcionar ao cliente um controle mais refinado do processo de desenvolvimento, porque após o término de cada iteração é possível verificar tanto o progresso como também alterar o direcionamento do desenvolvimento. Essa estratégia aproxima muito os clientes dos desenvolvedores de software, estabelecendo uma verdadeira parceria de negócios. Com esse modelo percebe-se que os clientes conseguem mudar as funcionalidades do sistema à medida que seu negócio muda, considerando também a forma como o sistema é utilizado na prática.

Outra contribuição importante das abordagens ágeis é que conseguimos ter maior visibilidade do real estado de um projeto, ao contrário das metodologias tradicionais, em que a qualidade acaba sendo medida pela análise da conformidade com seu planejamento. Quando a realidade e o planejamento divergirem, certamente causará um atraso no cronograma previsto inicialmente.

Quando usamos paradigmas de desenvolvimento ágeis, podemos proporcionar um constante retrabalho no plano de projeto de cada iteração. É sempre melhor que alterações necessárias sejam percebidas o mais cedo possível, em um momento em que ainda há tempo para se fazer algo a respeito. Os métodos ágeis vão além dos controles de riscos proporcionados pelo desenvolvimento iterativo, pois mantêm a duração das iterações bem curta, vendo possíveis solicitações de alterações como oportunidades.

É interessante observarmos que, tradicionalmente, um projeto é considerado bem-sucedido quando está dentro do prazo e do custo. Contudo, essa medida não faz sentido algum em um ambiente ágil, pois para seus praticantes a questão principal é o valor do negócio, ou seja, um software

de valor é aquele que proporciona ao cliente mais do que o dinheiro que ele empregou em seu desenvolvimento. Martin Fowler dizia que um bom projeto predeterminista normalmente ocorre conforme seu planejamento. Já um bom projeto ágil irá construir algo diferente e melhor do que o plano original previu.

5.3.5 Metodologias ágeis: pessoas em primeiro lugar

Segundo Martin Fowler, executar um processo adaptativo não é fácil, exigindo uma equipe de desenvolvimento muito eficaz, composta de integrantes qualificados e também que interagem como um time de verdade. O conceito tradicional de processo implica o desenvolvimento de metodologias em que pessoas são partes substituíveis, ou seja, o conhecimento sobre o que deve ser feito se encontra nos processos.

Analisando esse modelo, percebemos que as pessoas não são menos importantes que as funções que elas devem desempenhar. Portanto, se um projeto é planejado, não importa quais analistas, programadores ou especialistas em testes irão compor a equipe, bastando apenas saber quantos deles serão necessários. Este é outro ponto significativamente importante relacionado ao uso de metodologias ágeis, pois as considerações anteriores levantam a seguinte questão: será que as pessoas envolvidas no desenvolvimento de um software são, de fato, peças substituíveis?

As metodologias ágeis pregam justamente o contrário, rejeitando essa premissa. Alistair Cockburn, em seu artigo "*Caracterizando pessoas em desenvolvimento de software como componentes de primeira ordem, não lineares*", indica que processos predeterministas exigem componentes previsíveis. Contudo, pessoas não são componentes previsíveis, sendo altamente variáveis e não lineares. Cockburn ainda cita que as pessoas são tratadas como "componentes" nas literaturas tradicionais que falam sobre processos ou metodologias. As metodologias tradicionais falham por não levar em conta fatores relacionados com a imprevisibilidade das pessoas durante a definição de processos e das metodologias utilizadas, o que pode proporcionar todo tipo de trajetórias não planejadas durante a condução de projetos.

Mesmo sendo Cockburn um dos mais explícitos em sua visão sobre o desenvolvimento do software orientado a pessoas, a noção sobre o fato de as pessoas estarem em primeiro lugar é um tema amplamente discutido na engenharia de software. As controvérsias acabam surgindo, pois, na maioria das vezes, as metodologias se opõem à noção de que as pessoas são fatores de primeira ordem no sucesso de um projeto.

A questão principal é que, se esperamos que nossos desenvolvedores sejam unidades de programação substituíveis, acabamos não tentando tratá-los como indivíduos, diminuindo a moral, a motivação e, consequentemente, a produtividade. Este cenário certamente contribui para fazer com que os bons profissionais procurem um lugar melhor para trabalhar, fazendo com que o processo atinja seu propósito inicial: proporcionar unidades de programação substituíveis.

Não é fácil decidir que as pessoas devem vir em primeiro lugar, requerendo muita determinação para que essa decisão seja levada adiante, pois a noção de que as pessoas são recursos está muito arraigada no pensamento de negócios. Isso se deve, em grande parte, aos conceitos herdados de Taylor; ele sugere que as pessoas que estão fazendo o trabalho não são as melhores pessoas para descobrir a melhor forma de fazer esse trabalho. Se observarmos uma fábrica, esse raciocínio pode ser verdade por alguns motivos. Em parte porque a maioria dos trabalhadores fabris não tem um

perfil criativo, em parte porque facilmente constatamos a existência de uma tensão entre a administração e os trabalhadores, motivada pelo fato de que a administração é mais bem remunerada.

Mas no caso do desenvolvimento de software, a história tem mostrado cada vez mais o quanto isso é falso, uma vez que políticas e ações adotadas pelas empresas de desenvolvimento de software têm alinhado os interesses dos programadores com os da empresa. Atualmente, o conceito básico adotado quando desejamos contratar ou reter bons profissionais é que, em primeiro lugar, você deve reconhecer que eles são profissionais competentes. Considerando essa característica, torna-se natural imaginar que elas seriam então as melhores pessoas para decidir como conduzir seus trabalhos técnicos. Portanto, a antiga noção Taylorista que prega o uso de um departamento de planejamento que decide como as coisas são feitas só funciona se quem está planejando entender melhor o problema do que quem está executando. Assim, se você tem pessoas capazes e motivadas executando o trabalho, essa estrutura não se sustenta.

5.3.6 Gestão orientada a pessoas

Nos processos ágeis podemos evidenciar a orientação a pessoas de várias formas, levando a diversos efeitos, nem todos consistentes. A principal preocupação, que está relacionada a um dos elementos-chave, é a aceitação do processo em vez da imposição do processo. Sabe-se que grande parte da resistência percebida ao uso de processos de software se deve ao fato de serem impostos pela alta administração. Uma das razões é que a administração está muito distante das atividades de desenvolvimento de software.

A grande diferença entre aceitar um processo e impor um processo é que aceitar um processo requer comprometimento e envolvimento de toda a equipe, o que não observamos em processos impostos. Refletindo sobre essa realidade, podemos concluir que apenas os próprios desenvolvedores poderiam optar por seguir um processo adaptativo, principalmente porque eles é que devem ser capazes de tomar todas as decisões técnicas, como, por exemplo, realizar estimativas de tempo sobre uma determinada tarefa que deve ser executada.

A liderança técnica delegada à equipe de desenvolvimento causa, consequentemente, uma grande mudança em pessoas com posições gerenciais, pois tal abordagem exige um compartilhamento de responsabilidades, em que desenvolvedores e gerentes acabam tendo uma divisão igual na liderança do projeto. É importante perceber que a gerência ainda desempenha um papel, mas, neste caso, reconhece o conhecimento dos desenvolvedores. O contato com esse paradigma deveria ser algo natural, uma vez que até mesmo técnicos entendem que suas habilidades técnicas se tornarão obsoletas ao entrarem para uma função gerencial. Precisam, portanto, reconhecer que suas habilidades técnicas desaparecerão rapidamente, precisando necessariamente confiar nos desenvolvedores mais atualizados.

Em um cenário no qual processos determinam que pessoas devam dizer como um trabalho deve ser feito, não sendo essas pessoas as mesmas que executam o trabalho de fato, gerentes buscam alguma forma de medir a efetividade (produtividade) dos seus colaboradores. Robert Austin discutiu questões relacionadas à dificuldade de medições em projetos de software, concluindo que inevitavelmente devemos escolher entre gerenciamento por medidas (que muitas vezes não leva em conta fatores importantes) e gerenciamento delegativo (em que a decisão sobre como o trabalho deve ser feito fica a cargo dos trabalhadores).

Gerenciamento por medidas tem sido mais adequado para trabalhos simples e repetitivos, com poucos requisitos e de produção facilmente mensurável, cenário exatamente oposto ao desenvolvimento de um software. Métodos de desenvolvimento de software tradicionais operam sob a premissa de que o gerenciamento por medidas é a maneira mais eficiente de gerenciar. A comunidade ágil reconhece que gerenciamento por medidas leva a altos índices de disfunção nessas medidas, sendo mais eficiente utilizar um estilo delegativo de gerenciamento.

Outro aspecto importante relacionado à gestão orientada a pessoas diz respeito ao papel da liderança de negócio. Sabemos que o pessoal técnico não pode fazer todo o processo sozinho, necessitando de orientações advindas das necessidades de negócio. Portanto, processos adaptativos precisam oferecer mecanismos que levam a um contato próximo do conhecimento do negócio, que vai além do envolvimento do negócio normalmente utilizado na maioria dos projetos.

Um aspecto fundamental da gestão orientada a pessoas é que equipes ágeis não podem ser concebidas com uma comunicação ocasional, necessitando de acesso contínuo ao conhecimento do negócio. É importante ressaltar que esse acesso contínuo não deve ser algo administrado em âmbito gerencial, mas algo necessariamente presente para cada desenvolvedor. Isso se justifica em função do fato constatado anteriormente de que o desenvolvimento adaptativo tem como premissa básica que as coisas mudam rapidamente, por isso precisamos estabelecer um contato contínuo para que tais mudanças sejam visíveis a todos. Esta é outra característica que contribui com aspectos motivacionais da equipe, pois não há nada mais frustrante para um desenvolvedor como ver seu árduo trabalho sendo desperdiçado.

5.3.7 Adaptabilidade de processos

Há outro ângulo sobre a adaptabilidade de processos que merece ser citado, relacionado ao fato de o próprio processo mudar ao longo do tempo. Portanto, é provável que um projeto que se inicia utilizando um processo adaptativo acabe não usando o mesmo processo um ano depois. Isso ocorre porque, com o tempo, a própria equipe irá encontrar atividades que funcionam para ela, fazendo com que o processo se altere para ajustar-se a novas necessidades.

A primeira regra a ser seguida quando buscamos uma autoadaptabilidade envolve revisões regulares do processo. Prega-se que, sempre ao final de cada iteração, uma rápida reunião ocorra, objetivando responder às seguintes perguntas, propostas por Norm Kerth:

- O que fizemos direito?
- O que aprendemos?
- O que podemos melhorar?
- O que nos intriga?

As respostas a essas perguntas levarão a novas ideias que devem proporcionar mudanças no processo, que serão utilizadas em uma nova iteração. Portanto, um processo que foi iniciado com problemas pode ir melhorando à medida que as iterações ocorrem, adaptando-se continuamente às necessidades observadas pela equipe que o utiliza.

Se observarmos os benefícios da autoadaptabilidade dentro de um projeto, certamente serão mais marcantes dentro do contexto da organização como um todo. Considerando as consequências

da autoadaptabilidade, podemos presumir que não devemos esperar encontrar uma metodologia corporativa única, pois cada equipe de desenvolvimento deve escolher sua própria metodologia, e mesmo assim precisa ser ajustada continuamente.

Estamos, neste momento, falando de motivações genéricas que podem levar à adoção de metodologias ágeis. Contudo, como veremos mais à frente, questões relacionadas à autoadaptabilidade podem ser mais marcantes em determinadas metodologias ágeis, por exemplo, como ocorre na ASD e Crystal, e podem parecer proibitivas em metodologias com regras mais rígidas, como o XP, embora saibamos que pode parecer uma impressão superficial, pois o XP, de fato, também estimula ajustes no processo. A diferença é que, neste caso, a sugestão é seguir literalmente o processo por várias iterações, antes de adaptá-lo. Outra diferença é que na XP a revisão não faz parte do processo nem é encorajada. Posteriormente conheceremos melhor as diferenças entre as diversas metodologias ágeis disponíveis.

Ainda segundo Martin Fowler, usar uma metodologia ágil não é para todos, embora ele entenda que essas metodologias são amplamente aplicáveis e deveriam ser utilizadas por mais pessoas, pois têm a vantagem de ser um passo bem mais largo que usar uma metodologia "pesada". Um dos aspectos limitadores dessas novas metodologias é como usá-las com times grandes de desenvolvedores, considerando que enfatizam a organização de equipes pequenas. É importante lembrar que muitas equipes de desenvolvimento de software poderiam ser reduzidas em tamanho sem prejuízo da produtividade geral. Algumas abordagens ágeis, como a FDD, foram originalmente concebidas para projetos contendo equipes de cinquenta pessoas. O SCRUM também já foi utilizado em projetos com equipes de tamanho similar.

5.4 Exercícios

1) O que caracteriza os chamados "métodos pesados" de desenvolvimento de software?

2) Qual é o objetivo do manifesto ágil?

3) Por que as metodologias ágeis são consideradas adaptativas em vez de predeterminantes?

4) Qual é a principal razão que pode motivar o uso de metodologias adaptativas?

5) Segundo o exposto neste capítulo, qual seria a chave para um desenvolvimento iterativo de sucesso?

6) As pessoas normalmente são tratadas como "componentes" do projeto nas literaturas tradicionais que falam de processos e metodologias. Qual é o ponto de vista das metodologias ágeis sobre o papel que as pessoas desempenham?

7) Sob o ponto de vista de uma gestão orientada a pessoas, qual é a grande diferença entre aceitar um processo e impor um processo?

8) As metodologias ágeis podem ser utilizadas em projetos que envolvem grande número de pessoas? Comente.

6

Feature Driven Development (FDD)

Objetivos

- *Apresentar as origens e características básicas da metodologia FDD.*
- *Exemplificar como o FDD pode ser utilizado na prática.*

6.1 Origens e características básicas

A metodologia ágil *Feature Driven Development*, ou FDD, como é comumente chamada, foi criada em Singapura nos anos 1990, mais especificamente em meados de 1997/1998. Foi utilizada pela primeira vez para o desenvolvimento de um sistema bancário internacional, considerado inicialmente inviável de ser desenvolvido em um prazo predeterminado. Criada por Jeff De Luca e Peter Coad, a FDD é uma metodologia ágil robusta e muito utilizada nos dias atuais. O projeto FDD mantém um site em que a metodologia é discutida em forma de fórum, disponível em www.featuredrivendevelopment.org. Nesse endereço também podemos encontrar um link para certificações na metodologia.

6.1.1 Características básicas da metodologia

Traz benefícios a gerentes, desenvolvedores e clientes

A FDD proporciona uma forma de trabalho que agrada todos os envolvidos no projeto, sugerindo formas de interação e controle fáceis e inteligentes. Os desenvolvedores se sentem muito à vontade durante a implementação, pois a FDD possui um conjunto de regras de fácil entendimento e de resultados rápidos, trazendo também vantagens para o cliente.

Benefício ao cliente por meio de trabalho significativo

Conforme determina o manifesto ágil, essa metodologia também procura realizar seus trabalhos de maneira significativa desde o início do projeto de software, ou seja, no início da implementação visa sempre agregar valor ao cliente, promovendo a disponibilização daquilo que já foi desenvolvido e testado.

Atende equipes pequenas, médias ou grandes

A FDD fornece uma estrutura capaz de atender todos os tipos de equipes de trabalho, desde pequenas até grandes. Oferece um conjunto sólido de atribuições capazes de ser ampliadas, se necessário, mesmo em projetos descentralizados, ou até mesmo em projetos web.

Software de qualidade

A produção de software utilizando a FDD engloba um conjunto de métricas de qualidade a serem aplicadas durante o desenvolvimento do projeto de software. A FDD enfatiza sempre que o software deve ter qualidade desde o início de sua implementação.

Entrega de resultados frequentes, tangíveis e funcionais

Durante a fase de implementação, a FDD sugere que sempre que alguma funcionalidade for implementada e testada ela deve ser disponibilizada aos usuários.

Permite o acompanhamento do progresso do desenvolvimento do projeto

Conforme veremos em exemplos práticos apresentados neste capítulo, a FDD possui uma forma muito simples de visualização do andamento do desenvolvimento do software, por meio do uso de um método gráfico. Esse recurso possibilita à equipe e aos clientes rapidamente verificar como o projeto está evoluindo.

6.2 Práticas da FDD

A seguir apresentamos um conjunto de boas práticas a serem seguidas, indicadas pela *Feature Driven Development*:

6.2.1 Modelagem de objetos do domínio

Trata-se de uma atividade que deve ser realizada com toda a equipe com a finalidade de estudar, analisar e modelar um sistema. O modelo é feito de forma a descrever uma visão macro de todo o projeto, com o objetivo de demonstrar o que será produzido e guiar a equipe. A ideia é produzir um documento de requisitos a partir de um método de coleta de dados, que pode ser uma entrevista, um questionário ou um formulário de pesquisa, além de alguns diagramas da UML que ajudam o desenvolvedor no momento da implementação.

6.2.2 Desenvolvimento por funcionalidade (feature)

Essa prática sugere que, a partir dos requisitos, construa-se uma lista de funcionalidades decompondo funcionalmente o domínio do negócio. Categorizada em três níveis:

a) Áreas de Negócio (*Business Areas*, *Major Feature Sets*)

b) Atividades de Negócio (*Business Activities*, *Feature Sets*)

c) Passos da Atividade de Negócio (*Activity Steps*, *Features*)

6.2.3 Entregas regulares (builds)

Os desenvolvedores devem sempre reconstruir o software, ou seja, ativar as funcionalidades e modificações realizadas. Assim, os demais desenvolvedores utilizam e trabalham em uma versão sempre atualizada do projeto. Os clientes também se beneficiam utilizando sempre a última compilação do projeto. Obviamente é preciso testar as modificações e manter um controle de versões caso o impacto apresente problemas.

6.2.4 Formação da equipe de projeto

Gerente de projeto

Tem como função o contato com direto com o cliente (*stakeholders*) do projeto e captar todos os requisitos, bem como as possíveis restrições. Cabe ao gerente de projeto formar a equipe de desenvolvimento de acordo com as funcionalidades ou "*features*". O gerente deve compor sua equipe analisando as afinidades e experiências das pessoas que irão trabalhar primeiramente com a modelagem do projeto. Dependendo do tamanho do projeto, pode ser necessário mais que uma pessoa para modelagem. Cabe ao gerente também acompanhar todo o desenvolvimento, atentando-se sempre para as boas práticas da FDD, atuando como um gerente de desenvolvimento para a equipe. O gerente deve também escrever um estudo completo juntamente com especialistas sobre as regras de negócio (domínio da aplicação) do projeto para que os demais desenvolvedores e membros da equipe tomem ciência de como tudo funciona.

Arquiteto-chefe/especialista no domínio

A equipe deve possuir um profissional especialista no assunto do projeto a ser desenvolvido, para sanar dúvidas sobre as regras de negócio. Em alguns projetos mais simples pode não ser necessário, mas em assuntos específicos pode ser imprescindível sua presença ou acesso constante a esse especialista.

Equipe de modelagem/planejamento

Trata-se de uma pessoa ou grupo definido pelo gerente capaz de elaborar a lista de funcionalidades *"features"* do sistema em questão. Esse profissional deve dominar, entre outras habilidades, técnicas de modelagem de sistemas, com conhecimento de UML suficiente para trabalhar com diver-

sos diagramas. Deve ainda dividir as *features* e distribuí-las para a equipe de funcionalidades, "*Team Features*", por meio do programador chefe, determinando assim a sequência do desenvolvimento.

Programador-chefe

Esse profissional deve, entre outras atribuições, refinar a lista de *features* e transformá-la em modelo de objetos. Deve também organizar os trabalhos, escolhendo a linguagem e as formas de armazenamento, bem como as integrações "*build*", revisando e aprovando a liberação para o cliente.

Equipe de funcionalidades *"features"*

São os desenvolvedores propriamente ditos, profissionais que devem conhecer as ferramentas de desenvolvimento necessárias para o desenvolvimento do projeto, além de como o sistema funciona. Cabe a essa equipe escrever os métodos e as classes necessárias para a concepção do projeto, alem de codificar todas as funcionalidades envolvidas. Essa equipe também deve ser responsável por construir testes de unidade para validar o projeto e inspecionar o software em todas suas fases.

6.2.5 Posse individual do código (classes/features)

Deve ser elaborada uma lista contendo as funcionalidades e seus proprietários, de forma que o programador (que pertence à equipe de *features*) seja responsável pela funcionalidade a ser implementada, e depois possa responder por ela em caso de problemas na implementação.

Inspeções regulares

O código é inspecionado de forma a garantir que o que foi proposto foi feito e validado com o cliente. Essa inspeção é constante e deve ser realizada sempre que uma nova funcionalidade for implementada.

Gerenciamento de configuração e mudanças

Todas as funcionalidades implementadas devem ser testadas de forma a funcionarem com as demais. O gerenciamento de configuração trabalha com um controle de versão, a fim de saber o histórico das modificações nas *features*, além dos testes e das inspeções realizadas. Esses procedimentos garantem que as implementações instaladas sempre correspondam às últimas implementadas.

Relatório/visibilidade de resultados

A FDD sugere a utilização de um modelo gráfico para visualizar e medir o progresso do desenvolvimento do projeto. Esse modelo mostra cada *feature*, bem como a porcentagem de completude em que ela se encontra no momento. A metodologia indica também um relatório de toda a implementação, organizado por *features* e datas, desde a criação, implementação e possíveis modificações realizadas, que corresponderá a um histórico do desenvolvimento do software.

6.3 Utilização da metodologia FDD para um projeto de software

Para exemplificarmos a utilização da metodologia FDD para um projeto de software, tomamos como exemplo uma fábrica de suplementos alimentares, a Proforça, que necessita de um sistema completo para sua empresa. Após a realização de uma entrevista, o gerente de projetos constatou os requisitos para o desenvolvimento do software, descritos a seguir.

Hipoteticamente o problema da Proforça é adquirir um software confiável a curto prazo de tempo, que atenda a seus requisitos de forma a oferecer um mínimo de documentação a seus usuários.

6.3.1 Visão geral

A Proforça é uma empresa do ramo nutricional que produz e vende produtos para suplementação de atletas e consumidores adultos praticantes de atividades físicas. Atualmente possui duas fábricas, sendo a matriz em Miami/USA e sua filial em São Paulo/Brasil. A empresa conta com 210 colaboradores, sendo 132 em Miami e o restante no Brasil. Distribui seus produtos a lojas no atacado, ou seja, não vende diretamente ao cliente final. As lojas estabelecem suas formas de venda livremente e fazem seus pedidos mensalmente. Os controles atuais são feitos de forma manual ou em planilhas eletrônicas. A empresa investe em equipamentos de informática, mas não possui nenhum sistema informatizado a não ser anotações individuais em planilhas eletrônicas e editores de texto. Considerando este cenário, a empresa necessita de um sistema com as seguintes funcionalidades básicas:

a) O sistema deve ser instalado tanto na matriz quanto na filial, devendo ser bilíngue (inglês/português) de forma a facilitar sua utilização. Existem dois servidores disponíveis para instalação do software com sistema Linux instalado, juntamente com um *link* dedicado à empresa para uso em período integral.

b) A empresa dispõe de um departamento de informática dedicado à infraestrutura, ou seja, as instalações físicas podem ser solicitadas a essa equipe. As máquinas tanto da empresa quanto da filial utilizam o sistema operacional Windows Seven©.

c) O software deve controlar os distribuidores que atualmente são mantidos em ficha, contendo na versão Brasil: Código Sequencial, Razão Social, CNPJ, Endereço, Bairro, Cidade, Telefone, E-Mail e Nome do Contato. Na versão USA: ID, Company Name, Address, City, State, Zip Code, Primary and Second Phone Number, Email, Contact. Esse controle deve permitir a inclusão, consulta, relatórios e alteração dos distribuidores e, no caso da exclusão, permitir apenas a função "inativar" distribuidor.

d) O sistema deve permitir que o próprio distribuidor crie uma conta de acesso via Internet para verificar como está o status de seus pedidos diretamente na página da empresa (extranet). Essa funcionalidade deve ser atualizada em tempo real pelos funcionários de cada setor da fabricação dos produtos. Setores, a saber: Matéria-Prima, Engenharia Química, Mistura e Preparo, Embalagem, Qualidade e Logística. Esses setores devem estar cadastrados no sistema, devendo permitir a inclusão, consulta, alteração, exclusão e relatórios (que serão definidos posteriormente).

e) Será necessário também que o sistema controle os funcionários, seus horários de trabalho e medidas de produção. Os dados que a empresa possui sobre o funcionário são: nome, endereço, bairro, cidade, telefone, e-mail e foto; os demais dados estão no sistema de folha de pagamento e contabilidade, que é terceirizado. O sistema deve controlar o quanto o funcionário rende em termos de trabalho para a empresa, bem como as horas trabalhadas para executar suas tarefas. A ideia é estabelecer, por meio de relatórios, metas de produção e premiar os funcionários que as atingirem.

f) Um controle completo dos produtos enquanto matérias-primas também será necessário, bem como uma forma de o sistema notificar a necessidade de comprá-los (estoque mínimo). Dos produtos são necessários dados como: ID, descrição, tipo, peso, sabor, cor, unidade, quantidade em estoque, validade e fabricante. Dados de validade devem também ser informados caso uma matéria-prima esteja perto de vencer, podendo assim a empresa fazer as devidas devoluções ou utilizações.

Considerando as necessidades levantadas pela entrevista com os responsáveis pela empresa, podemos agora seguir alguns passos propostos pela metodologia FDD, para dar início ao desenvolvimento do software.

PASSO I - Modelagem de objetos do domínio

Tomando como base os requisitos listados, imagina-se que o **gerente de projeto** tenha visitado a empresa e, de alguma forma, levantado esses requisitos. Em linhas gerais, o levantamento de requisitos inicialmente pode ser feito em uma ou mais entrevistas. Durante elas, o gerente de projeto pode lançar mão de questionários de apoio que podem ser submetidos a todos os envolvidos a fim de captar diferentes visões sobre o sistema. Na prática, o gerente de projeto deve estar acompanhado de um analista de negócios ou especialista em requisitos para realizar essas entrevistas.

A partir dos requisitos levantados, o gerente de projeto modela os objetos do domínio, que pode ser feito por um diagrama que proporciona uma visão macro do sistema, semelhante ao ilustrado pela Figura 6.1.

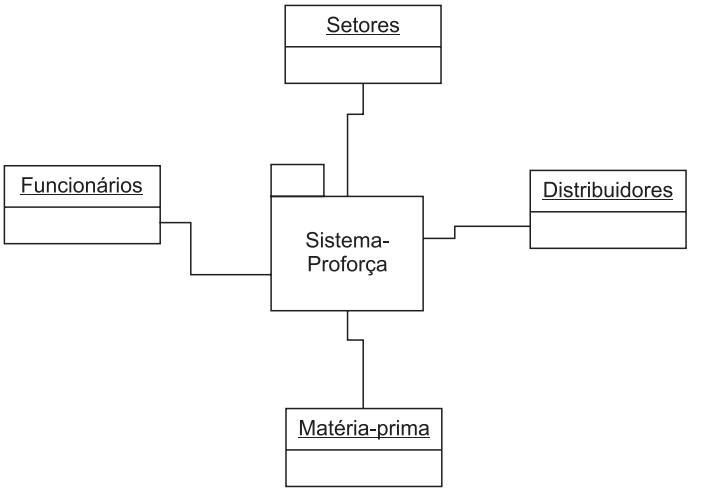


Figura 6.1 – Visão macro do sistema. Fonte: Arquivo pessoal do autor.

Nesse momento, o gerente de projeto tem uma lista preliminar de todos os componentes do sistema, porém sem um detalhamento das funcionalidades (lista feita em conjunto com o analista de negócios ou de requisitos). Para essa próxima fase, a FDD propõe que esse detalhamento seja realizado em grupo, proporcionando ao modelo maior abrangência, além de contar com a opinião de todos. Cabe também ao gerente escolher e definir a equipe que irá desenvolver o projeto em questão, bem como definir suas atribuições.

PASSO II - Regras de negócio

Após a criação do modelo de objetos do domínio, um **especialista** ou o maior conhecedor da área de negócio da empresa deve criar um relatório com as regras de negócio, ou seja, documentar como os processos da empresa funcionam. Esse relatório deve descrever em detalhes como tudo funciona na prática. Esse especialista, que também é chamado de **arquiteto-chefe**, precisa se reunir com o *stakeholders* para discutir todos os processos.

O resultado será um documento feito em comum acordo entre as partes interessadas, sendo, de preferência, assinado pelos envolvidos, evitando assim especulações posteriores. Cópias desse documento devem estar disponíveis para consultas de todos os envolvidos. Em outras palavras, nesse momento se estabelece o **documento de requisitos** oficial do projeto, que pode ser feito utilizando um *template*, como, por exemplo, o *template Volere* que, apesar de sua simplicidade, é o resultado de anos de prática, consultas e pesquisas relacionadas à engenharia de requisitos. Essa recomendação contempla um formulário de requisitos, que serve como guia para descrever cada exigência do projeto. Uma visão de uso desse formulário pode ser observada na Figura 6.2.

Ficha de Especificação de Requisito
(baseada no *template Volere*)

Identificação do Requisito: RF001 (Cadastro de distribuidores)

Tipo de Requisito: Funcional

Caso(s) de Uso(s) Vinculado(s): DCU 001 - Cadastros

Descrição:

O sistema deverá permitir que sejam armazenados em forma de cadastro os distribuidores da empresa. Esses distribuidores possuem alguns dados como: Código Sequencial, Razão Social, CNPJ, Endereço, Bairro, Cidade, Telefone, E-Mail e Nome do Contato. O CNPJ deverá ser validado conforme exigência da Receita Federal da União, e os demais dados não podem ser deixados em branco durante o preenchimento.

Observação: o sistema deverá disponibilizar também uma versão em inglês dessa ficha, bem como todas as suas funcionalidades.

Solicitante(s): Equipe de *stakeholder* (nome do envolvido)

Prioridade: 4 (alta)

Material de Apoio: Diagrama de Casos de Uso (DCU01), Diagrama de Classes (DC002)

Histórico: Solicitação Inicial (10/01/2011)

Solicitação de Mudança (___/___/___)

Figura 6.2 – Ficha de requisitos Template Volere©. Fonte: Atlantic Systems Guild (2006).

PASSO III - Lista de *features*

Nesse momento, com o documento de requisitos em mãos, a equipe de modelagem pode começar a criar uma lista de funcionalidades. Trata-se de uma listagem contendo informações detalhadas sobre o que deve ser desenvolvido, ordenadas por prioridade (captadas no documento de requisitos). Deve ser levado em conta o tempo de desenvolvimento de cada *feature*, ou seja, das funcionalidades, características ou recursos, de forma a poder agregar benefício ao cliente, ou seja, que algo seja implementado e testado nesses intervalos de tempo. Os requisitos devem ser separados em forma de funcionalidades, as chamadas *features*, obedecendo a três categorias:

- Áreas de Negócio;
- Atividades de Negócio;
- Passos da Atividade de Negócio.

A Tabela 6.1 apresenta um exemplo de lista de funcionalidades que poderiam ser levantadas a partir dos requisitos propostos para o nosso sistema exemplo.

Lista de Features			**Empresa:** Proforça
			10/01/2012
	Responsáveis: João Carlos Cintra, Roberto Manoel e Priscila Fantini		
Área de Negócio			
Requisito	Prioridade	Data entrega	Responsável
RNF 001 - Sistema Bilíngue	4	imediata	Rafaela Dias
RNF 002 - Plataforma Linux	4	imediata	Guimarães Rosa
RNF 003 - Banco de Dados	...	...	...
...			
Atividades de Negócio			
Requisitos	Prioridade	Data entrega	Responsável
RF001 - cadastro de distribuidores	4	06/02/2012	Sérgio Rosa
Passos da Atividade de Negócio			
Requisito	Prioridade	Data entrega	Responsável
RF001 - Validação CNPJ	4	06/02/2012	Sérgio Rosa

Tabela 6.1 – Exemplo de lista de features. Fonte: Arquivo pessoal do autor.

Essa lista de funcionalidades, além de favorecer o rastreamento, refina os requisitos, possibilitando um maior detalhamento do que deve ser produzido. Em alguns casos, em projetos maiores, pode ser necessário uma **equipe de planejamento** formada por representantes da empresa e da equipe de desenvolvimento, a fim de decidir as prioridades do projeto. Essa equipe de planejamento, entre outras coisas, definirá a sequência de desenvolvimento e atribuirá um conjunto de *features* aos programadores-chefes.

PASSO IV - Refinar a lista de *features*

A partir da lista de funcionalidades, o **programador-chefe**, que normalmente é o integrante da equipe mais experiente em termos de implementação, cria o modelo de objetos, que representa uma separação dos requisitos em partes que possam garantir pequenos pacotes a serem liberados para o cliente. Imagine um conjunto de funcionalidades que o cliente já pode ir utilizando enquanto novos pacotes são feitos. Conforme vimos anteriormente, esta é uma das premissas das metodologias ágeis. O programador-chefe também escolhe a linguagem de programação, as ferramentas e o tipo de banco de dados que irá suprir as necessidades do cliente e do sistema proposto.

Cabe ao programador-chefe ainda delegar os proprietários dos códigos gerados por cada entrega "*build*" e conferi-los em suas funcionalidades por meio de testes unitários, estruturais e de desempenho. Aconselha-se também gerar um documento contendo todos os minipacotes listados, quem são seus proprietários, bem como um mecanismo de controle de versão, além de datas de compilação e resultados dos testes. Pode-se também utilizar uma ferramenta para controle de alterações e diagnósticos dos testes solicitados e realizados, como, por exemplo, o Mantis, ilustrado na Figura 6.3, que é uma ferramenta para controle de defeitos no software gratuita, desenvolvida na linguagem PHP e que utiliza interface web com banco de dados MySQL.

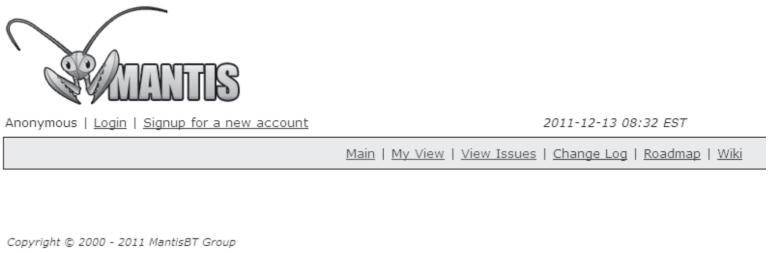


Figura 6.3 - Visão da interface do Mantis Bug Tracker.
Fonte: Copyright© 2000 - 2011 MantisBT Group.

Após o refinamento, pode-se criar o relatório de acompanhamento para medir o progresso do desenvolvimento. Esse relatório apresenta uma visão detalhada do andamento, devendo ser constantemente atualizado. Um sistema de cores pode ser utilizado ou um padrão de texturas. As cores adicionam um padrão semântico ao modelo, e visualmente conseguimos saber detalhes da implementação, além de padronizar o entendimento. O progresso (ou status) pode ser reportado por algumas texturas ou cores, como exemplificado na Figura 6.4.

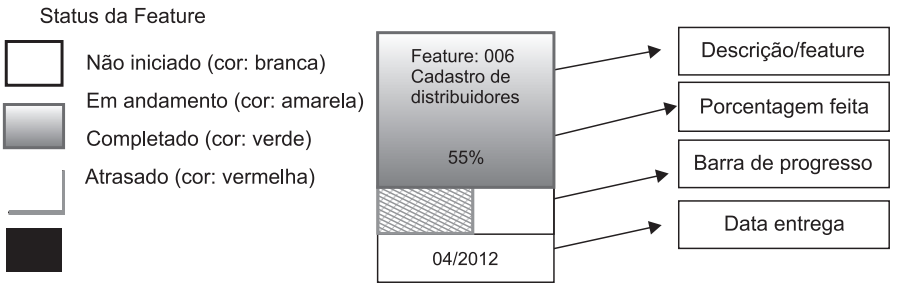


Figura 6.4 - Padronização do progresso de cada feature. Fonte: Arquivo pessoal do autor.

Observe que, quando temos todas as funcionalidades juntas, podemos gerar um relatório de progresso completo, denominado *parking lot*, conforme ilustra a Figura 6.5.

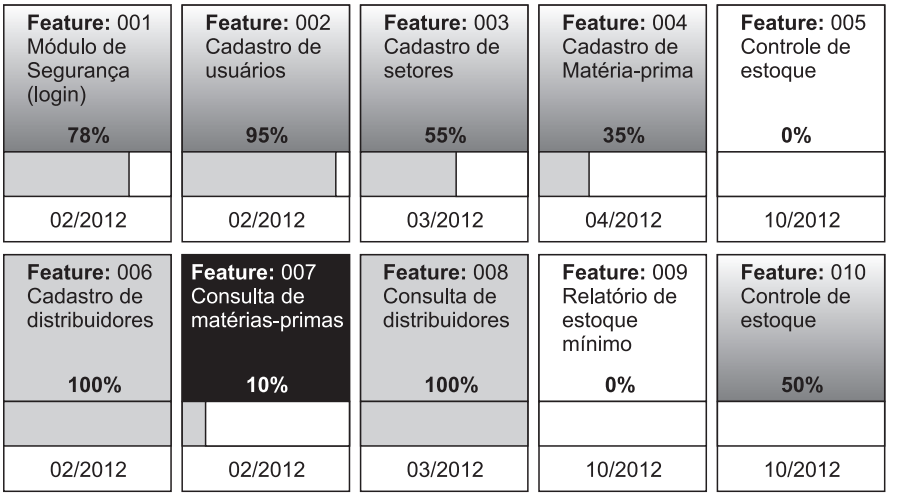


Figura 6.5 – Exemplo de relatório de progresso (parking lot). Fonte: Arquivo pessoal do autor.

Esse tipo de relatório permite ao gerente de projeto visualizar quais *features* precisam de mais atenção, quais ainda nem começaram a ser implementadas, além de proporcionar um meio fácil capaz de dar um *feedback* aos *stakeholders* sobre o andamento do projeto. Outros mecanismos e gráficos podem se somar a estes e reportar o andamento, como, por exemplo, o ***Feature Monitor***, ilustrado pela Tabela 6.2.

Feature	Descrição	Programador-chefe	Responsável pelo código	Iniciado		Interface		Insp. Interface		Codificado		Insp. do código		Liberado (build)	
				P	A	P	A	P	A	P	A	P	A	P	A
001	Módulo segurança (login)	Robson	Nilma	02-jan	03-jan	04-jan	04-jan	04-jan	05-jan	06-jan	07-jan	08-jan	08-jan	09-jan	
002	Cadastro de Usuários	Robson	Manoel	06-jan	06-jan	07-jan	07-jan	07-jan	08-jan	08-jan		09-jan			
003	Cadastro de Setores	Robson	Manoel	06-jan	06-jan	07-jan	07-jan	07-jan	08-jan	08-jan		09-jan		10-jan	
004	Cadastro de Matéria-Prima	Robson	Nilma	06-jan	06-jan	07-jan	07-jan	07-jan	08-jan	08-jan		09-jan		10-jan	
005	Controle de Estoque	Robson	Anderson	06-jan		07-jan		07-jan		08-jan		09-jan		10-jan	
006	Cadastro de Distribuidores	Robson	Nilma	02-jan	03-jan	04-jan	04-jan	04-jan	05-jan	06-jan	07-jan	08-jan	08-jan	09-jan	10-jan
007	Consulta de Matérias-primas	Robson	Anderson	06-jan	06-jan	07-jan		08-jan		09-jan		09-jan		10-jan	
008	Consulta de Distribuidores	Robson	Nilma	02-jan	03-jan	04-jan	04-jan	04-jan	05-jan	06-jan	07-jan	08-jan	08-jan	09-jan	10-jan

Tabela 6.2 – Exemplo de uso do monitoramento de features (Feature Monitor). Fonte: Arquivo pessoal do autor.

PASSO V - Implementação

Com a relação de todas as *features* em mãos, agora o programador-chefe poderá encaminhá-las, refinadas e organizadas, à **equipe de features**. Esse encaminhamento pode ser gradual ou total; normalmente se opta pela entrega gradual, ou seja, considerando cada conjunto predefinido pelo programador-chefe e de acordo com as prioridades. Caso a FDD tenha sido combinada com outra metodologia (*Extreme Programming* ou *Scrum*), outras técnicas podem ser utilizadas, como, por exemplo, a programação pareada usada na *Extreme Programming* (XP), abordada em outro capítulo. A cada implementação de uma *feature* ou conjunto delas, um registro deve ser lançado em um **relatório de *features* implementadas**, que deve conter informações complementares, conforme descrito na Tabela 6.3.

Relatório de Features Implementadas					
			Empresa:		Proforça
			Programador-Chefe		Robson Gomes
Data	**Responsável**	**Descrição**	**Caso de uso vinculado**	**Diag. de classe vinculado**	**Histórica de modificações**
21/02	Gabriela	Feature 006 - Segurança (login e senha)	DCU 06	DC 01	Validar usuário
21/02	Rafael	Feature 007 - Distribuidores	DCU 02	DC 01	Cadastro de distribuidores
25/02	Gabriela	Feature 006 - Segurança (login e senha)	DCU 06	DC 01	Permissões
...					

Tabela 6.3 – Exemplo de relatório de features implementadas. Fonte: Arquivo pessoal do autor.

Observe na Tabela 6.3 que a mesma funcionalidade pode aparecer lançada várias vezes, porém com históricos diferentes. É importante que cada implementação seja inspecionada pela equipe de *features*, cujos membros, de preferência, devem submeter o código a uma sequência automatizada de testes. Recomenda-se em um primeiro momento aplicar um teste caixa-preta, apenas para garantir os resultados, para posteriormente utilizarmos uma inspeção mais detalhada nos métodos, um teste caixa-branca, abrindo, analisando e anotando os resultados em um relatório de inspeção específico. Finalmente, cada *feature* ou lista delas são encaminhadas ao programador-chefe, que as promove para "*build*", ou seja, as disponibiliza para o usuário final.

PASSO VI - Finalização

Após os passos anteriores e à medida que o desenvolvimento for avançando, o gerente pode retirar desse processo uma vasta gama de conhecimentos para serem utilizados em outros projetos futuros de software. Além disso, o gerente pode avaliar sua equipe quanto ao ritmo de desenvolvimento e conseguir precisar prazos para novos projetos, levando em conta o que ela produziu.

Os exemplos utilizados neste capítulo são meramente ilustrativos e serviram apenas como base para mostrar o uso da metodologia FDD. Obviamente, em projetos maiores, os relatórios e as listas devem ser redimensionados. A entrega do projeto de software para a empresa solicitante deve ser acompanhada de um contrato com garantias e de um plano de manutenção, oferecendo assim a possibilidade de futuras versões.

6.4 Casos de sucesso

A seguir, algumas empresas especializadas na utilização da metodologia FDD:

designIT

Existem inúmeras empresas que adotaram a FDD como metodologia; algumas combinam o seu uso com outras metodologias a fim de obter melhores resultados. Um exemplo é o da empresa designIT relatado por seu gerente de projetos, disponível no endereço http://www.sitepoint.com/successful-development/. A empresa não possuía nenhuma metodologia, fazia seus projetos sem nem mesmo um controle de versões. A cada projeto alguns problemas se repetiam e seus prazos nunca eram cumpridos conforme o combinado. A utilização da metodologia FDD foi fundamental para a existência da empresa, que se reestruturou empregando-a em todos seus projetos.

Tangiblesolutions

Outro caso de sucesso, relatado por Grant Cause, consultor sênior de empresas como a wwTangiblesolutions, descreve a FDD como a solução para projetos de pequeno a grande porte, e coloca suas experiências no site http://www.methodsandtools.com/archive/archive.php?id=19. Nesse artigo, descreve como se surpreendeu com a palestra de Jeff DeLuca em um seminário organizado pela Borland. Ele achava que a FDD seria apenas mais uma metodologia e não fazia ideia de como seria útil e perfeita para seus propósitos como consultor.

6.5 Exercícios

1) O que é uma *feature* para a metodologia FDD?
2) Quais as melhores práticas da FDD? Cite cinco.
3) Para quais números de integrantes (equipe) a FDD é ideal?
4) Quais diagramas da UML podem ser utilizados pela metodologia FDD?
5) Que tipos de teste são sugeridos pela metodologia FDD?

7

Dynamic Systems Development Methodology (DSDM)

Objetivos

- *Apresentar as origens e características básicas da metodologia.*
- *Exemplificar como a DSDM pode ser utilizada na prática.*

7.1 Origens e características básicas

Dynamic Systems Development Methodology (DSDM) é uma metodologia de desenvolvimento de projetos de software centrada em estabelecer os recursos e o tempo fixo para o desenvolvimento de um projeto, ajustando suas funcionalidades de maneira a atender os prazos estipulados. Foi criada em 1994 e continua sendo mantida por um consórcio de diversas empresas associadas.

Sua origem se deu no Reino Unido por uma organização sem fins lucrativos e não proprietária que reúne membros como Hewlett Packard, BT, British Airways, Ministério da Defesa, PWC, Vodafone e KPMG. Alcançou notoriedade no desenvolvimento rápido de aplicativos RAD (*Rapid Application Development*), tornando-se assim um framework para o desenvolvimento de projetos. Para o termo framework podemos imaginar um conjunto de funcionalidades que podem ser utilizadas em vários projetos, um conjunto de classes que podem servir de base para aplicações pertencentes a um mesmo domínio de um problema. Em linhas gerais, podemos considerar um framework como uma abstração que une códigos comuns entre vários projetos. Sendo assim, algumas linguagens de programação possuem seus próprios frameworks a fim de ajudar o desenvolvedor, promovendo funcionalidades previamente programadas ou permitindo que ele crie seus próprios recursos.

O consórcio mantém uma página na Internet com suas atualizações, treinamentos e certificações, que pode ser acessada no endereço www.dsdm.og. Também podemos encontrar um fórum de discussões sobre a metodologia no mesmo web site. A estrutura do DSDM baseia-se em nove princípios, que são considerados boas práticas de utilização dessa metodologia. A seguir descrevemos resumidamente esses princípios, com o objetivo de caracterizar essa metodologia.

a) **Participação ativa dos usuários e *stakeholders*:** todos os envolvidos no projeto e conhecedores do processo devem acompanhar o desenvolvimento a fim de garantir que tudo seja entregue a tempo e de acordo com o solicitado.

b) **Abordagem cooperativa e compartilhada:** todas as partes interessadas devem cooperar e assumir o compromisso das entregas de partes do software, bem como escolher a ordem de sua implementação, ou seja, essas escolhas devem ser feitas em comum acordo.

c) **Equipes com poder de decisão:** as pessoas envolvidas no desenvolvimento devem ter conhecimento e autonomia suficiente para decidir o destino do sistema; não é tolerável aguardar decisões por um longo período de tempo.

d) **Entregas contínuas que fazem a diferença:** é um critério básico da DSDM tentar trazer um retorno ao cliente do projeto desde o início, a fim de promover a validação do que está sendo feito (fazer o produto certo).

e) **Desenvolvimento iterativo e incremental:** a ideia é convergir o sistema para o negócio e melhorá-lo continuamente em um processo iterativo, buscando e corrigindo os problemas o mais cedo possível.

f) **Feedback:** o foco está nas frequentes entregas de produtos de software, permitindo ao usuário colocar suas opiniões e solicitar modificações.

g) **Todas as possíveis alterações durante o desenvolvimento devem ser reversíveis:** deve-se permitir que o impacto das modificações seja testado e, caso não surta os efeitos desejados, as modificações possam ser restabelecidas.

h) **Fixar os requisitos essenciais:** os requisitos principais devem ser capturados no início, a fim de estabelecer os objetivos gerais. Os requisitos específicos serão norteados pelos principais e sujeitos a modificações, mas sempre focando os objetivos.

i) **Teste em todo ciclo de vida:** devido ao prazo normalmente apertado não podemos deixar a fase de testes exclusivamente no final da implementação, devendo ser realizada em todas as fases e componentes do projeto. O teste de regressão é normalmente o mais utilizado pela DSDM devido ao desenvolvimento incremental.

Nota:

"Podemos resumidamente definir o teste de regressão como sendo responsável por testar as funcionalidades do sistema após alguma alteração de grande risco, e isso inclui os segmentos já testados. É um teste do tipo caixa-preta, em que o código-fonte não é verificado e leva muito tempo para ser feito."

Além destes princípios, a DSDM possui alguns conceitos particulares que devem ser entendidos, como o conceito de "*Timeboxing*". Trata-se do encapsulamento do tempo reservado para o desenvolvimento das funcionalidades, por meio da divisão dos projetos em porções, cada uma com um orçamento próprio, e fixas por uma data de entrega predefinida. Um outro conceito importante é a técnica "MoSCoW", utilizada para auxiliar na definição das prioridades dos requisitos. É uma sigla derivada da frase em inglês "Must have this, Should have this if at all possible, Could have this if it does not affect anything else, Won't have this time but Would like in the future". Traduzindo, temos:

- TEM que ter isto;
- DEVE ter isto e se for possível completamente;
- PODE ter isto se não afetar o resto;
- NÃO VAI ter isto agora, mas SERIA bom ter no futuro.

Estas frases devem ser acrescentadas no momento em que se relacionam os requisitos, ou seja, na análise do projeto ou na ficha de requisitos, auxiliando assim a definição das prioridades. Um outro conceito é o de prototipagem, um dos principais fatores de sucesso da DSDM, pois possibilita a verificação e a descoberta antecipada de problemas e limitações do projeto de sistema.

7.2 Restrições e experiências

É importante observar que não é todo tipo de projeto de software que pode se adaptar ao DSDM, existindo algumas restrições que devemos conhecer. Por se tratar de uma metodologia ágil que exige pouca formalidade, os sistemas de segurança crítica, ou seja, sistemas que não podem falhar por colocar vidas em risco, não devem ser conduzidos com DSDM, bem como por outras metodologias menos rigorosas. Além de sistemas críticos à segurança humana, se não houver a participação efetiva dos clientes (*stakeholders*), se eles não estiverem disponíveis, e se não houver confiança e responsabilidade, os trabalhos podem ser comprometidos.

Os sistemas que não permitem a prototipagem ou arquitetura pouco modular e complexa também não são adequados ao uso dessa metodologia, ou seja, sistemas críticos que envolvem vidas humanas normalmente não podem ser prototipados, devem ser produzidos diretamente e com muita formalidade.

Existem inúmeros projetos que foram desenvolvidos a partir da DSDM e que tiveram essa experiência registrada por empresas ao redor do mundo, em relatos de *cases* de sucesso. Como exemplo podemos citar a empresa Sema Group, disponível em http://www.semagroup.com.au/, que utiliza a DSDM em inúmeros projetos. Uma outra empresa, a Boston Globe, desenvolveu um projeto chamado Sysdeco utilizando a DSDM. Sysdeco é um sistema voltado para gerenciamento e produção de jornais, produzido em quatro meses com o uso de DSDM, após ter tido uma tentativa fracassada de 18 meses empregando técnicas tradicionais de desenvolvimento. Observe que os métodos tradicionais utilizados para esse projeto não envolviam os usuários, enquanto a DSDM exigiu a participação de todos os interessados.

Um outro *case* de sucesso foi implementado pela *Scottish Natural Heritage* em conjunto com a *Newell & Bugde,* unificando informações de sistemas existentes e disponibilizando registro de parcerias com relatórios disponíveis via web. Os trabalhos começaram em fevereiro de 2003 com

data de entrega para outubro do mesmo ano. Neste caso, a DSDM foi amplamente utilizada com prazos rigorosamente cumpridos. Maiores detalhes sobre o uso dessa metodologia no projeto citado podem ser encontrados em http://www.snh.org.uk/pdfs/publications/commissioned_reports/F02CI01.pdf. A DSDM também pode ser utilizada em sistemas para projetos de novos produtos (em que os tempos para execução são apertados), novas tecnologias ou reengenharia de projetos já existentes. Projetos de âmbito comercial, multidisciplinares e ergonômicos, sempre com participação coletiva, são os ideais para o uso dessa metodologia. Sempre que um projeto for urgente, criativo e não estiver nas restrições citadas ele pode ser um forte candidato a DSDM.

7.3 Fases da DSDM

A DSDM possui cinco fases distintas para aplicação de sua metodologia: estudo da viabilidade, estudo de negócio, modelo de iteração funcional, projeto e construção de iteração, e implementação. Apresentamos a seguir detalhes das fases citadas.

Estudo da viabilidade

Essa fase deve ser feita apenas uma vez durante o projeto. Como o nome sugere, obviamente tem o propósito de analisar a viabilidade do projeto, ou seja, se é possível concebê-lo com o uso da DSDM. Nessa fase, observa-se a organização como um todo, listando as pessoas envolvidas e mapeando os riscos a serem enfrentados. Analisam-se também as técnicas a serem utilizadas, bem como as ferramentas de trabalho. Esse estudo não deve passar de algumas semanas; duas ou três são consideradas tempo suficiente para deliberar sobre a viabilidade. Como produto final, são criados um relatório de viabilidade e um plano geral de desenvolvimento, que serão demonstrados ainda neste capítulo.

Estudo de negócio

Nessa fase, as regras de negócio são analisadas, bem como todos os processos envolvidos, a fim de captar as características do negócio. Uma forma de executá-la é propor vários workshops, reuniões com representantes da empresa capazes de argumentar e decidir sobre os aspectos relevantes do sistema a ser implementado. Esses *stakeholders* devem chegar a um acordo sobre as prioridades de desenvolvimento e dividi-las em áreas de negócio, refinando o plano geral de desenvolvimento. Uma lista de prioridades deve ser alcançada e informações sobre as classes e seus responsáveis (pessoas-chave) devem fazer parte dela.

Um exemplo dessa lista também será apresentado neste capítulo. Outros artefatos devem ser criados a partir desse estudo de negócio; um deles se relaciona com a definição da arquitetura do sistema e outro com o plano de prototipagem. Devem indicar, respectivamente, um esboço de como o sistema será implementado e a estratégia de criação para as fases seguintes. Finalmente, vem o plano de gerenciamento de configuração, que indica os resultados dos testes e manutenção.

Modelo de iteração funcional

Trata-se de um planejamento de conteúdo e abordagem aplicado após a execução de cada iteração para analisar se os resultados estão corretos com o objetivo de calibrá-los. Nesse modelo, as funcionalidades são analisadas e implementadas, resultando em um protótipo que serve como modelo e

experiência para melhoria de todo o processo de iteração. Os protótipos resultantes não precisam ser totalmente descartados e podem ser aproveitados e incluídos no sistema final. Um modelo funcional é produzido como resultado dessa análise, contendo códigos e protótipos. Os testes são imprescindíveis nessa fase, e a partir deles podemos melhorar e/ou incrementar o que foi feito. Há ainda quatro saídas dessa fase: atualização da lista de prioridades, lista de comentários dos usuários para iterações subsequentes, lista dos requisitos não funcionais e documento de análise de risco.

Projeto e construção de iteração

É a fase em que o sistema é implementado de fato. A saída dessa fase é um sistema testado e confrontado com os requisitos iniciais, de acordo com as necessidades dos usuários. Sendo assim, os envolvidos revisam o sistema e comentam seus resultados, dando um *feedback* aos desenvolvedores. Nessa fase, a lista de comentários dos usuários também pode ser atualizada.

Implementação

Finalmente chegamos à fase que coloca o sistema em funcionamento no ambiente real. Nesse momento os usuários devem ser treinados a utilizarem todo o sistema ao mesmo tempo em que os resultados do uso do sistema são analisados. Isso pode levar algum tempo, e nesse período a implementação pode ser iterada, ou seja, pequenos ajustes podem ocorrer. Este é o momento em que devemos entregar um manual do usuário e um relatório final de análise do projeto. Este último refere-se ao desfecho do projeto, bem como a todos os detalhes da implementação, com base nos resultados colocados de forma técnica. Nessa fase pode ocorrer a percepção de novos requisitos que devem ser implementados, pois determinadas funcionalidades são percebidas no final do projeto. Neste caso, um novo modelo funcional deve ser feito e novos prazos e recursos devem ser definidos, além de ocorrer o planejamento de novas iterações.

7.3.1 Equipe de projeto

A DSDM apresenta 15 (quinze) papéis de usuários (*stakeholders*) e desenvolvedores, que podem ser atribuídos ou não de acordo com as proporções do projeto. A seguir faremos uma breve descrição dos propósitos desses papéis.

a) **Coordenador técnico/arquiteto:** responsável por definir qual será a arquitetura do sistema, controla o gerenciamento de configuração e coordena o time de desenvolvedores a utilizar todos os princípios da DSDM e das ferramentas escolhidas.

b) **Desenvolvedor sênior:** líder de equipe e maior conhecedor das ferramentas de desenvolvimento, devendo dar suporte a toda equipe (também conhecido como arquiteto de software).

c) **Analista:** experiente conhecedor das técnicas de análise e modelagem, responsável pelas entrevistas com os clientes e a condução dos workshops, capta os requisitos e os interpreta para os programadores, podendo utilizar a UML como apoio.

d) **Programador:** codificador, responsável por traduzir requisitos em implementações de código de máquina na linguagem escolhida.

e) **Designers:** responsável pela construção das interfaces, conhecedor da ergonomia dos sistemas de informação e sua arquitetura.

f) **Testadores:** profissionais responsáveis por criar o ambiente de testes em todo o ciclo de vida do sistema, bem como promover esses testes e coletar seus resultados.

g) **Usuário embaixador:** representa todo o corpo de usuários, devendo ser capaz de tomar decisões em nome de todo o grupo; deve também conduzir e conferir se todo o desenvolvimento corresponde aos requisitos. Tem poder de decidir os rumos da implementação, além de revisar toda a documentação. Deve ter tempo exclusivo para o projeto e não necessita ser um funcionário de alto escalão; basta ter o respeito da comunidade de usuários.

h) **Usuário visionário:** garante o projeto do ponto de vista empresarial, devendo conhecer todos os objetivos do negócio, bem como interagir diretamente com o usuário embaixador. Porém, diferentemente do embaixador, não precisa conhecer exatamente os detalhes dos processos, mas deve entender perfeitamente os objetivos da empresa. Deve estar disponível em todo o tempo de projeto e promover as decisões no nível de gestão.

i) **Usuário conselheiro:** acompanha o dia a dia do trabalho de automação, normalmente o usuário final do sistema, pode ter conhecimentos de TI ou simplesmente ser um funcionário de algum setor a ser informatizado. É interessante que esse usuário tenha um bom poder de comunicação, haja vista que ele deve aprovar e fornecer o *feedback* sobre as funcionalidades implementadas.

j) **Patrocinador executivo:** responsável pelas decisões financeiras da empresa. Normalmente representa alguém do alto escalão com capacidade de julgar e decidir sobre os investimentos em recursos do projeto.

k) **Redator técnico:** acompanha todo o desenvolvimento e testes das implementações a fim de construir os manuais de usuário do software, bem como apostilas de treinamento e atas de reuniões realizadas (também conhecido como documentador).

l) **Desenvolvedor:** conhecedor dos diagramas da UML, responsável por implementar junto com o programador os testes de unidade.

m) **DBA (*Data Base Administrator*):** profissional conhecedor dos sistemas gerenciadores de banco de dados, que exerce o papel de especialista de apoio à equipe de desenvolvimento em questões específicas. Modela e desenvolve a base de dados considerando seu conhecimento sobre o negócio.

n) **Configurador:** especialista em instalações de software, com altos conhecimentos sobre plataformas de utilização e redes de computadores.

o) **Suporte técnico:** profissional responsável por captar e coordenar os chamados de helpdesk, geralmente pertencendo ao quadro de funcionários da própria empresa. De preferência deve ter formação na área de TI, com conhecimentos técnicos de sistemas e habilidade para traduzir problemas dos usuários em solicitações aos desenvolvedores.

As funções citadas podem sem acumuladas pelos profissionais envolvidos dependendo do tamanho e da amplitude do projeto. Obviamente que em projetos menores não é necessário ter tantos envolvidos. A DSDM sugere de duas a seis pessoas envolvidas, que podem acumular as funções descritas como uma equipe, e alguns projetos podem ter mais de uma equipe envolvida. A razão de indicar no mínimo duas pessoas para integrar um projeto que utiliza a metodologia DSDM parte do princípio de que uma delas desempenharia o papel de desenvolvedor e a outra o usuário

do sistema. A razão de indicar seis integrantes no máximo é que um projeto pode ser dividido em partes, ou seja, os componentes podem ser desenvolvidos por equipes diferentes e integrados no final. Obviamente, apenas reforçando, os 15 papéis podem ser acumulados pelas seis pessoas que a DSDM sugere.

7.4 Utilização da metodologia DSDM em um projeto de software

Vamos utilizar um *case* hipotético como exemplo de abordagem da metodologia. Para isso, vamos observar a visão geral do sistema proposto:

A empresa SVS deseja informatizar todos os seus processos, que atualmente contam com relatórios feitos a mão ou digitados em planilhas eletrônicas. Atualmente, apenas as partes relacionadas à contabilidade e à folha de pagamento não necessitam da automação, pois são realizadas por terceiros.

A empresa trabalha no ramo de equipamentos eletrônicos, mais especificamente com isoladores, para-raios e chaves de fusível. Utiliza vários insumos (matérias-primas) na fabricação desses produtos, e alguns envolvem elementos com venda controlada e rastreada pela Anvisa (Agência Nacional de Vigilância Sanitária), mas não oferecem riscos diretos à vida humana, mesmo porque são instalados em conjunto com componentes elétricos nas vias públicas, ou seja, nas ruas. Uma falha no sistema apenas implicará um produto com defeito de fabricação. O sistema proposto deve controlar todo o processo de fabricação até a logística de entrega dos produtos, bem como o sistema de faturamento da empresa. Vamos ver agora como aplicar a metodologia DSDM a este cenário, por meio do estudo prático da aplicação de suas fases.

7.4.1 Fase 1 - Visão do projeto/estudo da viabilidade

Nessa fase, verifica-se se o projeto é adequado ao uso da DSDM. Por meio de um relatório de viabilidade, podemos verificar se o projeto vai ao encontro das necessidades apontadas. Podemos observar na Tabela 7.1 um exemplo desse relatório aplicado ao cenário apresentado. Os nomes citados são fictícios apenas para contextualizar melhor o projeto.

Projeto:	**SVS - Chão de Fábrica**	**Data:**
		Janeiro
Documento de Visão		
1. Introdução		
Este documento descreve em detalhes uma proposta de projeto de software para a empresa SVS Ind. Ltda. Apresenta uma visão de todas as funcionalidades e limites do sistema, além de caracterizá-lo. O documento traz também uma lista de todos os envolvidos no processo, os chamados *stakeholders*, que representam pessoas das mais diferentes áreas de atuação, que terão contato direto com o software desenvolvido. No final será feita uma explanação do produto a ser implementado, com todas as suas regras de negócio e indicações de uso.		

2. Descrição do problema

A empresa SVS necessita controlar todas as suas ações. A empresa não possui nenhum tipo de controle automatizado, suas informações ainda são feitas de maneira manual por meio de formulários impressos e preenchidos pelos funcionários dos setores. Devido a isso os resultados nem sempre conferem com o esperado, ou seja, quando os cálculos de gastos convergem com as vendas, sempre há dúvidas sobre os valores de seus lucros. Uma solução seria a implementação de um software capaz de registrar todas as ações, que vão desde a matéria-prima e custos de fabricação até o produto final. Se os setores forem controlados por esse software, no final de cada mês a SVS terá números reais e poderá tomar decisões com base neles. Outro benefício seria a possibilidade de os clientes visualizarem seus pedidos pela web e acompanhá-los enquanto atravessam as fases dentro da empresa, evitando assim ter de informar sobre o andamento do pedido na fábrica, pois o próprio cliente visualiza isso na Intranet. Muitas funcionalidades do software ocorrerão por ideias da empresa e a SVS não terá problemas em se adaptar a elas. Isso significa que, como não há automação de processos, algumas sequências de tarefas executadas pelos funcionários terão de mudar.

3. *Stakeholders*

Lista dos interessados e colaboradores do projeto.

Nome	Descrição	Responsabilidade
Joel Carlos Meneses	Auditor financeiro	Aprova financiamento e recursos
Carla Soares Gurgel	Supervisora de fábrica	Monitora o progresso do projeto
Gilberto Guiller	Gerente de produção	Usuário mais experiente e conhecedor
Sandra Morelli Rafah	Analista de TI	Infraestrutura e sistemas operacionais
Pedro Henrique Sefo	Engenharia	Supervisão de projetos
Mara Ana Fernandes	Gestora	Administração e gestão da informação

4. Ambiente

A empresa SVS está dividida em sete setores, cada setor com uma quantidade diferente de profissionais, sendo em sua maioria relacionados à produção. Os processos de fabricação são contínuos; isso implica que a empresa não para suas atividades, atuando com três turnos diários. O sistema deve atuar primeiramente no setor de produção, tratando desde a matéria-prima, e estabelecer o produto, registrando e automatizando seus processos. Para esse setor, a supervisora e o gerente de produção, citados como usuários anteriormente, darão todo o suporte. Os processos e as tarefas envolvidas não devem mudar com o tempo. Isso pode ocorrer no caso de novos lançamentos de produtos, em que todo o processo pode ser readequado, porém em todos os casos a parte que se refere ao software continua a mesma. Explicando um pouco melhor, as matérias-primas chegam por um processo de compras em que os fornecedores são selecionados por melhor oferta de preço e qualidade, sendo em seguida lançadas em um livro e controladas de forma manual. Esses insumos vão desde parafusos até misturas químicas utilizadas para o acabamento das peças, muitas vezes de efeito isolante. O setor de produção solicita via requisição essa matéria-prima e dá início ao processo de fabricação. Vários funcionários manipulam essa matéria-prima que passa por vários subsetores do setor. Cada setor faz um lançamento do que está ocorrendo com o material até chegar ao final do ciclo no subsetor de qualidade, que avalia os inúmeros itens de qualidade do produto. Após liberação do produto ou de um lote de produtos, eles serão transportados para o setor de logística que realiza o processo de embalagem, etiqueta e registro (manual) de fabricação, depois os separa para atender os pedidos de despacho para a transportadora. Os pedidos são realizados via setor de vendas, que também faz seus controles de forma manual ou com o uso de planilhas eletrônicas. Essas planilhas não precisam ser conectadas ao software desde que este faça um processo para tal. A empresa conta com uma infraestrutura de equipamentos conectados em rede. Como não possui nenhum software específico, aceita sugestões e novas configurações para uso do sistema de software proposto.

5. Visão geral do sistema

Neste primeiro momento será necessário um sistema de chão de fábrica, que controle todos os processos de fabricação citados anteriormente. A prioridade será controlar as entradas e saídas de matéria-prima, lotes de fabricação, registro do andamento e manufatura, lançamento das peças fabricadas, das possíveis perdas, registro das embalagens atrelados aos pedidos recebidos, despacho, nesta ordem. Obviamente cada uma dessas funcionalidades automatizadas vai exigir relatórios gerenciais e consultas de todos os tipos possíveis. O sistema deve estar preparado para saídas de suas informações também via web, incluindo um controle de acesso por níveis de usuário. Um tempo de resposta deve ser considerado dentro das limitações dos equipamentos da empresa e dentro dos padrões de acesso externo. No final, na integração com os pedidos, o sistema deve ter condições de gerar o faturamento baseado nos pedidos faturados e estabelecer relatórios que apontem o custo, os gastos e o lucro para tomadas de decisões.

Necessidade	Prioridade	Funcionalidades	Liberação
Controle de insumos	1	Cadastrar, consultar, alterar os insumos	Janeiro 2011
Cadastro de setores	1	Cadastrar, consultar, alterar os setores	Janeiro 2011
Cadastro de subsetor	1	Cadastrar, consultar, alterar os subsetores	Janeiro 2011
Controle de lotes de fabricação	2	Cadastro de lotes via código de barras	Fevereiro 2011
Registro manufatura	2	Tabela de registro de informações sobre serviços executados	Fevereiro 2011
Cadastro de produtos em estoque	2	Cadastrar, consultar, alterar os produtos	Fevereiro 2011
Registro de perdas	2	Cadastrar, consultar, alterar as perdas	Fevereiro 2011
Controle de pedidos	3	Cadastrar, consultar, alterar os pedidos	Março 2011
Cadastro de transportadoras	4	Cadastrar, consultar, alterar transportadoras	Março 2011
Controle de acesso	4	Login dos usuário por nível	Março 2011
Página web	4	Home page para acesso à Intranet	Março 2011
Embalagens	5	Cadastrar, consultar, alterar	Abril 2011
Despacho	5	Cadastrar, consultar, alterar	Abril 2011
Relatórios gerenciais	3	Emissão de relatórios para tomadas de decisões	Janeiro/Abril 2011

Outros requisitos		
Requisito	**Prioridade**	**Liberação**
Disponível em plataforma Windows ou Linux	1	Janeiro 2011
Sistema de login via web	2	Março 2011
Tempo de resposta compatível com os equipamentos	1	Janeiro 2011
Manual do usuário	3	Abril 2011
Treinamento	2	Março/Abril 2011

Tabela 7.1 – Visão e estudo da viabilidade.

O exemplo apresentado na Tabela 7.1 serve apenas como apoio para o entendimento de como relatar os possíveis elementos de viabilidade do sistema. Obviamente o exemplo é hipotético e poderia ter muito mais detalhes. Serve como base para decisões contratuais e mostra os limites do projeto, onde começa e onde termina do ponto de vista dos clientes interessados. É possível também analisar as variáveis para definir as ferramentas, os recursos e negociar os prazos, haja vista que esta é a visão do cliente sobre o que ele necessita, ou seja, suas expectativas.

7.4.2 Fase 2 - Estudo do negócio/workshops

Antes de qualquer coisa, separe a equipe de trabalho de acordo com os papéis definidos pela DSDM e estabeleça um cronograma de reuniões. Pode-se fazer uma lista com os nomes e suas colaborações como sugerido pela Tabela 7.2.

Membro da equipe	Função no projeto
Joel Carlos Meneses	Patrocinador Executivo
Carla Soares Gurgel	Usuário Embaixador
Gilberto Guiller	Usuário Conselheiro
Sandra Morelli Rafah	Usuário Conselheiro
Pedro Henrique Sefo	Usuário Embaixador
Mara Ana Fernandes	Usuário Visionário
Rael Alcântara	Desenvolvedor Sênior
Priscila Rampazzo	Arquiteto de Software e DBA
Cirilo Ramos Cardoso	Analista e Programador
Gabriel Antares Coimbra	Desenvolvedor e Programador
Fabiana Senna	Designer e Testador
Sebastien Grogie	Desenvolvedor e Programador

Tabela 7.2 - Organização do projeto (membros da equipe).

Em seguida, estabeleça um cronograma para apresentar (em um workshop) a essas pessoas suas funções e como se darão o relacionamentos. Não se esqueça de apresentar as atribuições dos usuários colaboradores, que neste caso não as conhecem. Percebe-se que existem funções repetidas nessa lista, o que indica mais de uma equipe envolvida no projeto.

Esta é apenas uma sugestão de divisão de responsabilidades. Outras ordens podem ser utilizadas, respeitando os limites impostos pela DSDM, que foram vistos anteriormente. Os workshops devem ser conduzidos pelo desenvolvedor sênior juntamente com um representante da empresa, que normalmente o escolhe. Apresente na primeira reunião a visão anteriormente refinada e uma lista de como o projeto será acompanhado em suas fases.

7.4.3 Fase 3 - Análise funcional

Nessa fase, cria-se a lista de itens de trabalho, descrita como o plano geral de desenvolvimento. Normalmente contém os requisitos detalhados, os casos de uso e a lista de riscos que o projeto possui. A Tabela 7.3 apresenta um modelo de itens que uma lista de riscos deve conter.

ID	Data	Título	Descrição	Tipo	Impacto	Probabilidade	Efeito	Proprietário
1	01/11	Rotatividade	Pessoal experiente pode deixar o projeto	Projeto	Atraso	Alta	1	Rael
2	01/11	Atraso	Atraso na especificação dos requisitos	Produto	Prazo	Baixa	2	Cirilo
3	01/11	Tecnologia	Mudança na tecnologia	Negócio	Custo	Moderada	1	Joel

Tabela 7.3 - Lista de riscos do projeto.

O efeito descrito na Tabela 7.3 refere-se a uma variação que pode ser livremente criada pela organização desenvolvedora, composta de fatores indicativos no caso de algum risco ocorrer, como, por exemplo: Efeito 1 = Catastróficos, Efeito 2 = Sérios e assim por diante. O detalhamento dos requisitos pode ser feito com o uso de uma *template* ou de uma lista, como podemos observar na Tabela 7.4.

RF001 - Cadastro de Insumos				**Oculto ( )**
Descrição: A partir da seleção no menu o sistema deverá oferecer uma interface para cadastrar insumos (matéria-prima). Devem também ser solicitadas ao usuário informações como descrição do insumo, tipo, peso, unidade, cor, custo, quantidade, fornecedor, data do cadastro, nota fiscal de compra e observações. O sistema deverá então gravar as informações no banco de dados.				
Requisitos não funcionais agregados				
ID	**Descrição**	**Categoria**	**Desejável**	**Permanente**
RNF 001	Código do insumo automático	Controle	(x)	(x)
RNF 002	Dados do fornecedor automáticos	Especificação	(x)	( )

Tabela 7.4 - Requisitos funcionais e não funcionais.

Todos os requisitos podem ser descritos desta forma, Tabela 7.4, e colocados juntos em um documento de requisitos, bem como os diagramas de casos de uso, classe, atividades e sequência produzidos conforme a UML. Com esses artefatos é possível ter uma definição detalhada das regras de negócio da empresa, realçando os passos mais importantes. A saída dessa fase é o modelo funcional que servirá de base para a criação do primeiro protótipo da aplicação, além da arquitetura do sistema, pois com esses artefatos podemos visualizar todo o projeto.

7.4.4 Fase 4 - Desenho/projeto

Após o refinamento de todos os requisitos e a aprovação por parte dos envolvidos, pode-se estabelecer o plano de prototipagem, que nada mais é do que uma etapa do desenvolvimento que visa atingir metas e objetivos da empresa de forma a analisar as entradas e saídas produzidas. As

entradas referem-se às condições e hipóteses captadas na execução dos processos e as saídas àquilo que se espera do sistema (entrada - processamento - saída).

Esse plano deve conter as classes criadas, seus responsáveis e os diagramas que apresentem a sequência de execução das tarefas e funcionalidades. Com relação à linguagem UML, pode-se utilizar todos os diagramas necessários ao projeto, exigindo que no mínimo haja os **diagramas de classe**, **casos de uso** e **sequência/atividade**, imprescindíveis a qualquer projeto.

Vale a pena salientar que esse conjunto básico de artefatos deve ser utilizado a título de "pacote mínimo" em todas as metodologias de desenvolvimento ágil. Após esse plano também ter sido refinado e aprovado, podemos dar início ao(s) protótipo(s) propriamente dito(s). Em paralelo a isso, o redator do projeto, ou um desenvolvedor responsável, começa a monitorar o progresso da aplicação, podendo utilizar uma ferramenta de apoio ou simplesmente anotações contendo a funcionalidade, datas de início e fim, responsável, versão, comentários do usuário. Esse relatório está exemplificado na Tabela 7.5, e servirá também para acompanhar e resolver os problemas apontados pelo usuário, resultando na lista de comentários citada anteriormente.

Projeto: SVS - Chão de Fábrica						
ID	**Funcionalidade**	**DT início**	**DT final**	**Responsável**	**Versão**	**Comentários**
RF111	Cadastro de usuários	01/03/2011	01/04/2011	Rael	1.2	Cadastro OK

Tabela 7.5 - Exemplo de controle e monitoramento de progresso.

A Figura 7.1 apresenta um modelo gráfico de acompanhamento do projeto em função do tempo e dos requisitos.

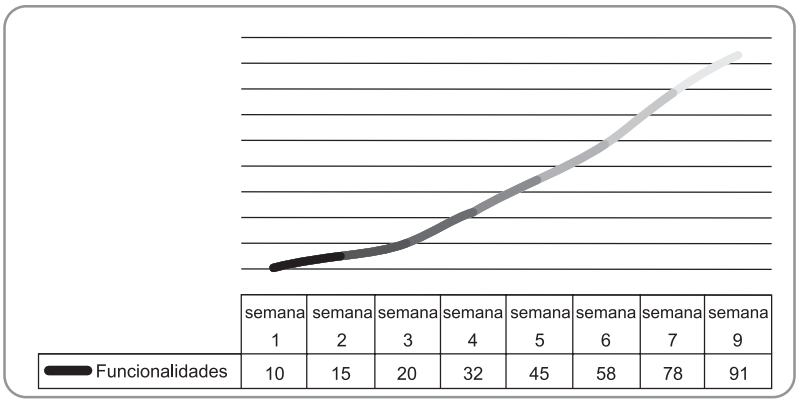


Figura 7.1 - Gráfico de funcionalidades DSDM.

7.4.5 Fase 5 - Implementação/implantação

Finalmente,chegamos à fase de implementação, na qual o software é efetivamente instalado na íntegra para os usuários. Após os protótipos serem testados, criticados e alterados de acordo com o cliente, seus módulos são integrados e colocados para o funcionamento definitivo. Para cada um dos módulos é preciso providenciar o manual do usuário, que deve ser feito em conjunto com o redator

e os desenvolvedores, a fim de descrever todas as telas criadas e suas funcionalidades. O manual do usuário deve conter, além de explicações tela a tela, informações sobre como agir em casos de erros.

De uma forma geral, é interessante que o software esteja livre de erros causados por má operação, ou seja, erros provocados por inconsistência dos dados informados. Como exemplo, podemos deixar de preencher um campo ou realizar um preenchimento equivocado dos dados. De qualquer modo, as mensagens de informação nesse sentido podem ser codificadas e colocadas com maiores detalhes no manual. O manual do usuário minimamente deve conter:

a) **Introdução os software:** explicar o que é o software, quais os recursos disponíveis na versão.

b) **Informações de registros:** licença de utilização.

c) **Procedimentos de instalação passo a passo:** lista de exigências de hardware e software mínimos para instalar.

d) **Guia de funcionalidades tela a tela:** explicar com detalhes cada tela, bem como formas de preenchimento dos campos e procedimentos (inclusão, alteração, consultas, relatórios).

e) **Lista de erros comuns codificados:** uma espécie de FAQ de ajuda ao usuário, escrita com detalhes e explicando cada um dos erros.

f) **Suporte e contato:** uma lista de formas de contato com os desenvolvedores (empresa). Em caso de helpdesk on-line, passar os endereços (URLs); em caso de suporte, informar os nomes e telefones disponíveis.

Depois de instalado o software na rede ou nos computadores, o processo deve ser monitorado pelos envolvidos, e um plano de gerenciamento de configuração deve ser preparado, contendo todos os detalhes e possíveis formas de utilização dos resultados oriundos do sistema. Esse plano de gerenciamento deve apresentar todas as possibilidades e recursos para proveito máximo dos dados obtidos, possibilitando a criação de outros relatórios gerenciais, recursos e funcionalidades.

O plano apresenta também formas de instalar novas versões do software, como manter sua distribuição, configurar os serviços, entre outros. Um relatório final descrevendo os limites do sistema para efeitos contratuais deve ser feito e assinado pelas partes envolvidas, fechando assim o que foi acordado e abrindo possibilidades de melhoria em novos projetos. As garantias também devem fazer parte do plano, bem como os procedimentos de chamados e visitas técnicas. Os exemplos de tabelas, listas e figuras apresentados neste capítulo podem ser melhorados e modificados, pois serviram apenas como exemplo de utilização da metodologia DSDM.

7.5 Exercícios

1) A DSDM baseia-se em nove princípios para conduzir os projetos de software. Quais são eles?

2) Como podemos definir o termo "*Timeboxing*" na DSDM?

3) Qual o significado da técnica MOSCOW na DSDM?

4) Qual a diferença entre o usuário visionário e o usuário conselheiro na DSDM?

5) A que se refere a fase de estudo da viabilidade na DSDM?

Anotações

8

Adaptative Software Development (ASD)

Objetivos

- *Apresentar as origens e características básicas da metodologia ASD.*
- *Exemplificar como a ASD pode ser utilizada na prática.*

8.1 Origens e características básicas

Assim como ocorreu com outras metodologias ágeis, o ASD (*Adaptative Software Development* ou Desenvolvimento Adaptável de Software) foi criado com base na necessidade de implementar projetos de software de maneira rápida. Ele traz um conjunto de perspectivas embasadas nas metodologias tradicionais, as quais agilizam o desenvolvimento e ao mesmo tempo garantem um software de qualidade para o cliente.

Essa metodologia ágil foi criada por Sam Bayer e James (Jim) Highsmith, em 1997, partindo de anos de experiência em metodologias tradicionais somados às técnicas RAD (*Rapid Application Development* ou Desenvolvimento Rápido de Aplicativos), que utilizam ferramentas de apoio ao desenvolvedor. Uma das páginas mantidas para divulgar o projeto ASD está disponível em www.adaptivesd.com, criada pelo autor James Highsmith em 2002. A seguir, vamos explorar algumas características dessa metodologia.

Emergência x Complexidade

Sabe-se que as empresas, sempre que decidem implementar uma solução de software, indicam que a informação está se perdendo de alguma forma e/ou a falta de sua organização está gerando

algum tipo de prejuízo. Sendo assim, independentemente da complexidade do software, existe um anseio de que a aplicação fique pronta o mais rápido possível. Por falta de conhecimento dos administradores, muitas vezes são exigidos prazos exorbitantes no desenvolvimento, ou seja, torna-se inviável fazer um projeto com qualidade em um prazo tão pequeno. Os gerentes de TI acabam repassando essa exigência aos desenvolvedores, que nem sempre têm voz ativa ou poder de decisão para argumentar sobre isso.

A metodologia ASD procura apresentar alternativas de desenvolvimento às metodologias tradicionais, de forma a evitar que algumas fases tomem um tempo maior do que deveriam e possibilitar que gerem imediatamente artefatos de valor ao cliente. Permite ainda dividir o software em partes a serem implementadas e testadas, atendendo as emergências independente de sua complexidade.

8.2 Ciclo de vida ASD

A metodologia ASD trata seu ciclo de vida levando em conta três itens, a saber: a especulação, a colaboração e o aprendizado. A Figura 8.1 ilustra graficamente essa visão.

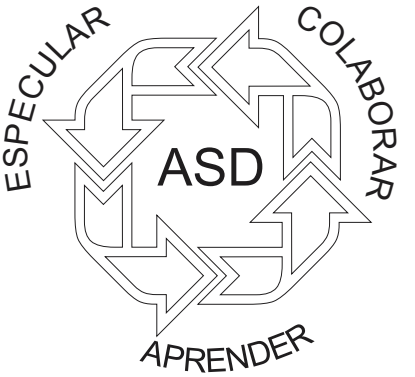


Figura 8.1 - Ciclo de vida ASD. Fonte: Arquivo pessoal do autor.

8.2.1 Especular, colaborar e aprender

Os termos especular, colaborar e aprender muitas vezes se convergem, se invertem e se misturam. Quando se especula, aprende-se; quando se aprende, colabora; quando se colabora, invariavelmente se especula novamente. A seguir apresentamos uma breve descrição de como cada item desse ciclo se comporta.

a) Especular

O termo especular equivale a observar, indagar, pesquisar; em outras palavras, questionar as causas de algum assunto. No caso da metodologia ASD, utiliza-se como substituto do "planejar". O planejamento é considerado muito amplo nessa metodologia, pois quando se planeja, busca-se o entendimento e definem-se estratégias para atendê-lo. Assim, antecipadamente, o ASD estimula a criação de um conjunto de regras e metas que muitas vezes precisam ser modificadas ao longo do desenvolvimento. Entende-se que, ao planejar, não temos ideia se tudo irá ocorrer conforme o

previsto inicialmente. Portanto, o ASD utiliza o ato de especular como subterfúgio ao replanejar, ouvindo as necessidades do cliente ao longo do processo de desenvolvimento com o objetivo de evitar o retrabalho.

b) Colaborar

Quando o software a ser desenvolvido é considerado de alta complexidade, as previsões de como ele será implementado e de como vai se comportar no ambiente de utilização muitas vezes podem se mostrar inúteis devido ao fato de não conseguirmos controlá-las de antemão.

Conforme apresentamos anteriormente, existem práticas de gestão que atuam em partes do processo de desenvolvimento desde que estas sejam previsíveis, mas para projetos que possuem elementos e variáveis desconhecidos, os métodos de gestão tradicionais não se aplicam, ou seja, não funcionam a contento. O PMBOK, por exemplo, é uma metodologia considerada tradicional, em que os processos precisam ser bem definidos, e alguns autores radicais consideram impossível trabalhar em conjunto com uma metodologia ágil.

A colaboração, neste contexto, traduz-se no ato de criação e manutenção de elementos de software capazes de atender as emergências por parte do cliente. O gerente de projeto, juntamente com os *stakeholders*, decide as prioridades, ou seja, aquilo que é previsível em curto prazo, fazendo com que sejam gerados benefícios aos usuários. É importante observar que a colaboração não visa "apagar incêndios" ou resolver problemas de processos da empresa. A colaboração pretende organizar o desenvolvimento, de forma a permitir que as funcionalidades sejam realizadas em ordem de utilização e suprindo as necessidades mais urgentes.

c) Aprender

A metodologia ASD propõe que as partes interessadas participem efetivamente do desenvolvimento da solução a ser construída. Determina que todo artefato desenvolvido seja apresentado a todos para que valorizem o que foi produzido, ou seja, utilizam práticas de revisões junto com o cliente e testes com versões betas, a fim de obter resultados para novas análises e modificações. O aprendizado está no fato de que, com o andamento do projeto, o desenvolvedor passa a conhecer os desejos do cliente, adquirindo experiência e domínio do assunto e, consequentemente, da aplicação, além de conhecer antecipadamente os próximos passos a serem desenvolvidos. Os ciclos de revisões e testes devem ser curtos o suficiente para aprender apenas com pequenos erros, não oferecendo grandes riscos ao projeto.

8.3 Como começar a desenvolver um projeto de software utilizando ASD

Vamos tomar como exemplo um projeto fictício da empresa SMA Agro, que atua no ramo de vendas de agrotóxicos, situada no interior paulista e com filiais no sul do Estado de Minas Gerais. A empresa possuía, no ano de 2003, uma estrutura desorganizada e apenas uma loja, e nessa época problemas surgiam a todo tempo. A empresa até então era inscrita no modelo fiscal "simples" e emitia suas notas fiscais de venda manualmente, preenchendo a caneta os produtos vendidos e os dados para geração de impostos.

Com o crescimento da empresa, observado pelo aumento da demanda por produtos, a SMA Agro precisou urgentemente melhorar seus processos de atendimento, bem como as emissões de suas notas fiscais. Um dos principais problemas no início era a exigência da Receita Federal de que as empresas automatizassem suas emissões, fazendo com que tudo fosse fiscalizado com mais facilidade. Outra questão observada foi o aumento de vendas no estado vizinho (Minas Gerais), estimulando a SMA Agro a expandir seus negócios, abrindo uma filial no sul de Minas. Com a nova filial, a empresa observou que passou a gastar muito com os valores de frete e logística de um estado para outro, gerando outro problema a ser resolvido. Este cenário fez com que a empresa observasse que tanto a matriz quanto a filial precisavam ser informatizadas, estabelecendo um link direto com a matriz para evitar que as vendas superassem seus estoques, além de permitir um melhor planejamento e uma tomada de decisões mais rápida.

Em um primeiro momento contrataram uma empresa de desenvolvimento de software sob medida e em um período de seis meses a empresa desenvolvedora documentou os requisitos e começou a desenvolver o software pretendido. Durante todo esse tempo, que se mostrou insuficiente, apenas os cadastros básicos foram implementados e a comunicação entre matriz e filial era feita por disquetes, que continham uma cópia dos bancos de dados, com o objetivo de serem unificados na matriz.

A empresa desenvolvedora recebeu todo o valor combinado referente ao desenvolvimento do sistema, mas não cumpriu os prazos acordados, colocando a culpa na empresa solicitante. Mesmo assim, essa situação continuou por mais seis meses, até que a SMA Agro recebeu sua primeira multa por não cumprimento ao estabelecido pela Receita Federal. Então, em meados de 2005, os administradores da empresa finalmente decidiram cancelar o contrato com a empresa desenvolvedora. Para que o problema não se repetisse, contrataram uma equipe de TI a fim de ela mesma desenvolver e manter sua própria solução.

Nesse momento, também decidiram abrir, simultaneamente, mais duas filiais, o que poderia aumentar ainda mais os requisitos e as necessidades do sistema, bem como ocasionar mais multas por falta de um sistema informatizado. A equipe de TI que foi contratada era formada por seis profissionais, um gerente de TI com experiência ampla em desenvolvimento, um analista de negócios com experiência principalmente em metodologias ágeis, dois programadores com conhecimentos variados (incluindo testes de software), um especialista em banco de dados e modelagem (UML) e um designer de aplicações web que também tinha experiências anteriores no uso de metodologias ágeis. Enfim, formavam uma equipe concisa e coesa em suas atribuições, com profissionais recrutados no mercado e em universidades da região da SMA Agro.

Essa nova equipe de TI decidiu adotar a metodologia ASD, em comum acordo com o gerente de TI e o analista de negócios, que entendiam a urgência requerida pela empresa (que na ocasião havia se tornado um grupo). Analisando os requisitos, perceberam que o sistema em si não oferecia grande complexidade, embora a distribuição das informações pelas filiais merecesse destaque, pois precisava ser altamente confiável e dependia de um link VPN para funcionar. O link VPN é uma ligação entre uma empresa e outra (por exemplo, matriz e filial, que utiliza a Internet como meio de comunicação).

A seguir, vamos comentar dez fases de melhores práticas no uso da ASD, utilizadas no desenvolvimento de software mencionado no cenário apresentado anteriormente.

Fase 1

O primeiro passo discutido pela equipe de TI diz respeito aos limites do software a ser desenvolvido; foram listados os requisitos funcionais e não funcionais em forma de histórias. Essa fase foi

conduzida pelo gerente de TI juntamente com os *stakeholders* interessados no sistema. Também é necessária a presença do analista de negócios ou, no caso de não haver esse profissional, um engenheiro de requisitos. Observe que pode não ser fácil chegar a um acordo comum sobre esses limites em apenas uma reunião, fazendo com que outras desse tipo sejam agendadas.

O produto dessa fase é um documento de brainstorming que representa um apanhado organizado de todas as ideias, pensamentos e possíveis requisitos informados pelos usuários do software. A Figura 8.2 apresenta um exemplo de layout que mostra como pode ser feito um documento de brainstorming pelo gerente de TI e o analista de negócios.

Brainstorming	
Projeto: <<SM Agro - Sistema de Armazenagem e Distribuição de Insumos Agrícolas>>	
Data: <<23/07/2006>>	Duração: <<45 minutos>>
Participantes:	
<<Gerente de TI>>	<<Analista de negócios>>
<<Stakeholders>>	
Objetivo: <<Módulo de Vendas>>	
Descrição: <<Relacionar todas as ideias>>	Tipo
Utilizar o cadastro de produtos dos fornecedores em vez de criá-lo	Descartado
Vendas efetuadas diretamente via Internet com uso de tablets	Futuro
Utilizar RF-ID para baixar estoque	Possível
Cadastrar todos os produtos devidamente com todos os dados, inclusive dados do Fisco	Imediato

Figura 8.2 – Documento de brainstorming. Fonte: Arquivo pessoal do autor.

Fase 2

Após as reuniões de brainstorming, os requisitos que já estão classificados de acordo com suas prioridades devem ser reescritos em um formulário, com o objetivo de detalhar mais e confrontar com as histórias do usuário. Essas histórias devem ser feitas pelos *stakeholders* em uma linguagem natural, ou seja, com as próprias palavras, descrevendo o que precisam com o máximo de precisão e objetividade. Essas histórias darão uma ideia do esforço necessário para cumpri-las durante o desenvolvimento e servirão como instrumento de inspeção e validação do produto gerado e não devem ser confundidas com os requisitos nem com casos de uso. A Figura 8.3 apresenta um modelo de ficha de história do usuário.

História - Cadastrar Produtos	**Oculto ( )**
Descrição: Deverão ser solicitados os dados de identificação do produto, como descrição, peso, cor, tamanho, nome científico, fabricante, vencimento, categoria, tipo, código da Anvisa, composição química, contraindicações (se possível, incluir a bula digitalizada), data de fabricação, sugestão de uso, informações de segurança em caso de acidentes, procedimentos de descarte da embalagem.O código do produto poderá ser o código de barras impresso na embalagem e, se possível, permitir o autopreenchimento de informações como datas e categorias que podem ser previamente cadastradas.	

Figura 8.3 – Modelo de história do usuário. Fonte: Arquivo pessoal do autor.

Fase 3

Nessa fase, iniciam-se os protótipos do software pretendido, em que os requisitos começam a ser modelados e a se transformar em programas implementados. A modelagem na metodologia ASD utiliza conceitos e técnicas da UML por meio de diagramas de apoio. Normalmente, os diagramas mais importantes e imprescindíveis são os casos de uso, diagrama de classes e diagrama de atividades, mas outros diagramas podem ser inseridos de acordo com as necessidades do projeto. Em seguida, um protótipo é implementado com base nesses diagramas e em seus respectivos requisitos. O protótipo não tem pretensões diretas de ser o software final e deve servir para obter-se um *feedback* dos *stakeholders* sobre a parte do software que está sendo implementada no momento.

As práticas de especular, aprender e colaborar citadas no início do capítulo são aplicadas visando assim atender o cliente da melhor forma. Para a empresa SMA Agro foram implementados vários protótipos ao longo do desenvolvimento, de forma modular e devidamente apresentados para os envolvidos.

A partir das histórias descritas pelo usuário, deve-se criar a lista dos requisitos funcionais e não funcionais, e podemos utilizar qualquer *template* para tal. Esses requisitos devem conter detalhes técnicos sobre a funcionalidade a ser desenvolvida, bem como a relação entre os diagramas de apoio que servirão como suporte. Na Figura 8.4 podemos observar a relação entre o requisito e seus diagramas.

RF11 - Cadastro de produtos Caso de uso vinculado: DCU02 <<diagrama de casos de uso>> Diagrama de classe vinculado: DC01 <<diagrama de classes>>			Oculto ( ) <<caso seja uma funcionalidade sem interface para o usuário>>	
Descrição: Este requisito terá como objetivo cadastrar o produto. Deverão ser solicitados os dados de identificação do produto, como descrição, peso, cor, tamanho, nome científico, fabricante, vencimento, categoria, tipo, código da Anvisa, composição química, contraindicações, data de fabricação, sugestão de uso, informações de segurança em caso de acidentes, procedimentos de descarte da embalagem.				
Requisitos não funcionais				
Nome	**Restrição**	**Categoria**	**Desejável**	**Permanente**
NF11.1 Identificação do produto com código de barras	Não permitir produtos diferentes com o mesmo código de barras	Especificação	(X)	(X)
NF11.2 Preenchimento de dados	Não permitir que campos sejam deixados em branco (descrição, fabricante em vencimento)	Interface	(X)	(X)
Analista/responsável: <<nome>> Data criação: <<data >> Data última modificação: <<data>>				

Figura 8.4 - Ficha de requisito. Fonte: Arquivo pessoal do autor.

Após as fichas de requisitos serem criadas e refinadas juntamente com seus respectivos diagramas, os protótipos podem ser implementados, dando início às iterações de desenvolvimento. Recomenda-se que as iterações ocorram em períodos de quatro a oito semanas no máximo, e no final sempre devem gerar alguma funcionalidade para o cliente. Quando consideramos o desenvolvimento de um protótipo, esse período deve ser mais breve, não ultrapassando quatro semanas.

Observe que a implementação dos protótipos pode ser parcial, ou seja, pode-se fazer um modelo que sirva de referência para outros; por exemplo, uma funcionalidade de cadastro de cliente pode servir de subsídio para uma de fornecedor. Sendo assim, após a aprovação dos protótipos, a codificação do sistema final poderá ocorrer mais rapidamente.

Fase 4

A partir dos protótipos construídos e a desmistificação dos requisitos, foi elaborado um plano de alto nível, que serviu de base para estabelecer o cronograma de desenvolvimento. Foi obtida então uma lista com os requisitos refinados, apontando os limites do software, ou seja, as funcionalidades que devem ser atingidas no primeiro momento. Esse plano serviu de apoio para formalizar o custo, os prazos, a quantidade de pessoas envolvidas, bem como o ROI (retorno do investimento) a ser apresentado para o cliente. Nessa etapa, mais uma vez foi apresentado para a empresa SMA Agro um planejamento com todas as variáveis a serem consideradas no projeto.

Fase 5

No início dessa fase foi necessário marcar uma reunião de lançamento de início do desenvolvimento. Todos os envolvidos foram convocados, e o gerente de TI, juntamente com o analista de negócios, apresentou os protótipos e o plano de alto nível anteriormente citado.

Os envolvidos puderam, na ocasião, criticar o trabalho realizado até então, sugerindo modificações, informando equívocos e até mesmo apontando possíveis ambiguidades dos requisitos anteriormente captados. Coube ao gerente de TI eleger um representante para lavrar uma ata da reunião, que foi posteriormente assinada pelos envolvidos.

Fase 6

Após a reunião de lançamento, o gerente convocou a equipe desenvolvedora para uma reunião de revisão, em que as críticas apontadas durante a reunião anterior foram novamente discutidas com o grupo, culminando com uma revisão dos requisitos de acordo com as necessidades. O grupo então chegou a um acordo sobre as modificações, realizando posteriormente alterações na modelagem que já havia sido feita. Nesse momento os protótipos foram analisados para possíveis aproveitamentos do código-fonte que foi implementado, descartando o que não interessava mais.

Fase 7

Nessa fase, definiu-se o ciclo de iteração, em que os requisitos são novamente analisados e divididos em períodos de tempo. A ideia é proporcionar uma entrega de valor ao cliente, possibilitando que ele vá utilizando o software gradualmente. Essa fase foi protagonizada pelo gerente de TI e o analista de negócios, que preferencialmente devem possuir certo conhecimento das limitações ou competências de sua equipe de desenvolvimento.

Caso a equipe possua um desenvolvedor sênior, ele também pode fazer parte das decisões desse ciclo de iteração. O artefato resultante dessa fase é um plano de iteração, que lista todos os ciclos em função do tempo para implementá-los. No caso que estamos estudando, a empresa optou por criar seu plano de iteração baseando-se na quantidade de desenvolvedores e profissionais, analisando a urgência do software e entendendo que a empresa já estava acumulando prejuízos e problemas.

Fase 8

Essa fase é dedicada à divisão das tarefas, uma vez que o plano de iteração mostrou quais foram as contribuições de cada desenvolvedor ao longo do projeto. Os ciclos são apresentados e criticados pelos integrantes da equipe, que podem sugerir mudanças ou apontar ajustes. É possível haver modificações que, ao longo do tempo, podem ser ajustadas. Nesse momento o software já está sendo produzido.

Fase 9

Posteriormente à implementação devem ser realizados os testes manuais e automatizados. Trata-se de uma bateria de lançamentos no sistema para analisar como ele se comporta. Esse tipo de teste pode ser implementado como se fosse um programa que lê uma base de dados fictícia e alimenta o software real, provocando erros que vão desde o preenchimento equivocado até múltiplos acessos, exigindo da performance.

Após esse procedimento, devem ser feitos os testes de campo e aceitação dos clientes. O software é instalado no cliente, que testa e encaminha seus *feedbacks* apontando irregularidades ou simplesmente concordando com as funcionalidades requisitadas. No caso da SMA Agro, os testes de campo foram fundamentais para o melhor funcionamento de todos os processos, mesmo porque a urgência do software exigiu que isso ocorresse rapidamente.

Fase 10

Finalmente chegou a fase da entrega do software, em que o cliente avalia o software implementado e recebe o termo de encerramento. Após as devidas conferências do produto e seus requisitos, dá-se início ao plano de manutenção que visa, além de corrigir problemas no futuro, apresentar propostas de melhorias nas novas versões do software. Novas versões e modificações foram realizadas na empresa apresentada, que obteve extremo sucesso em seus controles, gerando lucros e novos investimentos.

Um plano de manutenção deve começar com a decisão de desenvolver um novo sistema ou uma nova versão do sistema atual. O plano deve ser feito pelo gerente ou coordenador de projeto, que no início deve especificar como os usuários irão reportar suas solicitações de modificação no sistema. Além disso, o plano deve conter informações de custo (recursos orçamentários, financeiros e humanos) para realização das modificações. A norma ISO/IEC 14764 fornece as diretrizes base para criar esse plano.

8.4 Exercícios

1) Quais os itens abordados no ciclo de vida da metodologia ASD?

2) Para que serve o brainstorming na metodologia ASD?

3) Como podemos classificar as "histórias" dos usuários quanto à elicitação dos requisitos na metodologia de desenvolvimento adaptativo?

4) Qual a função do uso de protótipos na metodologia ASD?

5) De quanto tempo são as iterações na metodologia ASD?

9

Família Crystal de Cockburn

Objetivos

- *Apresentar as origens e características básicas das metodologias de Alistair Cockburn.*
- *Exemplificar como a Crystal Clear pode ser utilizada na prática.*

9.1 Origens e características básicas

Em meados da década de 1990, houve uma grande mudança de paradigmas nos métodos de programação de computadores, sendo uma época na qual havia estudos sobre programação orientada a objetos e técnicas de modelagens usando diagramas. Paralelamente, as pequenas e médias empresas ampliaram seus acessos às tecnologias, automatizando seus processos. Novos sistemas começaram a surgir e com alto grau complexidade, fazendo com que os desenvolvedores se adaptassem rapidamente a essa nova realidade. Os sistemas deixaram de ser feitos exclusivamente para grandes empresas e passaram a atender essa nova demanda. Considerando esse novo cenário, as equipes de desenvolvimento ao redor do mundo começaram a ter problemas com o uso das metodologias tradicionais, principalmente relacionados com o dimensionamento de projetos e das pessoas envolvidas. Então, em 1998, Alistair Cockburn criou uma família de metodologias com o objetivo de suprir as necessidades observadas e colaborar para a resolução desses problemas, batizando-a com o nome de família "Crystal".

Trata-se de uma família de metodologias com um código genético comum que atende diferentes tipos e tamanhos de projetos. Para uma melhor compreensão, essa metodologia foi dividida em cores; quanto mais escuro, mais crítico o sistema seria. Cada cor tem um propósito

diferente e elas são separadas de acordo com a criticidade, ou seja, projetos menores envolvem poucos desenvolvedores e, no caso de problemas, o prejuízo tende a ser menor. Já os projetos maiores ou de segurança crítica normalmente envolvem mais profissionais, portanto o prejuízo deve ser muito maior, podendo colocar a vida de pessoas em risco, como podemos observar na Tabela 9.1.

Cores	Número de desenvolvedores	Em caso de falha...
Clear	1-6	Perdem dinheiro, mas recuperam facilmente.
Yellow	7-20	Perdem dinheiro discretamente.
Orange	21-40	Perdem dinheiro substancialmente.
Red	41-100	Há perda substancial de dinheiro e, possivelmente, vidas humanas.

Tabela 9.1 - Divisão da família Crystal.

Cockburn mantém uma página desse projeto disponível em: http://alistair.cockburn.us/crystal. Em 2004, lançou um livro sobre uma das metodologias da família, o "*Crystal Clear*".

9.2 Princípios e filosofia da Crystal

Independentemente das cores, a família Crystal possui sete princípios básicos, que devem ser seguidos:

1) Trabalho face a face com o cliente: considera que envolver o cliente nas iterações e nas decisões é muito mais produtivo.
2) Peso significa custo: quanto maior a complexidade, maior o custo.
3) Usar metodologias diferenciadas para equipes maiores.
4) Mais cerimônias maior criticidade: quanto mais diálogos com os envolvidos melhor.
5) Comunicação eficiente (*feedback*) é melhor que entregas que não funcionam.
6) Habitabilidade: tolerância em lidar com seres humanos.
7) Eficiência no desenvolvimento.

Além destes princípios, essa família de metodologias considera algumas filosofias que auxiliam no desenvolvimento, fazendo com que os envolvidos entendam como lidar com a possibilidade de um novo projeto. Neste sentido, as metodologias de Cockburn indicam os seguintes itens:

Pessoas trabalham de maneiras diferentes

Devemos considerar que nem todos os desenvolvedores trabalham da mesma forma e não possuem os mesmos conhecimentos. As limitações de cada um podem fazer diferença durante o desenvolvimento, principalmente se considerarmos que uma fábrica de software pode trabalhar com projetos nas mais diferentes áreas de atuação.

Projetos da mesma área/tipo diferem em necessidades

Mesmo que o projeto seja desenvolvido para uma mesma área dentro de uma empresa, ou mesmo que seja desenvolvido para resolver os mesmos problemas e automatizar os processos, eles podem ser diferentes. Isso não significa que as funcionalidades e a lógica envolvida não possam ser reaproveitadas em outros projetos.

Comunicação intensa

No desenvolvimento de projetos de software a comunicação deve ser intensa, baseada sempre nas experiências dos envolvidos. Recomenda-se sempre ouvir as necessidades e ideias dos usuários, mesmo quando são equivocadas ou inviáveis para o momento.

Foco na qualidade

Fazer o necessário com muita qualidade é sempre melhor. Os recursos visuais que deixam o software mais atrativo só devem ser implementados caso haja tempo e apenas se os requisitos forem completamente atendidos com muita qualidade.

Evolução tecnológica

Considerar que as técnicas e as tecnologias mudam com o tempo, portanto novos conceitos de software e novos conceitos de hardware devem ser sempre considerados.

Experiências de valor

As pessoas aprendem muito mais com suas experiências do que a partir de uma metodologia. Novos projetos sempre demandam experiências adquiridas anteriormente. Isso evita que os desenvolvedores corram riscos que outrora já foram resolvidos ou previstos. Os projetos de software que fizeram uso dessa metodologia obtiveram sucesso devido às características de boa comunicação com o cliente, a eliminação do excesso de burocracia, ótimos testes de regressão e com entregas de funcionalidades rápidas e com frequência.

9.3 Ciclo de vida

A família de metodologias Crystal sugere um ciclo de vida baseado em integrações, onde tudo deve funcionar como um relógio. Um time de qualidade que siga suas filosofias e princípios pode obter resultados incríveis. Na Figura 9.1 podemos observar um modelo do ciclo de vida que considera o desenvolvimento de produtos de software com alta qualidade.

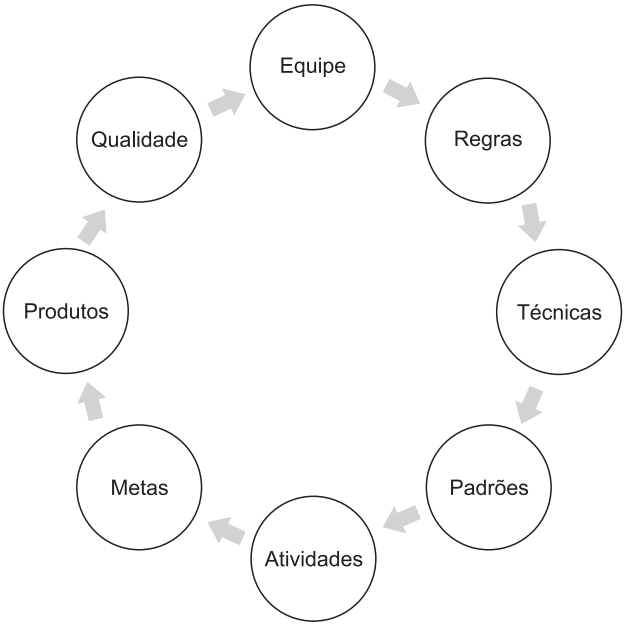


Figura 9.1 - Ciclo de vida da família de metodologias Crystal. Fonte: Arquivo pessoal do autor.

9.3.1 Equipes de projeto

Quando optamos por utilizar qualquer metodologia (ou cor) dessa família, devemos considerar uma "equipe base" de desenvolvimento que deve ser formada. A seguir apresentamos uma lista dos profissionais envolvidos, levando em conta que para equipes menores alguns podem acumular cargos (funções):

Patrocinador

Assim como em outras metodologias, o patrocinador, também chamado de *stakeholder*, é o responsável por ações de investimento financeiro da empresa. Cabe a ele julgar e decidir a aquisição de recursos para viabilizar a realização do projeto.

Coordenador de projeto

Realiza ações de coordenação e liderança da equipe de desenvolvedores. Normalmente é considerado o maior entendedor das práticas e filosofias da família de metodologias Crystal. Cabe a esse profissional dividir os trabalhos, coordenar as entregas, dialogar com os *stakeholders* e com o patrocinador, além de monitorar o progresso do desenvolvimento utilizando ferramentas de apoio e relatórios gerenciais. Em metodologias da família Crystal em que a equipe é muito grande, podem existir vários coordenadores, que devem agir como coordenadores de times, ou seja, pequenos grupos integrados de acordo com os módulos de desenvolvimento.

Analista de negócios

Em equipes menores, como, por exemplo, a *Crystal Clear*, o analista de negócios pode acumular funções como a de coordenador de projetos. Além de captar os requisitos do sistema, esse profissional é o responsável pela modelagem, proporcionando a validação e o rastreamento dos requisitos captados. Esse rastreamento pode ser feito simplesmente com uma boa identificação do requisito e assegurar que essa identificação seja acompanhada em toda implementação. Se as funcionalidades em suas fichas de requisitos possuírem um ID (número sequencial do requisito) e se os diagramas forem também numerados, facilita bastante para localizar o requisito e validá-lo, além de ajudar nos testes de regressão, explicados posteriormente.

Usuário stakeholder

O desenvolvimento de projetos utilizando família de metodologias Crystal sugere que um grupo de usuários esteja sempre disponível. O número de usuários vai depender do tamanho do projeto e de desenvolvedores envolvidos. No caso de uma equipe pequena, deve haver pelo menos um usuário responsável e conhecedor dos processos para acompanhar todo o desenvolvimento. No caso de a empresa não dispor de nenhum usuário, a utilização da família de metodologias Crystal deve ser questionada, pois segundo as premissas dessa metodologia, torna-se inviável a realização de qualquer projeto em qualquer metodologia da família Crystal sem pelo menos um usuário presente.

Designer/projetista

A arquitetura do sistema dever ficar a cargo desse profissional, bem como elementos da interface humano/computador. Esse profissional deve ter bons conhecimentos das facetas ergonômicas de um sistema e também das regras de criação de uma boa interface, seja ela voltada para aplicações web ou para desktop.

Programador/designer

Em uma equipe de desenvolvimento da família Crystal pode haver vários programadores. Cabe a eles promover a implementação do sistema e a sua integração nos casos modulares, além da codificação, adaptação e configuração inicial dos sistemas. A família de metodologias Crystal sugere a utilização de estilos padronizados de codificação, o que implica a adoção de "*design patterns*", ou seja, os programadores devem conhecer padrões de projetos de forma a facilitar e garantir a manutenibilidade dos sistemas no futuro. O item *designer* colocado na função refere-se justamente à utilização desses padrões de desenvolvimento nos sistemas.

A ideia dos *design patterns* (padrões de projeto ou padrões de desenho de software) é descrever soluções para problemas recorrentes encontrados durante o desenvolvimento de sistemas de software orientados a objetos. Cada padrão de projeto estabelece um nome, define o problema, a solução, quando é possível aplicar essa solução e as suas consequências. Esse conceito não é novo, surgindo a partir de trabalhos desenvolvidos pelo engenheiro civil Christopher Alexander, na década de 1970. Sua inspiração foi baseada, como não poderia deixar de ser, em problemas que tinha de resolver nos projetos de engenharia civil dos quais participava. Segundo ele, cada pattern descreve um problema que ocorre com frequência no ambiente. A partir daí ele descreve o núcleo da solução para esse problema.

Testador

Trata-se de um ou mais profissionais com conhecimentos e habilidades para testar o sistema. Cabe a eles realizar os chamados testes de regressão, que na família Crystal são considerados obrigatórios. Esses testes devem ser feitos por funcionalidade implementada e por entregas ao cliente, as quais, independente do tamanho, devem acontecer duas vezes por mês. Esse tipo de teste será melhor explicado no próximo tópico.

Redator

Desde o início do desenvolvimento, mesmo nas fases de elicitação, o redator deve estar disponível para documentar os fatos. Cabe a ele redigir atas, tabular os questionários e entrevistas para consultas futuras, além da construção de um manual do usuário, que deve conter todas as especificações de utilização dos sistemas. Deve ter livre acesso a todas as implementações, bem como participar de todas as reuniões relativas ao projeto de sistemas.

A partir da metodologia ORANGE da família Crystal, pode ser necessário criar outras funções nas equipes de projeto, dentre as quais podemos destacar:

a) **Gerente de projeto:** responsável por coordenar todo o projeto e liderar as equipes de coordenadores. Responsável também pelo relacionamento clientes x desenvolvedores, administração dos recursos envolvidos e divisão das partes a serem implementadas/desenvolvidas.

b) **Arquiteto de software:** profissional conhecedor da arquitetura de sistemas, que deve garantir que os diferentes módulos se integrem, bem como escolher as técnicas de desenvolvimento a serem aplicadas pelos programadores. Deve ter conhecimento da arquitetura dos bancos de dados e de redes de computadores.

c) **Facilitador técnico:** um profissional capaz de entender conceitualmente o negócio no qual a empresa atua, devendo conhecer os produtos e seus processos produtivos. O facilitador é a pessoa que regula o ritmo do grupo, oferece uma variedade de possíveis modos para abordar os problemas, mantém a equipe focada na metodologia escolhida e assegura que todos os participantes interajam de forma construtiva. Em resumo, o facilitador é um profissional que cuida da dinâmica do grupo que está realizando o trabalho.

d) **Especialista em usabilidade:** a usabilidade refere-se à facilidade com que os usuários conseguem utilizar a ferramenta desenvolvida. O especialista deve promover técnicas para garantir essa utilização, além de relacionar as medições de desempenho do sistema. A partir dessas medições pode-se analisar o tempo para criação do modelo mental do usuário em utilizar a ferramenta e tomar atitudes para melhorá-la, se necessário.

9.4 Utilização da metodologia Crystal Clear em um projeto de software

Visando proporcionar um melhor entendimento, abordaremos uma das metodologias da família, denominada *Crystal Clear* de desenvolvimento. Trata-se da mais simples das metodologias da família por envolver o mínimo de profissionais, sendo utilizada para projetos relativamente pequenos.

Abordaremos então um *case* hipotético com dados e nomes fictícios para exemplificar e apontar os documentos necessários que devem ser utilizados por essa metodologia.

Projeto web para a empresa Antonio Catini Ltda. - Comércio de Veículos

A empresa deseja criar um sistema para Internet para comercializar seus veículos, bem como proporcionar que usuários realizem seus próprios negócios pelo site. Para captar os requisitos, a metodologia sugere uma entrevista e um questionário em forma de *briefing* com perguntas-chave sobre o projeto a ser desenvolvido.O quadro a seguir apresenta um modelo de como isso pode ser registrado:

Briefing entrevista/questionário

Briefing para projeto web		
Projeto:	Comércio de Veículos Web - Catini WebCar	
Analista/gerente responsável	Maria Luísa	Flávio Macedo
Data: Maio/2011		
1. O que o site pretende vender? Relacionar os produtos e os serviços oferecidos.		
A princípio comercializar veículos novos e usados, além de permitir que os internautas cadastrem seu próprios veículos em uma espécie de rede social na qual podemos juntar quem procura e quem vende.		
2. Vantagens/desvantagens sobre os concorrentes.		
A principal vantagem é a possibilidade de adquirir veículos com preços de fábrica, pois comprando pelo site, podemos oferecer uma entrega mais direta, sem ter de transportar até o estacionamento da matriz para depois despachar ao cliente. Uma outra vantagem é filmar os veículos em funcionamento e dar detalhes deles por dentro e por fora para quem quiser baixar os vídeos. Finalmente, uma outra vantagem é permitir que os profissionais (mecânicos e funileiros) se cadastrem como avaliadores e ganhem para avaliar veículos para os clientes de sua região. Desta forma, o usuário solicita uma avaliação e o site permite a escolha do avaliador de confiança.		
3. Parceiros ou referências de sites (outras empresas do grupo).		
O site deverá conter uma lista de todos os fabricantes de veículos nacionais e importados e seus respectivos links de acesso, haja vista que trabalhamos com todas as marcas disponíveis em lojas e montadoras do Brasil. Obviamente não será necessário colocar montadoras que não vendem seus veículos diretamente ao consumidor no Brasil. Essa lista pode ser confirmada posteriormente devido ao número mundial de montadoras.		
4. Objetivos a serem alcançados.		
Apresentar um web site completo e com ele ampliar as vendas que hoje se restringem ao estado de São Paulo.		
5. Público-alvo.		
Todos os públicos, principalmente os interessados em comprar ou trocar seus autos.		
6. Conteúdo do site (relação inicial das páginas).		
Home, Veículos, Serviços, Contato, Área do Usuário (Rede Social), Mapa Conceitual de Veículos por Região do País.		
7. Imagem a ser transmitida aos usuários (tradição ou modernidade, layout clean ou popular, cores mais adequadas etc.).		
Modernidade com cores a serem definidas (azul e branco são as cores oficiais).		

8. Tecnologias ou linguagens desejadas, caso seja do conhecimento da empresa (exemplo: html, flash, php, jsp e outras).
Não se aplica. A empresa desconhece as tecnologias atuais.
9. Ferramentas de marketing utilizadas atualmente (relacionar outdoor, comerciais etc.).
Outdoor nas principais cidades, panfletagem em estacionamentos e comércios, anúncios nos jornais e revistas da capital paulista.
10. Restrições (exemplo: não utilizar a cor tal... ou não utilizar a fonte...).
Não se aplica.
11. Tempo para desenvolvimento sugerido pela empresa (urgência).
Um ano (negociável).
12. Contato principal da empresa para assuntos de processos e funcionamento.
Marcílio Catini - proprietário do grupo.
13. Observações não previstas neste questionário.
As possíveis dúvidas podem ser retiradas em novas reuniões previamente agendadas.

Documento de requisitos

Após o recebimento do *briefing* devidamente respondido, pode-se dar início ao documento de requisitos, que relaciona todos os requisitos do sistema, bem como detalhes de seu funcionamento. Pode ser necessário realizar mais reuniões para garantir que o documento esteja correto e validado. A metodologia Crystal não economiza em reuniões para definir claramente o que se pretende. Os requisitos podem ser relacionados em fichas individuais juntamente com documentos disponibilizados como modelos pela empresa.

Visões do usuário

A cada fase, ou a cada documento gerado, o usuário principal envolvido (*stakeholders*) deve validar o documento, dando um *feedback* sobre o assunto. O usuário pode solicitar modificações e alterações de possíveis equívocos em cada momento. Ele ainda pode convocar reuniões para aparar arestas e decidir os destinos da aplicação a qualquer momento. O gerente de projeto ou coordenador, neste caso, pode solicitar, por meio de documento lavrado pelo redator, assinaturas concordando com os requisitos solicitados, tanto do usuário quanto do patrocinador do projeto. A ideia é evitar mudanças radicais nos requisitos e poder apresentar um cronograma preciso de ações, bem como o esforço necessário. Desta forma, facilita-se a confecção de contratos e de garantias entre os desenvolvedores e a parte interessada, mesmo que simplesmente para um centro de custos da empresa.

Modelagem

Com o documento de requisitos em mãos, é então realizada a sua modelagem, utilizando, no mínimo, os seguintes diagramas da UML: diagrama de casos de uso, diagrama de atividades ou sequência e diagrama de classes. No caso de sistemas maiores, pode ser necessário o envolvimento de outros diagramas, mas no caso da *Crystal Clear*, os diagramas mencionados são suficientes. Observe que, mais uma vez, esses diagramas devem ser explicados ao usuário e validados por ele, deixando clara a correspondência dos diagramas aos requisitos.

Design de projeto

Considerando que a modelagem foi feita e os requisitos definidos, as interfaces já podem ser criadas. Um conjunto de interfaces deve ser criado com base nos questionamentos do *briefing*, bem como as solicitações de cores, fontes e estilos feitas pelo *stakeholders*. Essa ação resulta em um artefato denominado "design do projeto", que deve ser validado e conferido tela por tela junto com o *stakeholder*. Nessa fase podem ocorrer novas reuniões para decidir modificações nos estilos de interfaces, além de decisões quanto a avanços futuros para novas versões. No caso da empresa estudada, seria interessante apresentar várias possibilidades de cores e estilo para escolha.

Sequências de releases

Após os requisitos modelados e as telas escolhidas, o coordenador de projeto cria um modelo de prioridades junto com os *stakeholders* e divide as funcionalidades e recursos entre os desenvolvedores. Os programadores passam a saber as datas previstas de entregas das funcionalidades ao usuário e começam a codificação do sistema. Em cada iteração os resultados são coletados e analisados para fases subsequentes, sempre levando em conta que a *Crystal Clear* exige o mínimo de duas entregas de funcionalidades ao cliente por mês. Para o sistema em questão, podemos observar um exemplo na Tabela 9.2.

Sequência de desenvolvimento			
Projeto:	Comércio de Veículos Web - Catini WebCar		
ID	**Descrição**	**Data Início**	**Data Fim**
RF001	Cadastro de veículos novos e usados	01/jun/12	15/jun/12
RF002	Cadastro de usuários para login administrativo	01/jun/12	15/jun/12
...	...		
...	... (listar os requisitos por ordem de prioridade)		

Tabela 9.2 – Sequência de releases.

Código-fonte

A programação pode ser individual ou feita em pares, como na *Extreme Programming*, abordada no capítulo 10. No entanto, para pequenos projetos como este, é interessante ter no mínimo dois programadores experientes, capazes de entender a fundo elementos da linguagem escolhida. O código-fonte deve ser compartilhado entre eles e padronizado, de forma a permitir que alterações sejam feitas com facilidade.

Casos de testes

A metodologia *Crystal Clear* prevê uma bateria de testes, começando por testes funcionais, que nada mais são que aqueles realizados no sistema finalizado (ou em partes prontas), com o objetivo de testar suas funcionalidades no ambiente de trabalho. No caso de web sites, esses testes podem ser feitos na interface web, considerando sua interação com a base de dados. Os casos de teste devem ser descritos para efetivamente verificar as funcionalidades, ou seja, cada processo deve ser

confrontado com o requisito que o solicitou (os chamados testes de regressão). Esse tipo de teste, entre outros propósitos, também confere se funcionalidades novas interferem em funcionalidades já testadas. Pelo fato de a *Crystal Clear* fazer entregas de software constantes, esse tipo de teste é fundamental.

Manual do usuário

O redator técnico vai incluindo no manual do usuário as funcionalidades que já foram devidamente testadas. Além da descrição, é recomendável que inclua outras formas de ajudar o usuário, por meio de descrições passo a passo. O redator deve realizar esse processo a cada release, ou seja, a cada novo conjunto de funcionalidades que foram testadas e instaladas, a fim de manter sempre atualizado o manual do usuário. Também deve providenciar uma lista dos possíveis erros e suas soluções para ajudar o usuário em sua resolução, lembrando que essa lista refere-se a problemas no preenchimento e a erros provocados pelo usuário ou ausência de algum recurso fundamental.

Finalmente, é feito pelo coordenador de projeto um monitoramento de todas as releases, sendo atualizado todos os dias durante o período de desenvolvimento, que na *Crystal Clear* não deve ultrapassar quatro meses. Além de todas as funcionalidades, esse esquema de monitoramento deve apresentar as porcentagens concluídas em relação às pendentes. Para isso, pode ser feito um quadro como o proposto pela metodologia FDD, abordada no capítulo 6. Esse quadro deve destacar o andamento do projeto, tendo sempre em mente que a família de metodologias Crystal preza, principalmente, a existência de requisitos estáveis para projetar, designs estáveis para documentar e uma aplicação correta e testada que possa ser entregue com qualidade ao cliente.

9.5 Exercícios

1) Em casos de equipes maiores, quais são os profissionais a serem inseridos no planejamento e no desenvolvimento do projeto utilizando a família de metodologias Crystal?
2) Qual a função do redator na metodologia Crystal?
3) Nas filosofias da metodologia, explique a que se refere a afirmação: "considerar que as técnicas e as tecnologias mudam com o tempo".
4) Para que servem os "*design patterns*" e como são utilizados na metodologia Crystal?
5) Como podemos definir os testes de regressão obrigatórios nessa metodologia?

10

Extreme Programming (XP)

Objetivos

- *Apresentar as características dessa metodologia.*
- *Descrever as melhores práticas adotadas.*
- *Apresentar exemplos de utilização.*

10.1 Considerações iniciais

A eXtreme Programming (XP) é considerada uma metodologia ágil, pois se ajusta bem a pequenas ou médias equipes de desenvolvimento de software, em que os projetos são conduzidos com base em requisitos vagos que se modificam rapidamente. O XP possui algumas características bem marcantes, que são:

- *Feedback* constante.
- Abordagem incremental.
- Encoraja a comunicação entre as pessoas envolvidas.

Como ocorre com muitas das metodologias ágeis, a maioria das regras da XP causa polêmica à primeira vista, e muitas não fazem sentido se aplicadas isoladamente. O que a torna praticável é a sinergia de seu conjunto que sustenta o sucesso da XP, que encabeçou uma verdadeira revolução no conceito de desenvolvimento de software. A ideia básica é enfatizar o desenvolvimento rápido do projeto, visando garantir a satisfação do cliente, além de favorecer o cumprimento das estimativas. Outro fator estimulante ao uso dessa metodologia são as regras práticas e valores, que proporcionam um ambiente de desenvolvimento de software agradável.

A XP é uma metodologia de desenvolvimento de projetos de software com menos formalidade possível. Foi criada no final dos anos de 1990, mais precisamente em 1996, e apresentada por Kent Beck a OOPLSA (*Object-Oriented Programming, Systems, Languages and Applications*), em 2000, uma das maiores conferências de orientação a objetos reconhecida internacionalmente, que atualmente faz parte da *System*, *Programming*, *Languages and Applications: Software for Humanity* (SPLASH).

Essa metodologia, assim como outras abordadas neste estudo, tem como meta atender as necessidades dos clientes com qualidade e da forma mais simples possível. Por ser uma metodologia ágil, busca desenvolver os projetos rapidamente e entregá-los dentro de um prazo predefinido, mesmo que os requisitos mudem com frequência. A XP dá preferência ao desenvolvimento orientado a objetos e permite trabalhar com pequenas equipes de até 12 desenvolvedores (programadores), além de outras funções. Utiliza um modelo incremental, ou seja, na medida em que o software é utilizado, novas melhorias são implementadas. Desta forma, o cliente sempre tem um produto a ser utilizado e testado, possibilitando ao desenvolvedor conhecer e aprender com os processos da empresa ou projeto a ser construído, bem como o cliente avaliar se o que foi proposto foi desenvolvido exatamente.

Trata-se se uma metodologia flexível que pode ser adaptada a projetos distintos. Contudo, caso se observe a necessidade de um maior formalismo e retrabalho, o uso dessa metodologia deve ser reavaliado. Na Internet existem vários sites a respeito dela; um deles é mantido pelos seus colaboradores e criadores (www.extremeprogramming.org), que serviu de base e referência em nosso trabalho. Na Figura 10.1 podemos observar a estrutura geral da metodologia *Extreme Programming*, evidenciando suas características e destacando suas divisões: valores, práticas e equipe, as quais serão detalhadas no restante do capítulo.

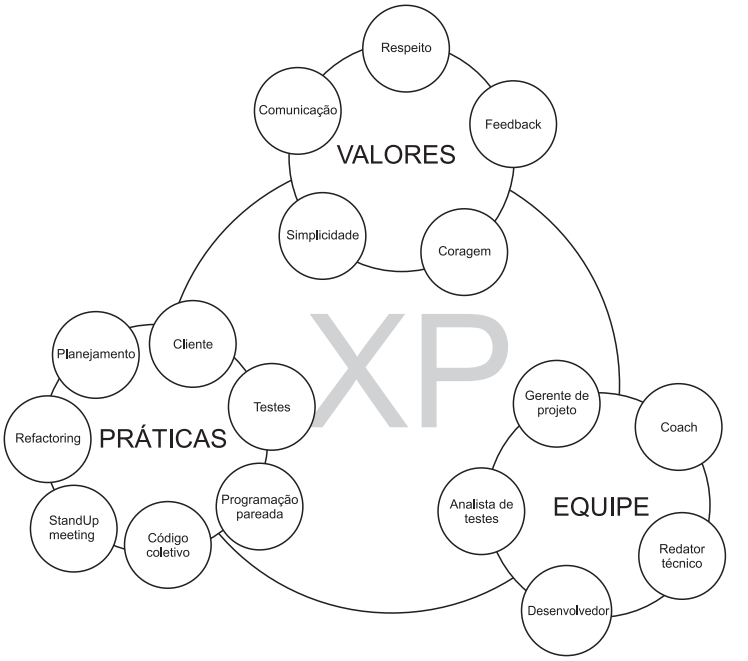


Figura 10.1 - Estrutura da metodologia.
Fonte: Arquivo pessoal do autor.

10.2 Valores da eXtreme Programming

Essa metodologia preza seus valores, ou seja, para caracterizar um projeto de software como tendo sido desenvolvido com XP, a equipe deve seguir à risca esse conjunto de valores sob a forma de diretrizes. As diretrizes devem ser encaradas como atitudes e prioridades de desenvolvimento, mesmo no caso de adaptações. A seguir, vamos descrever esses valores e suas vantagens, a fim de justificar tal utilização.

a) Comunicação

A metodologia XP propõe que os desenvolvedores devem se comunicar com os clientes de uma forma direta, de preferência face a face, ou seja, os membros da equipe de desenvolvimento devem tratar dos assuntos relativos ao projeto de forma que não haja mal-entendidos, especulações, dúvidas, entre outras coisas. Desta forma, as chances de o projeto ficar com pendências são reduzidas drasticamente, uma vez que os assuntos são tratados com muita agilidade e as dúvidas quanto ao funcionamento ou processos automatizados são sanadas continuamente.

Essa forma de tratar a comunicação permite a criação de um elo entre o cliente e o desenvolvedor, fazendo com que o cliente se sinta peça importante no desenvolvimento, enriquecendo os relacionamentos pessoais e garantindo fatores de confiança. Para que essa forma de comunicação funcione a contento, o gerente de projeto, estudado com mais detalhes posteriormente, deve atuar com a equipe estabelecendo contatos e mediando a relação cliente-desenvolvedor. Caso a equipe não consiga estabelecer esse contato, todo o projeto pode se comprometer. Portanto, membros que não sejam extrovertidos devem ser substituídos, pois não são adequados a equipes de XP. O estímulo a essa forma de comunicação também é importante, pois possibilita desenvolver o *feedback*, outro importante valor que será explicado a seguir.

b) Feedback

O feedback é sem dúvida um dos grades valores da XP, pois aproxima o desenvolvedor do cliente, que é o maior interessado no projeto de software. A ideia básica é que, à medida que o software é implementado, o cliente imediatamente o utiliza, podendo argumentar sobre essa utilização. A princípio pode levar algum tempo para esse contato ser estabelecido, pois o desenvolvedor vai criar as interfaces iniciais do software, mas a partir disso o cliente vai utilizar e dar sua opinião a respeito.

Convém observar que o cliente não será o único a opinar sobre o produto que está sendo desenvolvido, pois certamente existem soluções que o cliente pode desejar, e que não sejam possíveis em um ou outro contexto, ou podem provocar riscos às regras de negócio da empresa ou do projeto em questão. Esse tipo de *feedback* do desenvolvedor para o cliente não acontece com tanta frequência quanto ao inverso, ou seja, do cliente para o desenvolvedor, mas também é importante, pois busca atender e ao mesmo tempo estabelecer um acordo entre o desenvolvedor e o cliente. O *feedback* permite ainda que o cliente esteja sempre avaliando o produto e transmitindo seus problemas para os desenvolvedores, que a partir daí podem estabelecer prioridades no desenvolvimento. Outro ponto muito importante a respeito do *feedback* é que o cliente vai aprender com o sistema, criando com muito mais facilidade seu modelo mental de utilização e colaborando ainda para uma possível fase de adaptação e treinamento, que com o uso dessa metodologia não é necessário.

c) Simplicidade

Durante a concepção do projeto, os desenvolvedores devem ser orientados a não usarem recursos desnecessários aos objetivos do software. Para que a comunicação e o *feedback* funcionem corretamente, a equipe de desenvolvimento deve agir com simplicidade na resolução daquilo que o cliente deseja. Essa simplicidade não significa facilidade e muito menos privar o cliente de recursos importantes; a ideia é atender o cliente com o necessário para que os requisitos do sistema possam ser completamente atendidos.

Se a equipe agir dessa forma, tanto a comunicação quanto o *feedback* serão mais bem aproveitados, tendo em vista que a simplicidade gera melhor entendimento, pois rompe as barreiras impostas pela linguagem técnica. Existe uma "lei" da XP que diz: "Faça a coisa o mais simples possível, mas que funcione como o esperado". Como os requisitos mudam rapidamente e existe uma evolução natural dos sistemas para se ajustarem às novas necessidades das organizações, a XP procura evitar a utilização de elementos conhecidos como "perfumarias", ou seja, recursos extras que não contribuem com o processo e que provavelmente somente serão utilizados no futuro. O XP prega que devemos pensar no agora, naquilo que possa agregar e ser entregue ao cliente rapidamente.

d) Coragem

Esse valor da XP exige que os desenvolvedores, bem como a equipe de projeto, tomem atitudes muitas vezes complexas e/ou de alto risco. Na verdade, trata-se de premissas que contrariam os moldes tradicionais de desenvolvimento. A ideia é que alguns itens devem sempre ser pensados ao longo do desenvolvimento, outros antes mesmo de começar a implementar. Desta forma, ao receber a proposta para desenvolvimento e a lista dos requisitos, a equipe deve discutir e decidir como "atacar" o problema, sempre estabelecendo um contato próximo com o cliente. Após a análise dos requisitos, a equipe deve ter coragem para modificar, sugerir ou agir sobre o projeto a ser desenvolvido, mesmo que tais sugestões estejam fora dos interesses iniciais do cliente. A seguir apresentamos uma breve lista de itens que requerem coragem para serem discutidos:

- **Adoção de novos processos:** caso seja necessário para fins de automatização, a equipe pode sugerir a adoção de novos processos a empresa, bem como a modificação dos processos existentes.
- **Contrato de escopo variável:** se um projeto nunca foi automatizado, se não existe nem um software anterior, ou no caso de desconhecer os limites do novo sistema, a equipe deve conscientizar o cliente sobre essas possibilidades, pois podem inviabilizar o contrato atual.
- **Simplicidade do sistema:** como descrito anteriormente, a simplicidade na solução dos problemas deve sempre ser mantida, mesmo que isso cause algum desconforto aos desenvolvedores avançados.
- **Desenvolvimento incremental:** deixar claro à equipe que o projeto deve ser sempre incrementado e atualizado de acordo com as necessidades do cliente. Isso vale mesmo no caso de o projeto estar no final do desenvolvimento.
- **Ritmo sustentável:** determinar com a equipe um ritmo de desenvolvimento que não ultrapasse os limites de horário e dias preestabelecidos pela XP. Esse ritmo pode ser conseguido com base no conhecimento de projetos anteriores ou testando o rendimento individual de cada membro da equipe.

- **Assumir atrasos e problemas:** a equipe de desenvolvimento deve ter coragem de assumir problemas para cumprir o cronograma de entrega, de forma que o cliente esteja sempre bem informado, evitando desconforto e cobranças.
- **Expor o código-fonte:** todos os recursos, programas, linguagens e utilitários devem estar disponíveis a todos os membros da equipe, de forma que possam conhecer a lógica adotada, bem como técnicas de programação.
- ***Refactoring* contínuo:** os desenvolvedores devem investir parte do tempo melhorando seus códigos, haja vista que existe um planejamento em torno da entrega rápida de um produto.
- **Relação cliente x desenvolvedores:** os desenvolvedores dever ser colocados frente a frente com os clientes desde o início do projeto, a fim de estabelecer uma relação e apoiar a comunicação e o *feedback*.
- **Documentação:** a equipe deve documentar o software e/ou modelar apenas em caso de necessidade, salvo exigência do cliente.
- **Permitir que o cliente opine:** ter coragem para permitir que o cliente escolha as prioridades de desenvolvimento de acordo com suas necessidades.
- **Dividir em pares:** fazer com que os desenvolvedores trabalhem em pares, um codificando e outro revisando tudo o que está sendo feito.
- **Testes automatizados:** dedicar tempo da equipe para a realização de testes e automatizá-los para serem aproveitados em projetos futuros.
- **Responsabilidades:** ter a coragem de delegar responsabilidades a membros da equipe, permitindo assim formar uma equipe homogênea com diversas capacidades individuais.

e) Respeito

Este também é um valor muito importante da metodologia XP, pois estimula os desenvolvedores a respeitarem seus clientes e vice-versa, aceitando críticas sobre suas responsabilidades e dando ouvidos ao que outros membros da equipe sugerem.

10.3 Práticas da XP

Assim como os valores, existe na XP um conjunto de boas práticas a serem seguidas com o objetivo de garantir um ciclo de desenvolvimento fortemente dependente. Essas práticas são expressas por atitudes que devem ser seguidas pela equipe de desenvolvimento e somadas aos valores mencionados anteriormente. A seguir especificamos as práticas que o XP preconiza e as suas principais características.

10.3.1 Padrões de desenvolvimento

Antes do início de um projeto de software, assim que a equipe for formada, algumas regras devem ser definidas, ou seja, a equipe deve adotar um padrão de desenvolvimento que pode ser um *design*

pattern conhecido. A ideia é que, adotando um padrão, todos consigam entender mais rapidamente os códigos uns dos outros. Levando em conta que a comunicação é um dos valores da XP, essa prática possibilita a interação via código dos desenvolvedores. Além disso, cabe à equipe escolher a melhor linguagem de programação e as ferramentas que melhor se ajustam aos requisitos do projeto.

10.3.2 Design simples

Após a captação dos requisitos do projeto de software, a equipe separa-os em funcionalidades, também conhecidas como histórias. Em seguida, a implementação propriamente dita inicia, em que todas as interfaces e os códigos gerados devem ser feitos de forma simples e que atenda as necessidades do cliente. Levando em consideração que todo sistema pode sofrer alterações ao longo de seu desenvolvimento e que essas alterações, por sua vez, podem aumentar o custo e os prazos de um projeto, a XP tenta evitar esses imprevistos por meio de uma implementação básica. Atuando com um design simples, torna-se mais fácil conviver com alterações que o sistema pode sofrer.

Procura-se, desta forma, evitar programações desnecessárias que não têm utilidade para o cliente em um primeiro momento. Na medida em que o sistema vai se estabilizando e adquirindo consistência em relação aos seus requisitos mais importantes, a equipe pode estudar a possibilidade de acrescentar essas "perfumarias", caso necessário. A possibilidade de o sistema ser orientado a objetos facilita esse design, bem como as alterações solicitadas, mesmo que a linguagem de programação ou o framework seja atualizado.

10.3.3 Cliente sempre disponível ou presente

A metodologia XP trabalha de forma diferente das metodologias tradicionais (clássica), em que o cliente normalmente se adaptava ao software criado, e muitas vezes tinha de alterar seus processos para poder usá-lo. Como mencionamos anteriormente, a XP sugere que o cliente acompanhe o desenvolvimento e participe de todo o processo para que o sistema possa atendê-lo plenamente.

As histórias, documentos que descrevem as funcionalidades do sistema, devem ser escritas junto do cliente que as valida, continuamente, para que se possa estabelecer um ciclo de desenvolvimento. Na medida em que o software é implementado, o cliente tem a visão de como vai ficar, além de sugerir mudanças e participar da criação dos testes conceituais que serão os indicadores de uma boa codificação. Essa prática é importante, pois a não participação do cliente durante o processo de desenvolvimento pode levar a sérios riscos de projeto.

10.3.4 Jogo de planejamento

Quando a equipe de XP recebe os requisitos do cliente, divide todos em partes. No início se faz uma divisão por funcionalidades (as chamadas histórias). Em seguida o cliente recebe essa lista de histórias e prioriza cada uma delas. Essa classificação qualifica cada uma delas como sendo de prioridade alta, média ou baixa, para somente depois organizá-las em ordem numérica, que irá determinar a sequência de implementação. Com a lista de histórias podemos estimar o custo de cada uma delas e planejar sua execução, bem como as iterações necessárias.

Para representar um dia de programação, a XP utiliza uma unidade específica, chamada de "ponto", que orienta o desenvolvedor a se dedicar plenamente ao desenvolvimento, não executando outras tarefas que não objetivem atender a funcionalidade do projeto. No caso de uma funcionalidade demandar muito tempo, a XP sugere que essas funcionalidades sejam quebradas em partes de forma a atingir no máximo quatro dias de trabalho, ou quatro "pontos". Recomenda-se também que o cliente participe dessas divisões de tarefa e que esteja ciente de que o planejamento pode ser modificado ao longo do desenvolvimento do projeto.

A XP é uma metodologia incremental, em que o cliente pode utilizar o sistema constantemente com prazos de entrega de no máximo dois meses. Essas entregas do sistema para uso são chamadas de *release*, e permitem ao cliente beneficiar-se das funcionalidades do sistema em um curto prazo de desenvolvimento. Recomenda-se que as *releases* sejam planejadas no início do projeto, facilitando o *feedback*, bem como a possibilidade de aprendizado com o sistema. Observe que a *release* contém as iterações que, por sua vez, contêm as histórias a serem implementadas.

10.3.5 Stand up meeting

Trata-se de uma reunião rápida que normalmente é feita pela manhã, com aproximadamente 20 minutos de duração. Como na metodologia SCRUM, também abordada neste livro, essa reunião é realizada em pé, com o objetivo de ser objetiva, estimulando os integrantes a trocar experiências e dificuldades encontradas para implementar cada funcionalidade. A ideia dessa reunião também é informar o que foi feito no dia anterior e o que será feito no dia. É desta forma que são definidas as histórias a serem implementadas, bem como os responsáveis por elas.

10.3.6 Programação em pares

O conceito de *pair programming* também está presente em outras metodologias. Estudos apontam que programar em dupla pode render mais do que programar sozinho. Isso ocorre porque, na programação pareada, enquanto um desenvolvedor digita o código, o outro simultaneamente vai revisando, de forma a evitar erros de programação ou sugerir melhores estratégias de implementação. O desenvolvedor que digita é o condutor e seu parceiro o navegador, ambos com o objetivo de resolver o mesmo problema, em conjunto. A XP exige que todo código implementado seja feito em dupla e no mesmo computador.

Outra razão para se utilizar a programação em par é a constante troca de experiências da dupla, possibilitando que os problemas sejam divididos em dois. Também percebemos que, ao longo do projeto, duas pessoas acabam conhecendo o mesmo código gerado, e na ausência de uma delas, a outra pode suprir uma eventual demanda de manutenção desse código. Caso seja necessário incluir um membro na equipe, fica muito mais fácil a integração, assim como o compartilhamento de conhecimentos e treinamento.

10.3.7 Refactoring

Durante os testes das funcionalidades implementadas, o cliente normalmente dá um *feedback* sobre o resultado do trabalho. Baseado nas impressões do cliente, o desenvolvedor pode ter de

modificar alguma coisa no código para atendê-lo. Durante as revisões do código para modificação, caso o desenvolvedor se depare com um código malfeito, ou incorreto, cabe a ele melhorá-lo, deixando-o o mais legível possível e sem erros. Essa prática de correção e melhoria é chamada na XP de *refactoring*. O objetivo não é sair refazendo tudo, e sim melhorar o que já foi feito, tendo como base o conhecimento ou casos de melhores performances. Além de estar apoiado pelos testes automatizados e pelo *feedback*, o *refactoring* reforça a prática do código coletivo e da programação pareada.

10.3.8 Desenvolvimento guiado por testes

Toda e qualquer funcionalidade deve ser acompanhada de um teste automatizado. Isso permite verificar instantaneamente se a implementação atende ao requisito e ao cliente. Os testes podem ser de unidade, que verificam se os resultados gerados estão corretos, e os testes de aceitação, aprovados pelo cliente, atendem o que foi solicitado.

10.3.9 Código coletivo

Nas metodologias tradicionais, o projeto é dividido em partes e cada programador ou equipe se responsabiliza por entregá-las funcionando. Desta forma, normalmente apenas quem implementou consegue entender a lógica utilizada, tornando-se inviável ou muito onerosa a manutenção desse sistema. É possível que, em alguns casos, o projeto seja até interrompido por um problema específico, principalmente se quem projetou essa parte não estiver disponível para corrigir. É por essa razão que a metodologia XP sugere que todos os desenvolvedores tenham acesso a todas as partes do código, tendo a liberdade de alterar qualquer parte do código, sempre que necessário. Isso faz com que não exista um responsável por partes do código, e sim haja uma responsabilidade coletiva pela implementação.

10.3.10 Metáfora

Para facilitar a comunicação entre a equipe e os clientes, a XP adotou um artifício para criar analogias com os processos utilizados. Desta forma, o cliente não se sente constrangido por não entender os termos técnicos utilizados pelos desenvolvedores. A utilização de metáforas também colabora com a comunicação e o *feedback*, os quais, como vimos, são valores indispensáveis da XP ligados ao relacionamento com o cliente.

10.3.11 Ritmo sustentável

Durante a concepção de um projeto de software, uma das partes mais difíceis é precisar o prazo de entrega. Em metodologias tradicionais, a equipe de projeto é submetida a trabalhar muitas vezes em horários alternativos para compensar a falta de planejamento ou problemas de percurso, e em alguns casos os desenvolvedores passam os finais de semana trabalhando em soluções que deveriam ter sido entregues no prazo.

Em um primeiro momento essa dedicação extra faz com que o desenvolvimento do projeto cresça em rendimento, mas após algum tempo percebe-se que não adianta forçar o desenvolvedor a terminar certa tarefa rapidamente, pois o cansaço físico e mental acabará prejudicando o andamento e provocando situações de retrabalho, ou seja, o que foi desenvolvido nesses horários muitas vezes precisa ser refeito.

A XP proíbe que a equipe de projeto atue desta forma, ou seja, proíbe que a equipe trabalhe fora de seu horário normal. O ritmo sustentável sugerido pela XP não deve ultrapassar a carga horária de 40 (quarenta) horas semanais. Isso evita que o desenvolvedor perca sua concentração em função do cansaço e cause falhas na implementação.

10.3.12 Integração contínua

Trata-se da possibilidade de integrar o sistema várias vezes ao dia para que a equipe de desenvolvimento e o cliente estejam sempre com o software atualizado, melhorando assim o *feedback*. A XP sugere que a equipe de desenvolvedores sempre teste a integração ou o impacto que uma nova funcionalidade pode causar no software, considerando sua integração com o sistema como um todo. Obviamente essa integração é feita mediante um backup da versão imediatamente anterior para que o cliente não seja prejudicado por uma falha geral no software.

Algumas ferramentas *case* permitem "impactar" a aplicação para verificar seu funcionamento, ou seja, testar o que a nova ou as novas funcionalidades vão trazer de problemas ao projeto e permitir seu cancelamento imediato sem que o cliente (usuário) perceba. A XP pretende com essa integração evitar o que pode acontecer nas metodologias mais tradicionais, em que o desenvolvedor só se preocupa com o que ele implementou e não sabe o que os outros estão fazendo em paralelo. A integração contínua proposta pela XP faz com que toda a equipe saiba das modificações ocorridas diariamente.

10.3.13 Releases curtos

Release corresponde a pequenas partes do software, o qual é dividido de forma que essas partes possam ser implementadas em um curto espaço de tempo, atendendo e agregando valor ao cliente. Essa divisão é feita assim que os requisitos são captados, sendo realizada em forma de funcionalidades, e o cliente é quem vai decidir sobre as prioridades de desenvolvimento.

Nas metodologias tradicionais, o cliente muitas vezes começa a utilizar o software solicitado no final do desenvolvimento completo da aplicação. Neste caso, o cliente muitas vezes já pagou pelo desenvolvimento e não teve a oportunidade de utilizar nenhuma funcionalidade ao longo do seu desenvolvimento. A XP exige que partes do software sejam liberadas para utilização imediatamente após serem implementadas, fazendo jus ao investimento realizado até o momento e permitindo uma reflexão antecipada sobre sua usabilidade.

10.4 Equipe XP

Para que projetos de software sejam feitos com qualidade, a XP propõe uma formação de equipe com alguns papéis bem definidos,resumidos a seguir.

10.4.1 Gerente de projeto

Trata-se de um profissional que deve "vestir a camisa" da metodologia XP, ou seja, deve acreditar e seguir seus valores e práticas, bem como cobrá-los dos outros membros de sua equipe. Cabe ao gerente ser o maior elo de contato com o cliente, apresentando o cenário da metodologia e explicando como tudo funciona. Esse relacionamento próximo com o cliente estimula a participação ativa do cliente nas atividades de desenvolvimento. Negociações de prazos e custos também devem ser realizadas por esse profissional, que pode ainda organizar todo o procedimento de início e dar ritmo ao desenvolvimento do projeto de software.

10.4.2 Coach

Profissional com maior nível de conhecimento sobre a XP, cabendo a ele monitorar a equipe para que siga os valores e as práticas da metodologia. A equipe verá nesse profissional uma espécie de mentor para questões relacionadas à metodologia XP, e todas as dúvidas e respostas devem ser sanadas por ele. É importante que esteja sempre com a equipe em todas as fases de implantação.

10.4.3 Desenvolvedor

Nessa metodologia não existe diferença entre analista e programador. O chamado "desenvolvedor" deve ter conhecimentos de análise e programação. Portanto, durante a concepção do projeto, esse profissional pode exercer diversos papéis, como o de projetista, designer, analista, programador ou mesmo analista de banco de dados.

Como a programação na XP é pareada, basta que um dos desenvolvedores tenha experiência para que ela seja multiplicada para seu par, e assim sucessivamente ao longo do tempo de desenvolvimento desse projeto e de outros, contribuindo para que a equipe se torne uniformemente hábil e preparada para qualquer projeto. Vale a pena salientar que, em caso de os pares não terem experiência em programação e análise, a parceria deve ser revista e pelo menos uma das pessoas deve ser devidamente treinada.

10.4.4 Analista de testes

A qualidade dos sistemas projetados com uso da metodologia XP é medida por um conjunto de casos de testes, os quais devem ser inscritos pelo analista de testes juntamente com o cliente e aplicados no final de cada iteração. O analista de testes não deve acumular função com os desenvolvedores, para evitar uma visão tendenciosa do que foi desenvolvido. A qualidade pode ser atestada validando o produto ou a fase em que se encontra com o cliente.

10.4.5 Redator técnico

Mesmo sendo considerada uma metodologia ágil e sem burocracias, a XP propõe o uso de um mínimo de documentação, que existe de forma simples e é produzida paralelamente ao desenvolvimento. O redator técnico é responsável pelas documentações produzidas e deve acompanhar

todo o desenvolvimento, dia a dia. Os tipos de documentação do sistema exigidos pela XP serão exemplificados posteriormente. Existem inúmeras ferramentas *cases* para gerar a documentação adequada, além de ferramentas de engenharia reversa que possibilitam o entendimento da aplicação anterior (se houver).

10.5 Utilização da metodologia Extreme Programming

Para apresentarmos uma ideia prática do uso da metodologia XP, vamos tomar como exemplo um sistema para pet shop, cujos requisitos podem ser resumidos pelas seguintes características:

- A *pet shop* "Santo Expedito" necessita de um sistema que deve atuar com comunicação em rede, contemplando oito hosts divididos em três departamentos (caixa, consultório e estoque). Portanto, o sistema pode ser modular, mas deve acessar a mesma base de dados.
- O sistema deve controlar uma carteira de 2.500 clientes, sendo 435 considerados especiais, cujo tratamento é diferenciado. Atualmente a *pet shop* controla esses clientes por meio de uma ficha cadastral com preenchimento manual, que conta com os seguintes dados: nome, endereço, número, complemento, bairro, cidade, UF, CEP, telefone residencial, telefone celular, e-mail, RG, CPF, foto e data de nascimento. Além disso, em cada ficha constam também os dados de seus animais: nome, raça, cor predominante, idade, se é alérgico, horário em que costuma comer, marca da ração, número de registro e foto.
- Existe atualmente um número fixo de veterinários que trabalham na *pet shop*, mas há possibilidade de outros veterinários atenderem emergências nas dependências da clínica que a *pet shop* "Santo Expedito" disponibiliza. Dos veterinários é necessário saber CRMV, nome, endereço, número, bairro, telefone para emergências, e-mail e horários de atendimento. Os serviços realizados por veterinários que não são fixos geram contas a receber, ou seja, o veterinário paga uma taxa de utilização das instalações da *pet shop*.
- O sistema deve ter um módulo de agendamento de consultas, em que o cliente pode agendar suas consultas com os veterinários fixos durante o horário comercial em todos os dias da semana, exceto domingos e feriados. Para o agendamento, o cliente deve informar o nome do cadastro e o nome do animal, além de escolher entre os dias e horários disponíveis. Após o atendimento, o agendamento deve ser convertido em consulta e gerar uma conta a receber, que poderá ser quitada imediatamente no caixa ou posteriormente apenas para clientes especiais. A *pet shop* "Santo Expedito", além do pagamento em dinheiro, aceita todos os cartões de crédito e débito, inclusive cheques de todos os bancos nacionais.
- A *pet shop* conta também com um grande número de produtos para venda, que gira em torno de 1.500 tipos diferentes. Os produtos são controlados atualmente por uma lista contendo código de barras, descrição, unidade de medida, peso, quantidade em estoque, valor de custo unitário, valor de venda unitário, cor, tipo, fabricante e quantidade mínima em estoque.
- Os produtos vendidos podem ser somados às consultas realizadas e ambos pagos no caixa, que solicita a forma de pagamento do cliente. Em seguida o caixa pergunta se o cliente deseja os créditos da Nota Fiscal Paulista e, em caso positivo, imprime o cupom fiscal da venda após o pagamento. O sistema deve debitar a quantidade do produto vendido no estoque,

que em alguns casos deve informar a necessidade de novas aquisições baseadas no estoque mínimo do produto. No caso de veterinários externos, as consultas devem gerar uma conta a pagar para o profissional que atendeu a consulta.

- As compras de produtos são realizadas em fornecedores cadastrados que disponibilizam seus dados cadastrais, como CNPJ, razão social, endereço, bairro, cidade, telefone, e-mail, contato. O sistema deve emitir uma listagem de produtos a serem comprados (baseada no estoque mínimo); quando realizada a compra, o sistema deve permitir a digitação da nota fiscal do fornecedor, bem como o acréscimo do produto ao estoque.
- Diariamente o sistema deve possibilitar o fechamento do caixa, permitir a conferência e seu possível acerto, mediante as sobras e o troco. Esse fechamento deve ser arquivado para o fechamento mensal e disponibilizado em forma de relatório que pode ser emitido a qualquer momento. O sistema deve possibilitar a emissão de consultas e relatórios gerenciais, além de relatórios cadastrais.
- Sob o ponto de vista da segurança, cada usuário (funcionário) deve ser previamente cadastrado para utilizar o sistema, fornecendo dados como nome, endereço, bairro, cidade, telefone residencial, telefone celular, e-mail, RG, CPF, data de nascimento e foto. O sistema deve solicitar uma senha de acesso a esses usuários e possibilitar um controle de permissões por níveis de responsabilidade. Todas as ações devem ser monitoradas e, se necessário, possibilitar a emissão de relatório das transações realizadas por cada usuário (logs de acesso).
- Finalmente, o sistema deve ter uma interface amigável, responder rapidamente e atuar sob o sistema operacional Windows Seven. A *pet shop* pode dedicar um computador específico para ser o servidor da rede, bem como uma sala climatizada para abrigá-lo.

10.5.1 Uso da metodologia XP

Primeiro passo

Os requisitos devem ser separados e discutidos com a equipe em uma reunião conduzida pelo gerente de projeto. Em um primeiro momento, é preciso discutir as ferramentas que serão utilizadas, a linguagem de programação e o banco de dados mais adequados, eventuais protocolos de comunicação, os prazos e os custos iniciais para o desenvolvimento. O redator técnico deve anotar os pontos discutidos e todas as decisões tomadas em forma de uma ata, que pode ser observada na Figura 10.2. Em um segundo momento, a equipe pondera sobre o uso e a viabilidade das ferramentas sugeridas, utilizando testes, se for o caso. Caso não entrem em um acordo imediato, uma nova reunião deve ser agendada para tomar as devidas decisões.

Projeto:	Pet Shop "Santo Expedito" - Proprietária: Veterinária Dra. Beatriz Carrara		
Data início:	Janeiro 2012	**Prazo entrega:**	6 (seis) meses
Linguagem:	C# - Visual Studio 2010	**Banco de dados:**	SQL - Server

ATA DE REUNIÃO - INÍCIO DE PROJETO
Descrever tudo o que foi discutido.
Equipe

Figura 10.2 - Modelo de ATA para início de projeto XP. Fonte: Arquivo pessoal do autor.

Segundo passo

Com o documento de requisitos em mãos, uma nova reunião deve ser agendada, contando desta vez com a participação do cliente. Os requisitos devem ser expostos ao cliente e, em seguida, solicitar a ele que priorize as funcionalidades mais urgentes, para que sejam produzidas primeiro. Esse documento de requisitos pode ser feito utilizando algum template ou o modelo sugerido na Figura 10.3. Observe que a prioridade pode ser informada por número ou por qualidade: 1 - prioridade baixa, 2 - média ou 3 - alta.

RF01	O software deve permitir que o usuário cadastre os clientes da pet shop.			
Descrição	É preciso implementar um formulário no qual o usuário deve digitar os dados do cliente, como ID, nome, endereço, número, complemento, bairro, cidade, UF, CEP, telefone residencial, telefone celular, e-mail, RG, CPF, foto e data de nascimento. Em seguida o usuário aciona um botão para salvar os dados digitados.			
Restrições	O ID deve ser automático, o UF deve ser previamente preenchido com todos os estados e deve ser usada máscara nos campos RG, CPF, CEP e data de nascimento. O sistema não pode aceitar fotos maiores que 2 MB e deve validar o CPF.			
Prioridade	(x) Alta	( ) Média	( ) Baixa	
Data Início	**Modificado**	**Modificado**	**Modificado**	**Finalizado**
20/01/2012				
Diagramas	**Classes**	**Caso de Uso**	**Atividade**	**Sequência**
	DC001	DCU002	DAT001	
Desenvolvedores	João Batista e Maria Luísa			

Figura 10.3 - Modelo de ficha de requisitos. Fonte: Arquivo pessoal do autor.

Após o cliente especificar as prioridades, o gerente de projeto pode preparar as *releases* a serem implementadas de acordo com as funcionalidades (histórias), de forma a trazer retorno ao cliente, ou seja, permitir que o cliente utilize o que já foi feito, enquanto novas funcionalidades são desenvolvidas. Durante essa reunião, o gerente deve enfatizar para o cliente a necessidade de sua participação na criação do projeto, informando-o também sobre a participação dos desenvolvedores em seu ambiente de trabalho, além de dados sobre os prazos e custos de todo o projeto. Nesse momento, cabe ao redator técnico anotar os resultados do encontro, entregando posteriormente uma cópia ao cliente para que ele avalie seu conteúdo e utilize como base de consulta e para formalizar acordos sobre os pagamentos.

O valor do projeto pode ser calculado por pontos de função, já que *a Extreme Programming* não sugere como calcular, ou seja, por funcionalidade definida na captação dos requisitos. Os pontos de função são calculados a partir de vários itens, como, por exemplo, as fronteiras da aplicação, o acesso a dados, as transações e a complexidade que normalmente pode ser baixa, média ou alta.

Como sugestão, durante a realização de vários projetos, observe quanto tempo leva o desenvolvimento de determinados módulos do sistema e calcule parte do custo por meio das horas pagas aos desenvolvedores. Também não deixe de observar o grau de dificuldade do projeto. Existem várias formas de negociar um projeto de software. Algumas fábricas de software, por exemplo, optam por alugar seus sistemas, outras vendem os arquivos "fontes" e cobram as alterações e manutenções realizadas. Outras empresas ainda optam por desenvolver sob medida e deixar o resto a cargo do cliente, entre outros modelos de negócio. A XP não tem objeções sobre nenhuma dessas possibilidades.

Terceiro passo

Após o cliente ter aprovado a proposta de início do desenvolvimento, as histórias são organizadas em ordem de prioridade e, nesse momento, acontece o primeiro *Standup Meeting*. O gerente conduz essa rápida reunião e informa a todos como se dará o relacionamento com os funcionários da empresa, neste caso a pet shop "Santo Expedito". Portanto, essa reunião deve acontecer na empresa do cliente, pela manhã, em que rapidamente o cliente apresenta os funcionários, os equipamentos e as dependências a serem utilizadas. Os desenvolvedores devem trazer seus próprios equipamentos para serem utilizados durante o desenvolvimento, tomando o cuidado de não instalar nas máquinas do cliente softwares que não tenham relação com o produto final (sistema). O *coach* entra em cena e dá as primeiras dicas de como interagir com os funcionários, reafirmando os valores propostos pela XP.

Os desenvolvedores devem, por sua vez, criar os diagramas da UML necessários para o desenvolvimento, que no caso da XP devem ser minimamente o diagrama de casos de uso, o diagrama de classes e o diagrama de atividades ou sequência. Esses diagramas servem apenas para que os desenvolvedores e o cliente entendam o funcionamento de todo o sistema de forma gráfica, colaborando com as metáforas utilizadas para a explicação do sistema. O sistema então começa a ser construído e o analista de testes começa a criar os casos de testes junto com o cliente, fazendo com que o *feedback* comece a dar resultado. A equipe deve seguir todas as práticas até o final do projeto, observando sempre os valores da XP. Em paralelo o redator técnico vai documentando o sistema, considerando somente o que for realmente necessário, interagindo com o cliente sobre o tipo de documentação desejado.

Quarto passo

Finalmente o gerente organiza uma nova reunião com o cliente e expõe todo o desenvolvimento, solicitando que o cliente aponte modificações ou inclusões de requisitos que não foram pensadas no início do projeto. O redator técnico novamente faz uma ata dessa reunião para que sejam discutidos novos prazos e custos. Caso não tenham nada a acrescentar ao software, é estipulado um prazo de acompanhamento da sua utilização. No final do desenvolvimento devemos ter o cuidado de elaborar um contrato contendo as garantias e formas de manutenção do projeto.

10.6 Casos de Sucesso

Existem inúmeros casos de sucesso no uso da metodologia *Extreme Programming*. A seguir citamos alguns deles:

CQG

O primeiro deles publicado na Java Developer Journal, disponível no endereço da web: http://www2.sys-con.com/itsg/virtualcd/java/archives/0511/Westra/index.html, relata um caso da empresa CQG Inc. A CQG recebe dados das bolsas de valores ao redor do mundo e distribui aos empresários de mais de 50 países. Tem uma reputação a zelar e procura produzir os dados mais confiáveis para seus colaboradores e para a indústria, e ela deve estar disponível 24 horas por dia, 7 dias por semana, ou seja, com zero de downtime. A empresa está sediada em Denver, com filiais em Chicago, Nova York, Londres, Paris, Frankfurt, Moscou, Milão, Zurique e Tóquio. Centros CQG de desenvolvimento baseiam-se em Denver, Boulder, Moscou, Dallas e Londres. A empresa utilizou a metodologia XP como base para um projeto feito com a linguagem Java e seus componentes.

Primavera

Trata-se de uma empresa de software com mais de 20 anos de existência que desenvolve projetos e soluções de software. Desenvolve projetos de alta complexidade, atendendo inúmeros clientes, e possui um grupo de desenvolvedores das mais diversas linguagens e tecnologias. Começou a utilizar a *Extreme Programming* em seus projetos com o objetivo de levantar a autoestima de seus desenvolvedores que sempre trabalhavam exaustivamente em projetos, não vendo os resultados. Muitas vezes trabalhavam fora de horário e ficavam descontentes com isso. A XP resolveu esses problemas e o resultado da experiência é uma base de clientes altamente satisfeitos, e a equipe altamente motivada com um ambiente de desenvolvimento energético.

Estudo de caso disponível em:

http://www.objectmentor.com/resources/articles/Primavera.pdf.

Escrow.com

A empresa Escrow adotou a *Extreme Programming* após inúmeros problemas em vários projetos, em que o custo e o prazo nunca eram precisos. Atua em projetos de e-commerce nos quais os requisitos mudam com muita frequência e necessitam de muita qualidade e velocidade para realizar transações on-line. Pela coleta de um pequeno conjunto de métricas simples, a Escrow foi capaz de medir o sucesso relativo do esforço de adoção da metodologia XP.

Mantém alguns *cases* no endereço:

http://www.agilelogic.com/files/ExtremeProgrammingPerspectivesCh30.pdf

10.7 Exercícios

1) Em que consiste a programação em pares na metodologia *Extreme Programming*?
2) Na metodologia XP, qual o papel do redator?
3) Entre os valores da metodologia XP, como podemos descrever o termo *feedback*?
4) Na metodologia *Extreme Programming*, como podemos definir o fato de o código ser coletivo?
5) Na metodologia XP, o que é *refactoring*?

11

SCRUM

Objetivos

- *Apresentar as características dessa metodologia.*
- *Descrever as melhores práticas adotadas.*
- *Apresentar exemplos de utilização.*

11.1 Um pouco de história

A metodologia SCRUM, bem como outras metodologias consideradas ágeis, foi fortemente influenciada por boas práticas adotadas pela indústria japonesa, em especial por princípios da manufatura enxuta implementados pelas companhias Honda e Toyota. Takeuchi e Nonaka escreveram um famoso artigo em 1986 intitulado "*The new product development game*", pela *Harvard Business Review*. Nesse artigo, eles descrevem que projetos que usam equipes pequenas e multidisciplinares produzem melhores resultados.

A denominação dessa metodologia surgiu da associação dessas equipes de projeto altamente eficazes com uma típica formação do evento esportivo rugby denominada scrum. No rugby, essa formação é utilizada após determinado incidente ou quando a bola sai de campo, ou seja, é utilizada para reiniciar o jogo, reunindo todos os jogadores. O uso dessa terminologia pareceu adequado porque no rugby cada time age em conjunto, como uma unidade integrada, cada membro desempenha um papel específico e todos se ajudam em busca de um benefício comum.

Posteriormente, Jeff Sutherland, então vice-presidente de engenharia da Easel, percebeu que seu time de software precisava de uma metodologia de desenvolvimento mais adequada para proporcionar um rápido desenvolvimento de aplicações. Seu objetivo era apresentar ao CEO da Easel

versões de software funcionando ao final de curtas interações, em vez de diagramas de Gantt. Então, Jeff Sutherland, contando com a colaboração de John Scumniolates e Jeff Mackenna, concebeu, documentou e implementou o SCRUM, incorporando os estilos de gerenciamento observados por Takeuchi e Nonaka.

Durante mais ou menos esse mesmo período, Ken Schwaber estava procurando ativamente soluções para ajudar sua empresa, a *Advanced Development Methods Inc.* (ADM), a melhorar a produtividade do seu time quando desenvolvia projetos de software. Após a análise de vários casos de sucesso, percebeu que seus processos de desenvolvimento eram similares pelo fato de todos utilizarem processos empíricos, requerendo constante inspeção e atualização.

A pedido da *Object Management Group* (OMG), Jeff Sutherland e Ken Schwaber, contando também com a colaboração de Mike Beedle, trabalharam juntos para formalizar e definir o que tinham aprendido, criando assim a metodologia SCRUM como hoje a conhecemos. A partir de então, a metodologia SCRUM tem sido utilizada por todo o mundo, nos mais variados domínios de aplicação. Esse sucesso tem sido atribuído, principalmente, ao fato de os métodos tradicionais de desenvolvimento terem o foco na geração de documentação sobre o projeto e no cumprimento rígido de processos. Já os métodos ágeis, como o SCRUM, concentram atenções no produto final e nas interações dos indivíduos, conforme mencionamos anteriormente.

11.2 Características do SCRUM

A metodologia SCRUM segue os princípios do manifesto ágil, apresentado no capítulo 5, lembrando que Jeff Sutherland e Ken Schwaber também são seus signatários. Segundo Schwaber, o SCRUM baseia-se em seis características: flexibilidade dos resultados, flexibilidade dos prazos, times pequenos, revisões frequentes, colaboração e orientação a objetos. Como todos sabem, não existe uma solução mágica para problemas complexos, mas existe um consenso de que podemos usar o SCRUM para:

- Desenvolvimentos complexos em que os requisitos mudam rapidamente e constantemente;
- Gerenciar e controlar o desenvolvimento do trabalho;
- Tornar a equipe autogerenciável e funcional;
- Implementar o conceito iterativo e incremental no desenvolvimento de software e/ou produtos;
- Identificar causas de problemas e remover impedimentos;
- Valorizar os indivíduos.

É importante ressaltar que o SCRUM não se aplica exclusivamente ao desenvolvimento de software, uma vez que sua característica iterativa e incremental permite utilizá-lo no desenvolvimento de qualquer produto ou no gerenciamento de qualquer trabalho. Relembrando, o desenvolvimento iterativo e incremental é uma estratégia de planejamento em que o produto é construído em partes (iterações). Ao término de cada parte, é feito um novo incremento (parte funcional de um produto ou software) até que o desenvolvimento seja completado. Podemos dizer que o SCRUM é apoiado em quatro fundamentos, explorados mais adiante e ilustrados na Figura 11.1.

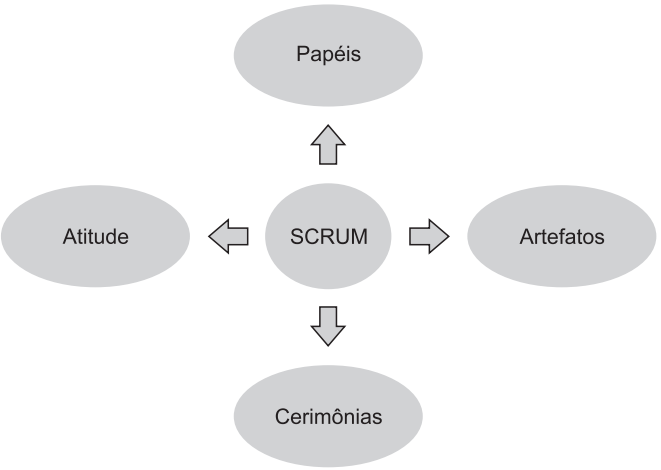


Figura 11.1 – Fundamentos básicos do SCRUM. Fonte: Arquivo pessoal do autor.

Além das seis características principais citadas anteriormente, o SCRUM também é conhecido pelo fato de estruturar seu funcionamento por ciclos, chamados de *sprints* (corrida de velocidade em inglês), que representam iterações de trabalho com duração variável, geralmente de duas a quatro semanas. Também estabelece um conjunto de práticas e regras que devem ser cumpridas pela equipe, em seus respectivos papéis, que no SCRUM se divide em três papéis principais: o *Product Owner*, o *SCRUM Master* e o *Team* ou Equipe SCRUM. Muitos acreditam ser prudente também incluir a figura do cliente entre os papéis, uma vez que de fato ele tem uma participação importante no processo de desenvolvimento.

Além dos papéis citados, podemos identificar quatro cerimônias existentes nessa metodologia, que são conduzidas periodicamente ao longo do desenvolvimento: o *Daily Meeting* (também chamado de *Daily Scrum*), a *Sprint Review*, o *Sprint Planning Meeting* e a *Sprint Retrospective*. Finalmente, como resultado dessas cerimônias são gerados três artefatos básicos, que são documentos denominados *Product Backlog*, *Sprint Backlog* e o *Burndown Chart*. Ao longo deste capítulo vamos esclarecer melhor os papéis, as cerimônias e os artefatos citados.

Outra característica importante do SCRUM é ter como princípio básico o reconhecimento de que os clientes podem mudar de ideia ao longo do desenvolvimento do projeto, simplesmente porque eles querem ou precisam. Por essa razão é que tais desafios imprevisíveis não podem ser tratados de uma maneira tradicional. É por isso que o SCRUM adota uma abordagem empírica, aceitando que o problema do cliente não pode ser totalmente entendido ou definido.

Também é curioso observar que o SCRUM pode ser implementado nas organizações por meio de uma ampla gama de recursos ou ferramentas. Observamos que muitas organizações utilizam ferramentas de software, como planilhas eletrônicas, para definir e manter alguns artefatos propostos pelo SCRUM. Por outro lado, existem organizações que utilizam o SCRUM sem o uso de qualquer ferramenta de software, mantendo seus artefatos em documentos em papel, quadros ou notas.

11.3 Ciclo de desenvolvimento

Uma visão geral do ciclo de desenvolvimento da metodologia SCRUM pode ser encontrada na Figura 11.2. Essa figura ilustra o fato de que, no início de cada projeto, clientes e desenvolvedores se reúnem com o objetivo de definir o *Backlog*[1] do produto (que representa a lista de requisitos). Nesse momento também são estimados os custos do projeto e é feita uma análise dos riscos, bem como as ferramentas de trabalho e os integrantes da equipe são escolhidos e são definidas as datas para entrega de resultados a partir de priorizações sinalizadas pelo cliente. Esse período também é utilizado para escolher o *SCRUM Master*, eleito entre a equipe alocada para o projeto. Depois que o *Product Backlog* é definido, a equipe deve se dedicar à definição da *Sprint Backlog*, que contém uma lista de atividades que serão realizadas na próxima *sprint*, momento em que também são definidas as responsabilidades de cada membro do time.

Após os desenvolvedores discutirem quais padrões serão adotados, as atividades de análise, codificação e testes devem se iniciar. Conforme ilustra a Figura 11.2, ao final de cada *sprint* um incremento do produto deve ser apresentado ao cliente para que o time obtenha uma retroalimentação. Caso algum defeito seja encontrado, deve ser adicionado ao *backlog* do produto. Na medida em que o ciclo de desenvolvimento ocorre, são aplicados diversos mecanismos de controle do SCRUM, como, por exemplo, ações sobre as funcionalidades não entregues, a necessidade de mudanças para correção de defeitos, problemas técnicos encontrados, bem como os riscos e as estratégias para evitá-los.

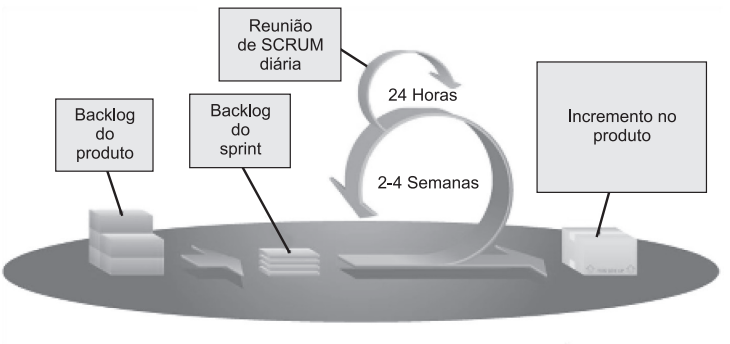


Figura 11.2 – Visão geral da dinâmica da metodologia SCRUM. Fonte: Cohn (2008).

11.4 Sprints

Projetos que utilizam o SCRUM progridem por intermédio de uma série sequencial de *sprints*, que correspondem a iterações. O objetivo é gerar um produto "entregável" de valor para o cliente, que foi previamente combinado com ele. Cada *sprint* deve ocorrer em um período de duas a quatro semanas. O produto é projetado, codificado e testado durante a *sprint*. Importante: nenhuma mudança deve ocorrer durante uma *sprint*, conforme ilustra a Figura 11.3.

[1] O termo *backlog* refere-se a um resumo histórico (log) de acumulação de trabalho em um determinado período de tempo.

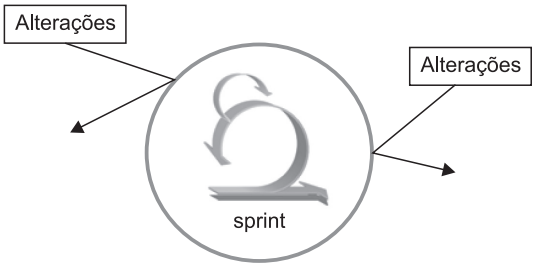


Figura 11.3 - Nenhuma alteração deve ser feita durante uma sprint.
Fonte: Arquivo pessoal do autor.

A *sprint* representa a unidade básica de desenvolvimento do SCRUM, entendida como um esforço dentro de uma "caixa de tempo"; em outras palavras, um esforço restrito a uma duração específica. Como veremos mais adiante, cada *sprint* é precedida por uma reunião de planejamento, com o objetivo de identificar quais tarefas serão executadas durante a *sprint* e assumir um objetivo para a *sprint*, resultando em um incremento de produto potencialmente entregável para o cliente.

As tarefas escolhidas para fazerem parte de uma *sprint* devem ser retiradas de outro documento, denominado *Product Backlog*, o qual contém um conjunto de requisitos que representam o trabalho que deve ser feito. No final de cada *sprint*, outra reunião deve ser realizada, objetivando revisar o que foi feito, avaliar o progresso e identificar lições aprendidas para serem usadas na próxima *sprint*. Cada *sprint* é uma iteração que segue um ciclo conhecido como PDCA (do inglês *Plan*, *Do*, *Check* e *Act*), que é aplicado para atingir resultados dentro de um sistema de gestão, podendo ser utilizado por qualquer empresa, independente da área de atuação.

11.5 Papéis

Como mencionamos anteriormente, podemos dizer que o SCRUM tem somente quatro papéis, sendo o *Product Owner*, o *SCRUM Master*, o *Team* e o cliente.

11.5.1 Product Owner

O *Product Owner* ("dono" do produto) representa o cliente e é responsável por garantir que a equipe SCRUM agregue valor ao negócio. Portanto, desempenha o papel de moderador entre os interesses do cliente e do *Team*, tendo como responsabilidade principal manter a equipe funcional e produtiva. De uma maneira resumida, o *Product Owner* é responsável por:

- Definir a visão e as funcionalidades do produto;
- Definir as prioridades;
- Elaborar e manter o *Product Backlog*;
- Definir as prioridades e o ROI (*Return of Investment*);
- Decidir sobre as datas de lançamento do produto;

- Representar o cliente (quando este não está presente);
- Aceitar ou rejeitar os resultados dos trabalhos.

11.5.2 SCRUM Master

Primordialmente, o *SCRUM Master* (mestre SCRUM) é o representante do cliente no projeto e também desempenha um papel importante de facilitador, responsável pela remoção de impedimentos (problemas técnicos, administração de conflitos, itens não planejados etc.) que eventualmente surjam durante o desenrolar do desenvolvimento.

Observe que o *SCRUM Master* não é o líder da equipe, preocupando-se com o uso correto do processo SCRUM e a aplicação das suas regras. Atua também na definição de funcionalidades de acordo com seu valor para o cliente, planejando e elaborando em conjunto com o *Product Owner* uma lista de prioridades. Portanto, o *SCRUM Master* desempenha um papel de responsabilidade técnica na condução do projeto, devendo proporcionar mecanismos preferencialmente informatizados de comunicação entre seus integrantes. O *SCRUM Master* também deve proteger a equipe e mantê-la focada em suas tarefas. A seguir podemos ver um resumo das responsabilidades inerentes ao *SCRUM Master*:

- Desempenhar o papel de líder, representando a gerência do projeto;
- Remover impedimentos;
- Proteger a equipe SCRUM;
- Ajudar o *Product Owner* com o *Product Backlog*;
- Ser o facilitador da equipe SCRUM, garantindo sua plena produtividade;
- Garantir a colaboração entre os diversos papéis e funções;
- Atuar como escudo para interferências externas;
- Aplicar os valores e as práticas SCRUM.

11.5.3 Team (time) ou equipe SCRUM

É o time responsável pelo desenvolvimento do projeto, sendo composto geralmente de um grupo de cinco a nove integrantes, que deve possuir uma característica multifuncional. No caso do desenvolvimento de software, por exemplo, uma equipe multifuncional deve contar com analistas, programadores, testadores, entre outros, que devem trabalhar com dedicação integral ao projeto, exceto em alguns casos, como, por exemplo, um DBA (*Data Base Administrator*). O SCRUM estimula a comunicação verbal entre todos os membros do projeto, principalmente pelo fato de o time possuir outra característica importante, relacionada à sua auto-organização. Preferencialmente seus integrantes não devem possuir títulos e os integrantes do time só devem ser trocados após o término de uma *sprint*. Resumindo, a equipe SCRUM é responsável por:

- Fazer as estimativas necessárias;
- Definir as tarefas que serão realizadas;

- Desenvolver o produto;
- Garantir a qualidade do produto;
- Apresentar o produto ao cliente.

Como ocorre com qualquer equipe de desenvolvimento, os integrantes do time podem possuir experiências diversas. Funcionários *trainees*, por exemplo, são normalmente recém-formados ou estão nos últimos anos do ensino superior e se encontram nos primeiros estágios da carreira. Os que são considerados plenos possuem entre três e cinco anos de experiência. Já os seniores são aqueles que já passaram pela fase de carreira plena.

11.5.4 Cliente

O cliente deve participar de tarefas relacionadas à implementação da lista de funcionalidades e atuar ao longo do desenvolvimento do projeto por meio de análises periódicas dos subprodutos gerados.

11.6 Cerimônias

São reuniões que ocorrem em momentos diferentes de uma determinada *sprint*. Resumem-se em quatro momentos específicos, denominados:

- Planejamento da *sprint (Sprint Planning Meeting)*;
- Reunião diária *(Daily Meeting ou Daily SCRUM)*;
- Revisão da *sprint (Sprint Review)*;
- Retrospectiva da *sprint (Sprint Retrospective)*.

11.6.1 Planejamento da sprint ou Sprint Planning Meeting

É a primeira reunião do projeto e todos precisam participar; deve ter uma duração de no máximo oito horas. Esta é a reunião em que o *Product Owner* planeja e elabora a lista de prioridades que devem ser cumpridas pelo projeto, dividindo a reunião em duas partes:

Parte 1: o *Product Owner* define suas prioridades, seleciona os itens do *backlog* e a meta da *sprint*.

Parte 2: a equipe define a *sprint backlog*, que é um documento que contêm as tarefas necessárias para cumprir a meta.

11.6.2 Pôquer do planejamento

Durante o planejamento da *sprint* podemos utilizar uma técnica que estima o tamanho do trabalho a ser realizado, aplicada por uma dinâmica de grupo conhecida como "pôquer do planejamento". Essa técnica dá a impressão de que os jogadores (time de desenvolvimento) estão jogando

cartas, como num jogo de pôquer. De fato estão, mas as cartas utilizadas têm um propósito lúdico específico. Como vimos, no SCRUM não pensamos em tempo, ou seja, duração das tarefas, mas no tamanho delas. Então, para que possamos definir esse tamanho, a equipe senta ao redor de uma mesa e um dos integrantes apresenta para todos uma "história", que está relacionada a uma funcionalidade que deve ser desenvolvida.

Após o time de desenvolvimento entender claramente o objetivo da história, cada membro do time escolhe uma carta, que foi previamente distribuída a todos, e coloca na mesa virada para baixo. Apenas quando todas as cartas estiverem lançadas é que elas são viradas. As cartas apresentam valores numéricos relacionados com o grau de complexidade da funcionalidade a ser desenvolvida, sob o ponto de vista de cada membro da equipe.

O objetivo é obter um consenso relacionado ao tamanho do desenvolvimento da funcionalidade. Portanto, se por algum motivo aparecer um valor bem diferente ou a equipe não chegar a um consenso, uma breve discussão deve ser iniciada para resolver essa divergência, seguida de uma nova rodada. Ao final, o time calcula a média dos valores apresentados, chegando a uma pontuação final sobre a história apresentada. A Figura 11.4 apresenta uma visão geral das cartas que podem ser utilizadas.

Figura 11.4 – Cartas utilizadas no "pôquer do planejamento".

Existem algumas cartas especiais, para serem utilizadas em situações específicas. A carta com o valor zero representa a percepção de um integrante do time sobre o fato de a história ser tão pequena que leva apenas alguns minutos de trabalho para concluí-la. A carta com a interrogação (?) representa que o integrante do time não faz a menor ideia do tempo que ele levaria para concluir a funcionalidade. Já a xícara de café representa a seguinte situação: "estou cansado demais para pensar. Vamos fazer uma pausa?".

11.6.3 Reunião diária ou Daily SCRUM

É uma reunião ao mesmo tempo simples e importante, sendo um dos grandes diferenciais do SCRUM. Nessa reunião cada membro do time deve responder sobre o que já fez, sobre o que pretende fazer e se há algum impedimento para a conclusão da(s) tarefa(s) sob sua responsabilidade. A reunião diária deve ter uma duração muito rápida, preferencialmente de no máximo quinze minutos, tendo como participantes apenas o time e o *SCRUM Master*. Conforme mencionamos, cada membro do time deve responder a três perguntas:

- O que eu fiz desde a última reunião?
- O que vou fazer até a próxima?
- Tive ou estou tendo algum impedimento? Quais?

É importante observar que todos os integrantes do time devem participar dessa reunião e fomentar a ideia de que não se trata de uma "pausa para o lanche", ou seja, por ser curta e objetiva, pode passar a ideia de que é uma brincadeira, principalmente para aqueles que não estão acostumados com o uso dessa metodologia. Tomar o cuidado de focar, no máximo, as três perguntas mencionadas, para que a reunião seja de fato efetiva, alcançando seu propósito.

Um bom momento para agendar as reuniões diárias é depois do almoço. Como elas não demoram, o ideal é que todos fiquem de pé, ao lado de um *task board* (quadro de tarefas), para que mantenham suas energias, considerando o fato de que depois do almoço normalmente as pessoas ficam cansadas e, muitas vezes, sonolentas. Outra razão para realizar a reunião nesse horário é que, como todo o time esteve trabalhando durante a manhã, provavelmente suas mentes ainda estão focadas no trabalho e não em questões pessoais.

11.6.4 Revisão da sprint ou Sprint Review

Basicamente é uma reunião de balanço sobre tudo o que foi feito durante uma *sprint*. Nessa reunião o time deve mostrar os resultados da *sprint* para o *Product Owner* e seus convidados. Observe que devem ser apresentados somente os itens que estiverem 100% prontos, ou seja, se faltou uma única atividade, o item não deve ser apresentado.

Para iniciar a reunião, um membro do time apresenta a meta do *sprint*, os itens do *Product Backlog* com os quais o time se comprometeu e todos os itens que foram concluídos. Em seguida, os membros do time fazem uma demonstração funcional dos itens desenvolvidos. Essa reunião deve ter uma duração estimada de quatro horas e precisa contar com os seguintes participantes: *Product Owner*, *SCRUM Master*, Equipe SCRUM e outros convidados (se alguém julgar necessário). Deve-se marcar essa reunião sempre no final da *sprint*. Portanto, os seguintes objetivos são esperados após a reunião:

- Apresentar o que a equipe fez durante a *sprint*;
- Entregar o produto (software funcionando) ao *Product Owner* (geralmente uma demo da parte implementada).

Após a apresentação, o *Product Owner* conversa com seus convidados e tem o direito de aceitar ou rejeitar a *sprint* com base no que foi apresentado. Qualquer necessidade de mudança ou inserção de novas *features* (funcionalidades) será incorporada ao *Product Backlog* em um momento oportuno, além de priorizada novamente.

11.6.5 Retrospectiva da sprint ou Sprint Retrospective

O objetivo dessa reunião é verificar o que houve de bom e o que pode ser melhorado em uma *sprint*. São avaliados aspectos relacionados ao trabalho em equipe, pontos positivos e negativos, e feitas reflexões sobre estratégias de melhoria que podem ser adotadas. Devem participar da reunião, obrigatoriamente, o time e o *SCRUM Master*. O *Product Owner* também pode participar, sempre que convidado. Todos os membros do time devem responder basicamente a duas perguntas: o que foi bom durante a *sprint* e o que se pode fazer para melhorar a próxima. O *SCRUM Master* deve tomar nota de tudo e o time deve priorizar os itens apontados em uma

ordem ideal de mudança. A retrospectiva é uma excelente forma de garantir a melhoria contínua do processo. Essa reunião deve acontecer logo após a revisão da *sprint*, com duração aproximada de três horas.

11.7 Artefatos

Os artefatos propostos pela metodologia SCRUM são produzidos em momentos diferentes de uma determinada *sprint*, resumindo-se a três: o *Product Backlog*, o *Sprint Backlog* e o *Burndown Chart*.

11.7.1 Product Backlog

É um documento que representa a visão do produto de forma modular, contendo todos os itens que devem ser desenvolvidos durante o projeto. Basicamente é uma lista de prioridades feitas logo no início do projeto, com o objetivo de esclarecer e elencar o que deve ser entregue para o cliente. Esses itens devem ser escritos de forma clara e simples, de fácil entendimento tanto para o time de desenvolvimento quanto para o cliente. O *Product Backlog* deve ser criado e mantido pelo *Product Owner*, que tem a liberdade de alterar esse documento quando quiser, desde que os itens alterados não estejam na *sprint* que estiver sendo desenvolvida no momento. A Figura 11.5 apresenta um exemplo de conteúdo de um *Product Backlog*.

Nível de Prioridade (ROI)	Ator	Requisitos Funcionais	Descrição do Item	Tamanho	Release	Sprint	Status
Essencial	Usuário	RF001	Cadastro de usuário				To Do
Essencial	Usuário	RF002	Cadastro de curriculum				To Do
Essencial	Usuário	RF003	Cadastro de interesse				To Do
Essencial	Usuário	RF004	Busca de oportunidades de emprego				To Do
Essencial	Empresas	RF005	Cadastro de empresas				To Do
Essencial	Empresas	RF006	Busca de candidato a emprego				To Do

Figura 11.5 - Exemplo de conteúdo de um Product Backlog. Fonte: arquivo pessoal do autor.

Os itens do *Product Backlog* devem ser priorizados em função do ROI (*Return of Investment*), ou seja, os que apresentarem maior valor para o negócio devem ser desenvolvidos primeiro. O tamanho de cada item deve ser estimado pelo time de desenvolvimento. O *Product Owner* deve manter e priorizar o *Product Backlog* constantemente.

11.7.2 Sprint Backlog

É um artefato oriundo da *Sprint Planning Meeting* e representa todas as tarefas que devem ser desenvolvidas durante uma *sprint* ou iteração. Conforme mencionamos, para melhor rendimento da *sprint*, o *Sprint Backlog* não deve sofrer alterações. A Figura 11.6 apresenta um exemplo de *Sprint Backlog*.

Item	Tarefa	1	2	3	4	5	6	7	8	9	10	Total
ES006	Criar Base de Dados	4	0	0	0	0	0	0	0	0	0	4
	Desenvolver Modelo	8	8	0	0	0	0	0	0	0	0	16
	Desenvolver Controle	0	0	4	4	0	0	0	0	0	0	8
	Desenvolver View	0	0	0	4	4	0	0	0	0	0	8
	Teste de Unidade	0	0	0	0	4	2	2	0	0	0	8
	Teste Funcional	0	0	0	0	0	4	4	2	0	0	10
	Documentação Técnica	0	0	0	0	0	0	4	4	2	2	12

Figura 11.6 - Exemplo de conteúdo de um Sprint Backlog. Fonte: arquivo pessoal do autor.

Assim como o *Product Backlog*, o *Sprint Backlog* deve ser representado em função do ROI. Cada item deve ser detalhado em tarefas e cada uma dessas tarefas deve ter uma estimativa de esforço, neste caso, em horas. Para melhor visibilidade do projeto, o *Sprint Backlog* pode ser apresentado na forma de quadro inspirado no kanban, que é uma canaleta visual utilizada para sinalizar o estado atual dos projetos. Tem como principal objetivo dar visibilidade ao andamento das tarefas, tendo se mostrado mais eficaz quando fica visível e de fácil acesso a todos os membros do time.

É interessante que o time só comece a trabalhar o segundo item sempre após a conclusão do primeiro. Esse tipo de comportamento evita que o time coloque a meta em risco, uma vez que o kanban também é priorizado em função do ROI, ou seja, quanto mais no topo estiver, maior é o valor que ele representa para o cliente. Esse quadro também é conhecido como *task board* (quadro de tarefas), cujas características serão detalhadas a seguir.

11.7.3 Task Board

É um quadro utilizado para o acompanhamento das *sprints*, principalmente durante as reuniões diárias. Por meio das informações contidas nesse registro, aliadas ao seu posicionamento no *task board*, torna-se possível a qualquer um observar o andamento do projeto, de maneira clara e intuitiva. As informações sobre as tarefas a serem realizadas são registradas no *task board* geralmente por "*post-it*" ou escritas diretamente em um quadro branco, previamente preparado para essa função. A Figura 11.7 ilustra uma visão geral dos principais elementos presentes no *task board*.

Product Backlog	To Do	Doing	Done
Sprint Backlog	**Unplanned Items**	**Impediments**	**To Discuss**

Figura 11.7 - Principais elementos de um task board. Fonte: arquivo pessoal do autor.

Post-it (registro de atividades no task board)

Quando a *sprint* se inicia, os integrantes do projeto decidem em comum acordo quais atividades irão desenvolver e normalmente registram-nas em pequenos papéis (*post-it*), transferindo-os em seguida para o espaço "*To Do*", comentado mais adiante. Muitas equipes que utilizam SCRUM fazem uso dos tradicionais "*post-it*" coloridos encontrados no mercado, nos quais escrevem o que devem fazer a caneta, de forma simples e objetiva. Contudo, existem outras propostas de registro de "*post-it*", como a desenvolvida pelos autores e ilustrada na Figura 11.8.

Neste exemplo, podemos observar a utilização de campos específicos, indicando informações que devem ser registradas. Essa alteração foi inspirada em registros propostos pela metodologia FDD (estudada no capítulo 6), que tem o propósito de indicar o andamento da tarefa, expressa em porcentagem de completude. A ideia é que os membros da equipe indiquem a porcentagem que corresponde ao andamento da tarefa, durante as reuniões diárias, possibilitando a todos uma percepção mais precisa do tempo necessário para a conclusão da tarefa. Essa ficha de tarefas indica ainda o ID do requisito, ou seja, a identificação da origem (funcionalidade) da tarefa que está sendo executada. Temos também a duração (em horas) que foi calculada previamente, além do nome e da descrição resumida da tarefa.

ID Requisito:					Duração (hs):				
Tarefa:									
Descrição:									
Responsável:									
Andamento (% concluída):									
10	20	30	40	50	60	70	80	90	100

Figura 11.8 – Exemplo de registro de tarefa em um ***task board****.*
Fonte: Arquivo pessoal do autor.

De maneira resumida, o *task board* possui a seguinte dinâmica:

- ***Product Backlog*:** o espaço reservado para o *Product Backlog* contém os "*post-it*" de todas as atividades necessárias para a conclusão do projeto.
- ***Sprint Backlog***: nesse local devemos relacionar todas as atividades de uma iteração específica (*sprint*), oriundas do espaço do *Product Backlog*. As atividades descritas correspondem

às funcionalidades desejadas. Portanto, dependendo da funcionalidade, essa atividade pode ser fragmentada em várias outras subtarefas, responsáveis pela sua completude.

- ***To Do* (para fazer):** espaço que indica as atividades necessárias para que todas as tarefas presentes do espaço da *Sprint Backlog* sejam desenvolvidas durante a *sprint* corrente.
- ***Doing* (fazendo):** esse espaço indica as atividades em andamento, o responsável por elas ou qualquer outra informação encontrada no "*post-it*". À medida que os integrantes do time concluem suas atividades, transferem-nas para a coluna "*Done*".
- ***Done* (feito):** esse espaço indica as atividades concluídas. Lembrar que toda movimentação de "*post-it*" pelo *task board* só deve ser feita durante a reunião diária.
- ***To Verify* (para verificar):** esse espaço é utilizado quando existem atividades concluídas que já podem ser testadas. Por meio desse local os testadores já podem se mobilizar e testar as funcionalidades à medida que são liberadas, se isso for possível.

O *task board* também pode conter espaços adicionais, utilizados em situações específicas, conforme ilustrado a seguir:

- ***Unplanned Items* (itens não planejados):** esse espaço é utilizado para indicar atividades que devem ser executadas, mas que não haviam sido previstas inicialmente.
- ***Impediments* (impedimentos):** indica atividades que estão paralisadas por alguma razão, devendo ser objeto de atenção especial principalmente do *SCRUM Master* que, como vimos, desempenha o papel de responsável técnico do projeto.
- ***To Discuss* (para discutir):** adicionalmente, pode-se usar a coluna "*To Discuss*" para evidenciar atividades cuja execução demanda discussão técnica.

Como vimos, a mudança das atividades através das colunas e espaços do *task board* é feita pelos integrantes de projeto durante as reuniões diárias. A *sprint* termina quando todas as atividades existentes na *Sprint Backlog* forem concluídas. A partir daí, nova *sprint* se inicia, com novos "*post-it*" advindos do *Product Backlog*.

11.7.4 Gráfico Burndown

Tem como objetivo mostrar o esforço restante para a conclusão da iteração, bem como mostrar o quão próximo ou distante o time está de atingir a meta. A Figura 11.9 mostra um exemplo desse gráfico, em que podemos observar uma coluna vertical, que representa a quantidade de esforço (quantidade de trabalho que ainda precisa ser realizada), e uma coluna horizontal, representando os dias de uma iteração.

A linha vermelha indica o fluxo ideal de trabalho, ou seja, quando o gráfico está acima da linha vermelha, o time está distante de atingir a meta. Em cima da linha vermelha, o time está em um fluxo de trabalho ideal. Abaixo da linha vermelha, o time está superando as expectativas. A metodologia prega que esse gráfico seja atualizado diariamente.

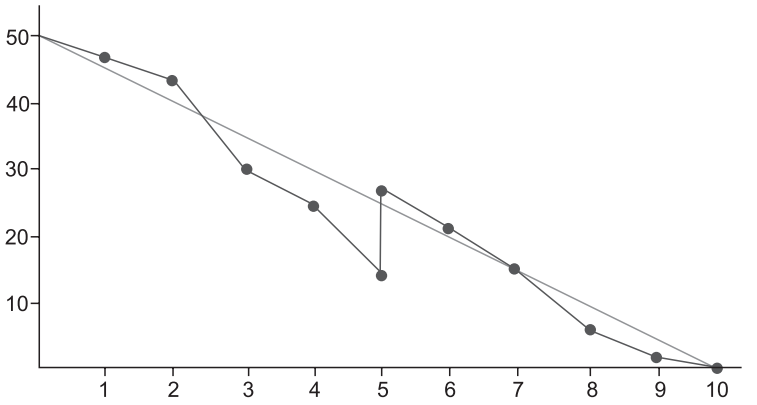


Figura 11.9 – Exemplo de uso de um Burndown Chart.

11.8 Exercícios

1) Podemos utilizar o SCRUM apenas para desenvolver software? Comente.

2) Na metodologia SCRUM, o que representa uma *sprint*?

3) Quais os três principais papéis utilizados pelos praticantes da metodologia SCRUM?

4) A metodologia SCRUM prega a realização de determinadas cerimônias durante o processo de desenvolvimento. Quais são essas cerimônias e os principais artefatos gerados ao final delas?

5) O que deve ocorrer ao final de cada *sprint*?

6) Qual o papel desempenhado pelo chamado *Product Owner*?

7) O *SCRUM Master* representa o líder da equipe? Comente seu papel.

8) Quais são as responsabilidades da equipe SCRUM?

9) Qual é o objetivo do pôquer do planejamento?

10) Qual é o objetivo da reunião de retrospectiva da *sprint* e quem deve participar dela?

12

Iconix Process

Objetivos

- *Contextualizar o estudante nas características da metodologia Iconix.*
- *Apresentar suas fases.*
- *Mostrar exemplo de utilização.*

12.1 Considerações iniciais

Essa metodologia ágil foi criada em 1999 e publicada em 2005 por Doug Rosenberg, fundador e presidente da *Iconix Software Engineering Inc.*, juntamente com Matt Stephens e Mark Colling-Cope, especialistas em projetos de desenvolvimento ágil, ambos com larga experiência em desenvolvimento, arquitetura, interação e design de softwares. A Iconix mantém um web site com informações sobre o processo Iconix, bem como outras tecnologias e pesquisas nessa área: www.iconixsw.com.

Iconix diz respeito a uma modelagem dirigida por casos de uso (*use case driven*), que tem como objetivo estudar e comunicar o comportamento do sistema sob o ponto de vista de um consumidor ou usuário final. Embora tenha sido criada antes da UML, utiliza-se de um de seus diagramas, que permite a compreensão da solução proposta por todas as pessoas, mesmo que não envolvidas no processo de desenvolvimento.

O Iconix é considerado um método ágil não tão burocrático e formal quanto o RUP (*Rational Unified Process*), nem tão radical quanto a XP (*Extreme Programming*). Possui uma quantidade razoável de documentação, sendo considerado uma metodologia prática e simples, mas também poderosa e com um componente de análise e representação dos problemas sólido e eficaz. Por esses motivos, a metodologia Iconix é caracterizada como um processo de desenvolvimento de software.

12.2 Características

O processo Iconix, além de fazer uso da linguagem de modelagem UML, possui uma característica exclusiva chamada "rastreabilidade dos requisitos" (*Traceability of Requirements*). Essa rastreabilidade permite, por meio de seus mecanismos, verificar em todas as fases se os requisitos estão sendo atendidos, à medida que o projeto vai sendo desenvolvido. Essa metodologia dá um enfoque especial aos requisitos e é perfeitamente adequada a empresas que pleiteiam um certificado de qualidade de software, mais precisamente certificados que exigem como primeiro passo a gerência de seus requisitos, como CMMI, MPS.BR etc.

A abordagem Iconix é flexível e aberta, isto é, se for necessário usar outro recurso da UML para complementar os recursos usados nas fases do Iconix, não há problema algum. Outra característica desse método é a utilização do diagrama de robustez, que representa um diagrama de classes estereotipado.

12.3 Fases do Iconix

A seguir, vamos resumir as fases dessa metodologia, bem como sua estrutura de funcionamento:

a) Modelagem do domínio

Fase que identifica o domínio do problema e seus objetos. Os desenvolvedores devem identificar e abstrair os objetos do domínio de aplicação existente no mundo real, seus relacionamentos, generalizações e agregações. Nesse momento, pode-se começar a criar um diagrama de classes de alto nível. Se for o caso, os desenvolvedores podem fazer um protótipo rápido do sistema ou obter informações sobre sistemas anteriores (legados), para que se faça uma reengenharia. Quando essa etapa terminar, um modelo razoavelmente correto do domínio deve ter sido construído, e todos os envolvidos no projeto (cliente, programadores, analistas, verificadores e mesmo usuários finais) poderão compartilhar e usar comumente o sistema prototipado. O modelo do domínio continuará a evoluir e crescer (até atingir o domínio do negócio) enquanto o projeto progride.

b) Identificação do domínio dos objetos quanto ao mundo real

Após ter refinado o progresso do trabalho usando um checklist, o próximo passo é encontrar substantivos e frases que podem ser transformados em classes e atributos. Os verbos encontrados transformam-se em operações e associações, e em certos casos algumas frases podem se tornar atributos. Isso é apenas uma regra geral, uma vez que naturalmente você pode encontrar alguns verbos que se transformam eventualmente em objetos ou operações, principalmente no caso da modelagem de um processo de negócio ou identificação de elementos de gerência ou classes de controle.

c) Comportamento dos requisitos por meio de casos de uso

Nessa fase os requisitos captados dos *stakeholders* são interpretados e revisados, a fim de que estejam livres de ambiguidades. Após essa revisão, utiliza-se o diagrama de casos de uso da UML para sua representação.

Como benefício desse processo é possível retirar vários elementos, tais como:

- O entendimento dos processos por qualquer pessoa (participação dos usuários);
- Facilidade em enxergar como tudo vai funcionar (interação do usuário);
- Divisão das atividades em grupos, possibilitando a criação de um diagrama de pacotes;
- Documentação do sistema através desse diagrama.

d) Análise de robustez

Essa análise também tem a finalidade de descobrir eventuais ambiguidades, não dos requisitos, mas dos casos de uso representados. Ela é feita por um diagrama de robustez que, conforme mencionado, representa um diagrama de classes estereotipado. Essa fase conecta a análise ao projeto, certificando-se de que a descrição dos casos de uso esteja correta. Também nessa fase é possível descobrir novos objetos, que não foram vistos no modelo de domínio. Ela envolve ainda a análise do texto narrativo de cada caso de uso e a identificação de um primeiro conjunto de possíveis objetos que participarão do caso de uso. A seguir, apresentamos as principais características do diagrama de robustez:

Diagrama de robustez

Trata-se de um diagrama aderente às arquiteturas atuais do MVC - Modelo, Visão e Controle (*Model View Controller*), que auxilia o desenvolvedor, gerenciando, visualizando e persistindo os dados de uma aplicação. A palavra robustez dá uma ideia de força e vigor, e neste caso específico indica a utilização de um método capaz de traçar passo a passo toda a interação do usuário com o sistema. Envolve um conjunto de símbolos representativos que indicam a interface utilizada, os menus e possíveis resultados obtidos, facilitando assim o entendimento e a identificação dos objetos e ações (operações) propostas pelo sistema. Trata-se se um diagrama composto basicamente de três símbolos, conforme ilustra a Figura 12.1.

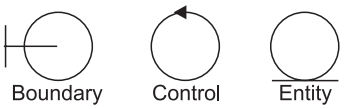


Figura 12.1 – Simbologia do diagrama de robustez.

- **Boundary:** refere-se à interface que será oferecida ao usuário do sistema.
- **Control:** refere-se ao controlador com as regras de negócio do sistema.
- **Entity:** refere-se ao armazenamento dos dados provenientes do sistema.

e) Comportamento dos objetos

Essa fase representa praticamente o começo do projeto, ou seja, dá-se início ao conjunto de ações ou passo a passo de todo o processo de utilização e regras de negócio. Para essa fase usam-se os diagramas de sequência ou de atividades. A ideia é utilizar uma representação gráfica da intera-

ção e dos resultados, objetivando mostrar como as ações executadas pelo sistema se comportam. Obviamente, deve-se, sempre que necessário, atualizar os diagramas gerados anteriormente, pois nessa fase muitos erros precisam ser corrigidos ou novas possibilidades não observadas inicialmente podem ser incluídas no projeto.

f) Modelo estático

Esse modelo baseia-se na construção de um diagrama de classe consolidado e totalmente de acordo com os diagramas de caso de uso e atividades (sequência) realizados anteriormente. O diagrama criado servirá de apoio para a divisão do projeto em partes, que serão utilizadas pela gerência de projeto para distribuição das tarefas da equipe de desenvolvimento. Esse modelo, além do diagrama, deve possuir os detalhes do projeto e também ser validado pela equipe de *stakeholders* (interessados no sistema).

g) Codificação (código-fonte)

Quando o modelo estático for validado, pode-se dar início à sua codificação, que deve ser acompanhada periodicamente da realização de testes de unidade em todos os módulos. Esses testes são feitos para garantir a qualidade do código e evitar retrabalho. A partir desse momento, o protótipo passa a ser considerado o sistema final.

h) Testes

Finalmente, nessa fase são realizados os testes "caixa-preta" e de aceitação por um grupo de usuários, juntamente com projetistas e programadores diferentes daqueles que codificaram o sistema. Esses testes são realizados exaustivamente até a aceitação do software pelo cliente, atestando que está livre, pelo menos até aquele momento, de erros.

Nota:

O teste caixa-preta consiste em testar as funcionalidades do software sem conhecer o código-fonte, ou seja, usar as interfaces oferecidas pelo software de forma a checar se tudo funciona conforme o planejado.

12.4 Exemplo de utilização

Para observarmos como o Iconix pode ser aplicado na prática, vamos tomar como exemplo um sistema para controle de uma clínica médica. Com este exemplo pretendemos usar alguns diagramas, apenas para ilustrar como o Iconix é utilizado.

Cenário a ser estudado

Uma clínica médica deseja informatizar todos os processos relativos a agendamento e consulta de pacientes. A clínica conta com uma atendente responsável pelo preenchimento da ficha dos pacientes com dados como ID (número de identificação), nome, endereço, cidade, telefone, e-mail, data de nascimento, nome do pai e nome da mãe. Além disso, cabe também à atendente agendar os pacientes, anotando o dia/mês, o horário previsto do atendimento e o nome do médico. O processo atualmente é feito de forma manual, com um livro para cada médico. Os médicos possuem especialidades diferentes, além de seus dados pessoais, como CRM, nome, endereço, telefone e e-mail.Observação: as consultas só podem ser realizadas mediante agendamento.

Primeiro passo: entrevista, questionário ou atas de reunião

Como primeiro passo para utilização da metodologia Iconix, o gerente de projeto, após sua equipe formada, irá solicitar que um analista visite a empresa (clínica) para captar os requisitos. Esses requisitos serão obtidos em uma entrevista ou questionário previamente elaborado. O questionário deve conter questões gerais sobre o sistema, que permitirão ao analista criar um modelo de domínio da aplicação desejada. Obviamente outras entrevistas serão necessárias para captar requisitos específicos de cada parte do processo.

Essa etapa da metodologia deve fazer parte da documentação a fim de realizar a verificação e validação do sistema proposto. O analista deve seguir as normas gerais de entrevistas, tomando sempre o cuidado de não constranger o cliente, como, por exemplo, fazendo parecer que entende mais do seu negócio do que ele. Não se esquecer também de conseguir cópias de fichas, relatórios ou páginas de livros utilizados nos processos manuais que serão informatizados.

Segundo passo: ficha de requisitos

Após a captação dos requisitos, eles devem ser transcritos em um modelo de formulário que permita ao desenvolvedor entender o que o cliente deseja, ou seja, o problema a ser resolvido. Essa ficha pode ser, por exemplo, o *template Volere*, que permite, entre outras opções, descrever o requisito e enfatizar suas prioridades. Esse *template* fornece um modelo sólido para especificação de requisitos utilizado em todo o mundo, criado pela empresa Atlantic Systems Guild, podendo ser encontrado no endereço web http://www.volere.co.uk/template.htm. A organização dos itens da ficha possibilita a rastreabilidade dos requisitos pelo seu número (identificação do requisito). A Figura 12.2 ilustra um exemplo de uso do *template Volere*©.

Ficha de Especificação de Requisitos
(baseada no template Volere)

Identificação do Requisito: RF001 (Agendamento)

Tipo de Requisito: funcional

Caso(s) de Uso Vinculado(s): Cadastrar Paciente, Cancelar Agendamento, Cadastrar Médicos.

Descrição:

Este requisito possibilita à atendente agendar os pacientes para consulta, anotando o dia/mês, o horário previsto do atendimento e o nome do médico que fará o atendimento.

Justificativa: agendar consultas entre médicos e pacientes.

Solicitante: *stakeholder* Dr. Júlio

Prioridade: 3 (alta)

Material de apoio: Diagrama de Casos de Uso (DCU-Base)
Diagrama de Classes (DC-Base)
Diagrama de Entidades e Relacionamentos (DER-Base)

Histórico: Solicitação inicial (/ /2012)
Solicitação de mudança (/ /2012)

Dados:

DataAtend

HoraAtend

Paciente

Médico

Figura 12.2 - Exemplo de uso do template Volere.

Terceiro passo: diagrama de casos de uso

Após as fichas de requisitos serem validadas e estarem livres de ambiguidades, podemos utilizar o diagrama de casos de uso, conforme ilustra a Figura 12.3.

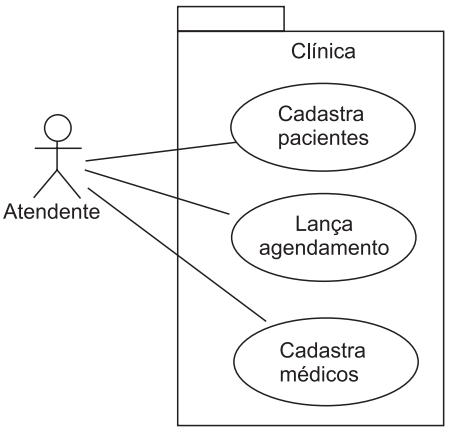


Figura 12.3 - Diagrama de casos de uso do cenário estudado.

Neste diagrama de exemplo, o ator atendente tem três tipos de funcionalidade disponíveis: cadastrar os pacientes, efetuar um agendamento e cadastrar os médicos da clínica.

Quarto passo: diagrama de robustez

A partir do diagrama de casos de uso podemos fazer o diagrama de robustez, conforme a Figura 12.4.

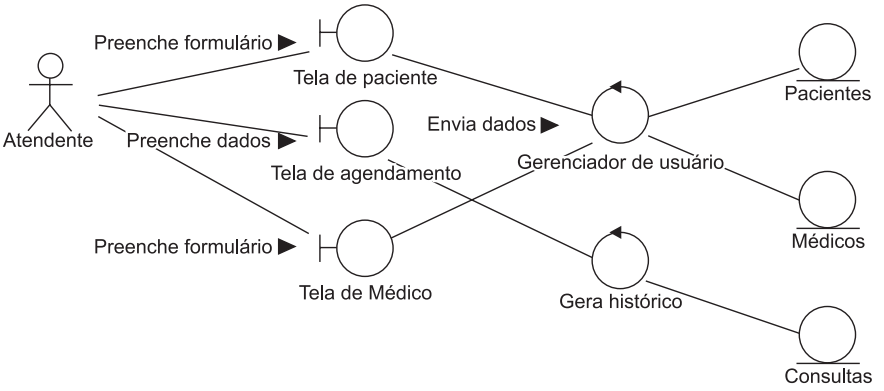


Figura 12.4 - Diagrama de robustez para o cenário estudado.

Este diagrama destaca a interação do atendente com as diferentes interfaces oferecidas pelo sistema.

Quinto passo: diagrama de sequência/atividades

Após a visão sistêmica de como os processos vão ocorrer, o arquiteto de software pode optar por fazer o diagrama de sequência (mais utilizado) ou de atividades, o qual complementa o diagrama de robustez no quesito *feedback*, pois as mensagens de retorno são o seu ponto forte. A Figura 12.5 apresenta um exemplo de uso do diagrama de sequência para o cenário estudado.

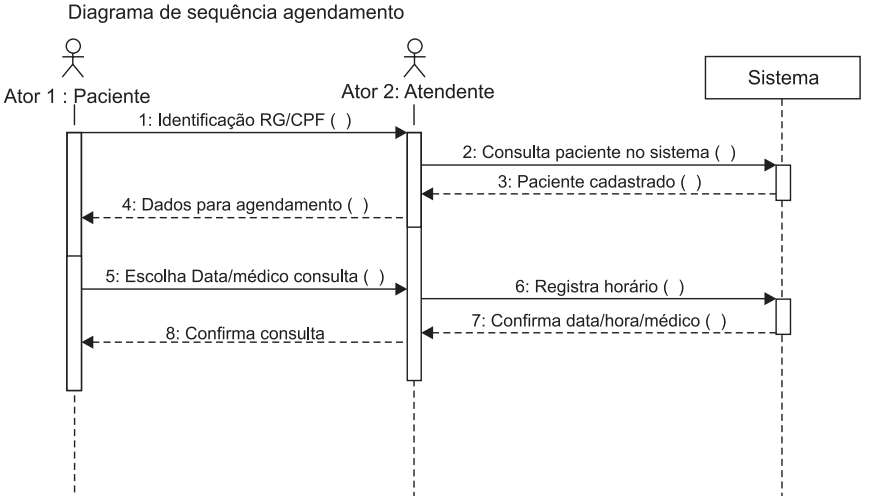


Figura 12.5 - Diagrama de sequência do cenário estudado.

Sexto passo: diagrama de classes

A partir dos diagramas anteriores torna-se possível elaborar o diagrama de classes, que é fundamental para a implementação do software. A Figura 12.6 mostra como esse diagrama poderia ser criado, considerando o cenário apresentado.

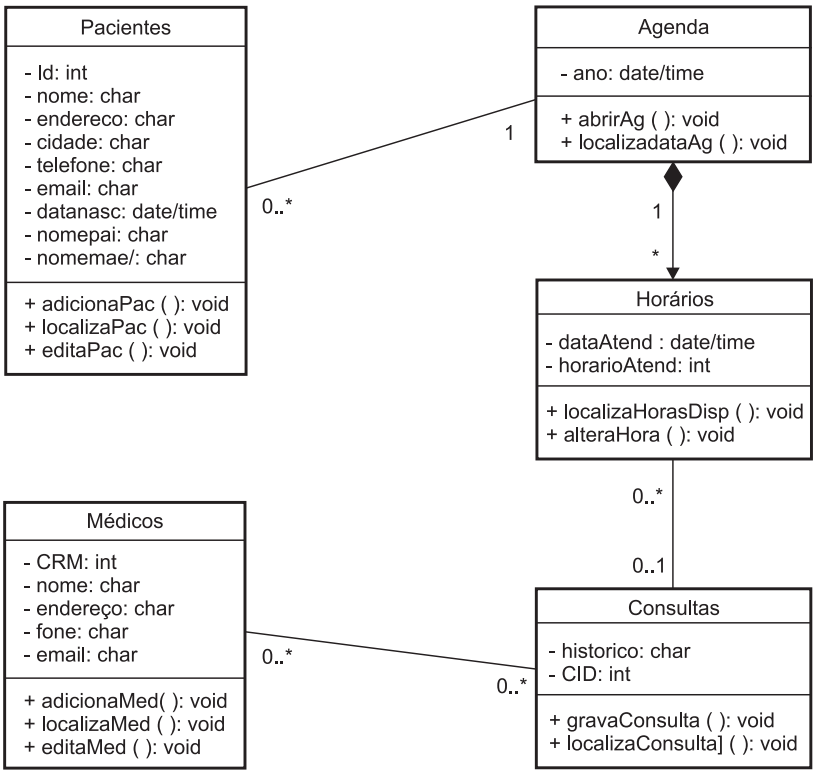


Figura 12.6 – Diagrama de classes do cenário estudado.

Sétimo passo: codificação

Após todos os diagramas serem validados, pode-se dar início à etapa da codificação, na qual o gerente de projeto pode solicitar algum tipo de padrão de projeto (como, por exemplo, o *Design Patterns*).

Oitavo passo: testes

O Iconix solicita que seja contemplada no projeto uma etapa de testes, utilizando testes funcionais (caixa-branca e preta), devidamente documentados.

Observação:

O exemplo abordado serve apenas para entender a sequência de utilização dos recursos e das exigências da metodologia Iconix. Fica a cargo da equipe de desenvolvimento escolher a melhor estratégia, diagramas e documentos necessários.

12.5 Casos de Sucesso

Algumas empresas de desenvolvimento utilizam em seus projetos a metodologia Iconix. A seguir, citamos alguns exemplos de projetos realizados:

12.5.1 DMV

Esse projeto foi feito para uma agência governamental americana de veículos motorizados, no estado da Virgínia. O sistema faz a regulação, o licenciamento e o enquadramento nas leis de trânsito. Trata-se de 74 centros de atendimento e 40 escritórios em todo o estado, os quais trabalhavam com um sistema legado que não atendia mais os requisitos e a demanda necessária. O *case* está disponível para leitura e conhecimento no endereço web: http://www.sparxsystems.com.au/downloads/pdf/dmv_csi_case_study.pdf, em que se pode verificar o uso da ferramenta Enterprise Architect©, que possui um *template* preparado para projetos que utilizam a metodologia Iconix.

12.5.2 SIG

Em um outro projeto feito sob medida para a Environmental Systems Research Institute (ESRI), a empresa Sparx Systems utilizou a metodologia Iconix para desenvolver um Sistema de Informação Geográfica (SIG), o qual oferece acesso a web services para plataformas móveis. Um artigo desse projeto está disponível para maiores detalhes em: http://www.sparxsystems.com.au/downloads/pdf/esri_lbs_delivery_case_study.pdf.

12.5.3 LSST

O Large Synoptic Survey Telescope (LSST) foi um projeto desenvolvido com a metodologia Iconix, o qual visa controlar e manter um acervo muito grande de imagens enviadas por um telescópio de grande porte, a fim de analisar os dados e emitir possíveis alertas à comunidade com relação a desastres causados pelo mau tempo. Maiores detalhes sobre esse *case*, bem como sobre a metodologia empregada, podem ser encontrados no endereço web http://iconixsw.com/Articles/LSST_case_study_20080825.pdf.

No web site www.iconixsw.com encontram-se várias informações sobre certificações e artigos de utilização da metodologia Iconix.

12.6 Exercícios

1) Na metodologia Iconix, como podemos descrever a rastreabilidade dos requisitos?

2) Qual o diagrama utilizado no padrão MVC adotado pela metodologia Iconix?

3) Na metodologia Iconix, quais métodos podem ser utilizados como instrumentos de coleta de dados na captação de requisitos?

4) Que tipo de teste pode ser utilizado na metodologia Iconix?

5) Quais as fases utilizadas pela metodologia Iconix?

Anotações

13

Metodologias Tradicionais X Ágeis

Objetivos

- *Enfatizar as diferenças entre as metodologias tradicionais e as ágeis.*
- *Apresentar uma visão comparativa entre as metodologias tradicionais e as consideradas ágeis.*
- *Apresentar considerações gerais sobre testes de software no contexto das metodologias ágeis.*
- *Apresentar considerações sobre a obtenção de qualidade de software pelo uso de metodologias ágeis.*

13.1 Introdução

Como pudemos observar ao longo dos capítulos anteriores, as metodologias ágeis têm sido apontadas como uma alternativa às abordagens tradicionais, por serem menos burocráticas e flexíveis, permitindo ajustes de direcionamento e escopo durante o processo de desenvolvimento. De uma maneira geral, as metodologias tradicionais são mais "pesadas" e devem ser utilizadas em situações em que os requisitos do sistema são estáveis. Por outro lado, em projetos nos quais os requisitos são passíveis de alterações e com equipes reduzidas, utilizar propostas mais ágeis tem se mostrado fundamental.

Como vimos, o processo de desenvolvimento das metodologias tradicionais baseia-se em um conjunto de atividades predefinidas, que invariavelmente se iniciam com o levantamento completo de um conjunto de requisitos, seguido por um projeto de alto nível e, consequentemente, de uma implementação, validação, até chegar à manutenção. Outro aspecto característico está relacionado com as atividades existentes nas metodologias tradicionais, ligadas a processos orientados a documentação, os quais, de certa forma, são considerados fatores limitadores aos desenvolvedores, pois muitas organizações não possuem recursos ou inclinação para processos burocráticos. Esta é uma das principais razões que levam organizações de pequeno porte a não usar nenhum tipo de

processo, o que pode proporcionar resultados indesejados na qualidade do produto desenvolvido, dificultando a entrega nos prazos acordados e adequação aos custos predefinidos.

Considerando as diferenças significativas entre as metodologias tradicionais e ágeis, este capítulo tem por objetivo realizar algumas comparações entre elas, além de dar uma visão sobre testes de software e aspectos da qualidade, levando em conta o uso de metodologias ágeis.

13.2 Diferenças entre o tradicional e o ágil

Existem vários aspectos responsáveis por diferenciar as metodologias tradicionais das ágeis, os quais podem ser observados em regras que explicam como produzir e conduzir processos, na forma peculiar de desenvolver software existente no ambiente de desenvolvimento e na maneira de pensar dos integrantes da equipe. Talvez esta última seja, de fato, a principal diferença que podemos observar, pois pensar diferente implica a adoção de princípios de desenvolvimento diferentes.

Outra característica importante das metodologias ágeis é que elas são preparadas para aceitar mudanças durante o processo de desenvolvimento de software, ao passo que as metodologias tradicionais são resistentes a elas. Metodologias ágeis são baseadas em dados estatísticos obtidos de históricos referentes à implementação do código. Já os métodos tradicionais utilizam como base normas que definem padrões a serem seguidos. A estratégia de trabalho adotada é imposta pela própria equipe de desenvolvimento, não exigindo um grande controle dos processos, o que é bem diferente do observado nas metodologias tradicionais, uma vez que percebemos claramente a imposição mais acirrada, por meio de inúmeras normas e políticas que devem ser respeitadas.

Quando pensamos em aspectos contratuais, diferentemente do que ocorre com as metodologias tradicionais, que são baseadas em rígidos contratos, as metodologias ágeis não pregam o uso de contratos, pois essa postura reforça o princípio de que as mudanças são bem-vindas. Se o contrato tiver de ser estabelecido, que seja o mais flexível possível, de forma a resguardar os interesses da equipe de desenvolvimento e do cliente.

A participação dos clientes também é diferenciada. Nas metodologias tradicionais, mesmo podendo participar de reuniões, não possuem poder de decisão no que diz respeito à forma de desenvolvimento do projeto. Nas metodologias ágeis, o cliente faz parte da equipe de desenvolvimento, podendo auxiliar e indicar o rumo que a implementação pode tomar.

Finalmente, podemos citar o reduzido tamanho das equipes de desenvolvimento de software que utilizam paradigmas ágeis, em torno de dez indivíduos que trabalham em um mesmo local. Projetos que usam metodologias tradicionais geralmente possuem uma grande quantidade de integrantes na equipe, muitas vezes geograficamente dispersos.

Percebemos também diferenças quando analisamos o custo gerado com mudanças ao longo do desenvolvimento do software. Considerando seus princípios, as metodologias ágeis já estão preparadas para aceitar mudanças no projeto, pois estão focadas nas pessoas e não nos processos, e por natureza denotam um controle rígido deles. Nas metodologias tradicionais, à medida que alterações são necessárias na fase próxima de seu encerramento, seu custo tende a crescer exponencialmente. Nas metodologias ágeis o custo não cresce ao final, mesmo que alterações de requisitos devam ser realizadas.

Um fator que contribui para que o custo se mantenha estável nas metodologias ágeis é que elas não estão amarradas a contratos ou documentações, além de sua iteratividade, que possibilita desenvolver o sistema incrementalmente. É importante lembrar que, mesmo considerando as vantagens das metodologias ágeis no desenvolvimento de software, seu uso sempre deve estar vinculado ao contexto do software que será desenvolvido. Certamente softwares cuja precisão e confiabilidade são fatores de decisão e risco, como ocorre com sistemas cirúrgicos ou aeroviários, não podem dispensar as recomendações implícitas características das metodologias tradicionais, principalmente quanto à necessidade de documentar o que deve ser feito.

13.3 Estudo comparativo

A Tabela 13.1 apresenta um quadro comparativo que traz uma visão geral das principais características das metodologias ágeis.

Metodologia	Criado por	Ano	Requisitos	Iterações	Incremental	Validação e teste	Diagramas da UML
XP	Kent Back	1996	Clientes escrevem	1-4 semanas com participação do cliente	Programação em duplas com *refactoring*	Teste de unidade e aceitação	Classes, caso de uso, sequência ou atividades
SCRUM	Jeff Sutherland, Ken Schwaber e Mike Beedle	1995	*Product Backlog*	*Sprint* de 30 dias	*Sprint* Backlog	---	---
FDD	Jeff de Luca e Peter Coad	1997	Artefato ficha	2 semanas por prioridade	Refinamento	Inspeções pelos programadores	Classes, caso de uso, sequência e atividades
ASD	Sam Bayer e James Highsmith	1997	Sessões	4 a 8 semanas por quantidade de requisitos	Implementação das quantidades de requisitos	Testes funcionais	Classes, caso de uso, sequência e atividades
CRYSTAL	Alistair Cockburn	1998	Entrevista, *briefing* e questionários	2 semanas	*Release* em sequência	Testes de regressão	Classes, caso de uso, sequência e atividades
ICONIX	Doug Rosenberg, Matt Stephens e Mark Colling Cope	1999	Entrevista, questionários e atas (rastreáveis)	2 semanas por prioridade	*Releases* em sequência	Testes funcionais, aceitação e de unidade	Classes, caso de uso, sequência ou atividades e robustez
DSDM	Consórcio de empresas associadas	1994	Workshop e estudo de viabilidade	2 semanas MOSCOW	*Timeboxing*	Testes de regressão	Classes, caso de uso, sequência e atividades
APSO	José Henrique T. C. Sbrocco e Paulo Cesar de Macedo	2011	Documento de visão, *Product Backlog*	1 semana (por orientação)	*Sprint Backlog*	Inspeções e testes funcionais	Classes, caso de uso, sequência ou atividades

Tabela 13.1 - Comparação das principais características das metodologias ágeis.

13.4 Testes de software X metodologias ágeis

Vimos que as metodologias de desenvolvimento ágeis se destacam dos processos de desenvolvimento tradicionais principalmente pelo fato de priorizarem a implementação de códigos executáveis em vez de documentação escrita. Sendo assim, torna-se relevante refletir sobre como garantir que requisitos funcionais e não funcionais estão sendo atendidos, considerando a falta de amparo de artefatos documentais.

Tradicionalmente, procuramos garantir a fidelidade do que está sendo desenvolvido em relação aos requisitos pela execução de testes. Essencialmente, os requisitos necessários para realizarmos testes quando utilizamos paradigmas ágeis não são muito diferentes dos métodos de desenvolvimento tradicionais. A grande diferença está no grau de importância que conferimos aos testes, considerando cada uma das abordagens. Nas metodologias tradicionais, percebemos a existência de uma extensa lista de artefatos documentais resultantes da análise do projeto que será desenvolvido. Neste contexto, os testes são vistos apenas como mais um artefato produzido pelo processo, que obviamente visa garantir a detecção de erros antes da liberação do produto. Como observamos anteriormente, nos processos ágeis não existe muito amparo documental, e os testes, neste caso, passam a ter um papel importantíssimo para o sucesso da metodologia ágil utilizada.

Uma das primeiras questões que pode surgir quando temos de definir estratégias de teste de software está relacionada ao uso de testes manuais ou automatizados. Os testes manuais, como o próprio nome sugere, são realizados pelos desenvolvedores ou pelo usuário final, com o objetivo de verificar anomalias quanto ao que havia sido combinado inicialmente. O problema do teste manual é que dificilmente alguém consegue testar exaustivamente um sistema de forma que não reste nenhuma falha, pois sempre haverá a possibilidade de esquecer algum detalhe ou operação que pode gerar um erro. O aspecto positivo desse tipo de teste é que, mesmo considerando tal fragilidade, é fácil de aplicar, uma vez que basta utilizar o sistema como se estivesse em produção.

Os testes automatizados utilizam aplicativos específicos capazes de realizar testes exaustivos por meio de scripts de testes pré-configurados. Mesmo abrangendo uma quantidade bem maior de situações que um teste manual, não devemos considerar que proporcionam 100% de cobertura, pois existem falhas lógicas que só ocorrem por combinações específicas de entrada de dados. Um aspecto que dificulta o uso dessas ferramentas é o fato de normalmente precisarem de um profissional especializado que, além do domínio da ferramenta, deve possuir qualificação em testes. Essas características têm complicado o uso de testes automatizados.

Independentemente da estratégia utilizada, testes de software são considerados essenciais nas metodologias ágeis, pois estão diretamente relacionados com a garantia da qualidade do produto.

Resumidamente, podemos observar que nas metodologias ágeis ocorrem os testes de unidade e também os testes de aceitação, pelo fato de contar com a participação do cliente durante o processo de desenvolvimento. É curioso notar que, embora os testes representem grande parcela do alicerce que sustenta as metodologias ágeis, pouca ou nenhuma menção a atividades de teste é encontrada, com exceção do método XP, que explicitamente enfatiza essas atividades.

13.4.1 Considerações gerais sobre testes de software

Do ponto de vista prático, os programadores que não usavam nenhuma metodologia pouco se importavam com essa fase quando implementavam seus projetos. Desta forma, deixavam que os problemas surgissem no momento da utilização e corrigiam aos poucos. Quase sempre deixavam um legado de erros sem correção ou utilizavam remendos para acobertá-los. Testar o software significa verificar o comportamento do sistema usando estímulos e entradas em busca de erros, verificar se o sistema foi feito dentro dos padrões preestabelecidos, além de analisar se os resultados de sua utilização correspondem aos processos e ao esperado. Partindo desse conceito, testar indica aumentar os custos, ou seja, executar os processos de verificação e validação por meio de testes leva tempo e pode envolver vários profissionais.

Os testes de software estão divididos em duas categorias básicas, sendo testes funcionais e testes estruturais. Ambos podem ser automatizados, permitindo que sejam previamente criados para testar alguns tipos de interações. A seguir, as Tabelas 13.2 e 13.3 apresentam os tipos de testes e suas implicações.

Testes funcionais	Descrição do teste	Quando executar
Tratamento de erros	Verifica se o sistema valida todas as transações e se retorna todas as mensagens de erro (*feedback*) no caso de receber informações erradas.	Em todo o ciclo de desenvolvimento
Requisitos	Verifica se o sistema executa todas as funcionalidades conforme o elicitado no documento de requisitos.	A cada iteração
Regressão	Testa o impacto de novas funcionalidades sobre as já existentes e deve também ser realizada na documentação, principalmente se houver alterações.	A cada iteração
Suporte manual	Testa o sistema de ajuda (help) sensível ao contexto e verifica se a documentação está completa e devidamente atualizada.	Na homologação (entrega do sistema)
Controle	Testa se o processamento corresponde ao esperado, principalmente em procedimentos alheios à aplicação; por exemplo: backups e recuperação de dados.	Pode ser executado a qualquer momento
Interconexão	Garante a comunicação dos módulos desenvolvidos, bem como sua integração com outros sistemas (se houver).	Pode ser executado a qualquer momento
Paralelo	Testa paralelamente o sistema em comparação ao sistema antigo, ou seja, verifica se atende as mesmas especificações, comparando resultados.	A cada iteração

Tabela 13.2 – Testes funcionais e suas características.

Testes estruturais	Descrição do teste	Quando executar
Unidade	Testa as funções, classes e objetos do sistema, considerando a performance e a lógica utilizada.	Em todo o ciclo de desenvolvimento
Execução	Analisa o desempenho do sistema com dados reais, testando a performance com múltiplos acessos simultaneamente.	No início e na homologação
Estresse	Testa os limites máximo e mínimo do sistema, a fim de avaliar seu comportamento em condições adversas.	A cada iteração e na homologação
Recuperação	Testa como um sistema pode recuperar-se de problemas físicos ou lógicos, desde falhas elétricas até de componentes de hardware e rede (contingência).	Na homologação ou a qualquer momento
Operação	Avalia o funcionamento do sistema, comparando com os processos manuais da empresa.	A cada iteração
Conformidade	Verifica se o sistema foi feito de acordo com as normas e os padrões previamente estabelecidos (*patterns*).	Em todo o ciclo de desenvolvimento
Segurança	Teste de confiabilidade que assegura se o sistema está preparado para impedir acessos não autorizados ou invasões, protegendo seus dados quanto a isso.	Na homologação ou a qualquer momento
Integração	Realizado pelo analista, conferindo e validando o que foi proposto nos casos de uso e o que foi projetado.	A cada iteração
Sistemas	Testa todo o sistema, utilizando cenários preestabelecidos a fim de confrontar resultados.	Na homologação ou a qualquer momento
Aceitação	Testa a validação final do sistema com o cliente, liberando para o usuário final.	Na homologação

Tabela 13.3 – Testes estruturais e suas características.

As metodologias ágeis utilizam esses mecanismos de testes em suas práticas de desenvolvimento. Algumas optam por mais de um tipo; quando combinados, geram resultados satisfatórios e garantem um software de qualidade. Um bom teste de software, independente do tipo, deve ser planejado e devidamente documentado. Recomenda-se criar um plano contendo todos os testes realizados, bem como seu detalhamento. A seguir, algumas recomendações e exemplos de utilização dos testes de software.

13.4.2 Plano de testes

Um documento que descreve o funcionamento e os responsáveis pelos testes no sistema, além de todos os dados das funcionalidades ou sistema a ser testado. A Tabela 13.4 mostra um modelo dos dados necessários para realizar o plano.

Plano de testes			
Projeto:			
Data:		Versão revisada:	
Responsáveis:			
Testes a serem realizados			
1. Teste de aceitação		Responsável:	<nome responsável>
2. Teste de regressão		Responsável:	<nome responsável>
...		...	

Tabela 13.4 - Exemplo de plano de testes.

Para cada tipo de teste listado no plano de testes deve-se providenciar um documento, contendo um detalhamento do teste realizado e seus resultados. A Tabela 13.5 apresenta um exemplo do teste de aceitação.

Teste de aceitação		
Projeto:		
Data realização:	<data>	Responsável: <nome do analista ou coordenador>
Objetivos do teste		
Testar e validar a versão final do sistema com o cliente, liberando para o usuário final. Verificar indicadores de boa usabilidade da versão final.		
Escopo do teste:	<colocar aqui os detalhes de como vai acontecer o teste>	
Histórico:	<data>	<responsável>
	<data>	<responsável>
Itens a serem testados		
ID	Descrição	
1		
2		
3		
4		
5		
...		
Itens que não serão testados		
ID	Descrição	
1		
2		
3		
...		

Critérios utilizados	
<incluir uma breve análise dos riscos>	
Resultados dos testes	
ID	Descrição dos resultados
Aprovado por:	<nome> <data> e <assinatura>

Tabela 13.5 - Exemplo utilizado para teste de aceitação.

Vale a pena frisar que para cada tipo de teste deve haver um documento com seus detalhes. O exemplo apresentado é o conteúdo mínimo a ser preenchido.

Um teste que a maioria das metodologias ágeis utiliza é o teste de unidade que, como citado anteriormente, garante que funcionalidades, objetos e classes trabalhem com uma lógica livre de erros. Uma das formas de realizar esse teste é criar uma implementação automatizada que, por meio de alguns procedimentos, consegue aferir se o sistema está realmente trabalhando corretamente. Tomamos como exemplo uma operação/funcionalidade do sistema que calcula o parcelamento de uma dívida:

Procedimento Gera_parcelas (NumP: inteiro, DataAtual: data e hora, Valor: moeda, TabJuros: inteiro)

São passados como parâmetros o número de parcelas (NumP), a data atual (DataAtual) e a tabela de juros utilizada (TabJuros). Em seguida, assim que o método recebe esses parâmetros, calcula as parcelas e as datas de vencimento de acordo com a tabela de juros indicada. Um teste unitário cria no ambiente de desenvolvimento ou anexo a ele (como no caso de frameworks modernos) um método que recebe os dados reais e calcula os resultados para comparar com os do sistema, por exemplo:

Procedimento TestaParcela( )

Datahoje=05/10/2011;

Valorhoje=R$300,00;

TabelaJuro=1;

NumeroParcelas=3;

Parcela30=05/11/2011;

Parcela60=05/12/2011;

Parcela90=05/01/2012;

Valor30=R$306,75;

Valor60=R$312,88;

Valor90=R$337,91;

Gera_parcelas(NumeroParcelas,Datahoje,Valorhoje,TabelaJuro)

Se (Parcela30=Parcela30_obtida). E . (Parcela60=Parcela60_obtida) . E . (Parcela90=Parcela90_obtida) então "Datas testadas estão OK" senão "Problemas nas datas"

Se (Valor30=Valor30_obtido). E . (Valor60=Valor60_obtido) . E . (Valor90=Valor90_obtido) então "Valores testados estão OK"senão "Problemas nos valores"

Os dados resultantes devem ser comparados com os dados do sistema a fim de localizar um possível erro. Obviamente este foi apenas um exemplo, podendo ter muito mais detalhes em uma implementação real. O que deve ser entendido neste momento é a possibilidade de usar um software para testar outro e comparar os resultados.

13.4.3 Casos de teste

Para cada funcionalidade a ser testada é necessária a criação de casos de testes, os quais devem acompanhar cada caso de uso da modelagem a fim de checar seus resultados. A Tabela 13.6 apresenta um exemplo do que deve conter um caso de teste.

Casos de teste		
ID:	<nome do caso>	
Itens a serem testados		
1	<descrição do item>	
2	<descrição do item>	
...		
Entradas	**Campo**	**Valor**
Saídas	**Campo**	**Valor**
Procedimento/função - método (sistema)		**Procedimento (teste)**
Aprovação:		Data:

Tabela 13.6 - Exemplo de caso de teste.

13.4.4 Considerações sobre testes de software nas metodologias SCRUM e XP

13.4.4.1 Testes de software na metodologia XP

A eXtreme Programming adota técnicas de teste conhecidas como *Test-First Design* ou *Test-Last*. Como proposta do *Test-First*, qualquer método de um objeto que possa falhar deve ter um teste que garanta seu funcionamento. Para isso, precisamos ter bom-senso ao definirmos que métodos devem ser testados, pois não podemos esquecer os mais importantes nem perder tempo testando métodos extremamente simples, em que não há como ocorrerem erros.

Na metodologia XP, os testes unitários fazem parte da modelagem do sistema, uma vez que tanto as funcionalidades quanto a documentação de um objeto são expressas por seus testes. Essa estratégia ajuda a diminuir a documentação na XP, pois sempre que alguém precisar saber a função de determinado objeto no contexto do sistema, basta procurar nos testes unitários, que estará registrada. Na XP, quando os programadores recebem a história que deve ser implementada, passam a escrever os testes unitários para ela. Por isso essa técnica é denominada *Test-First Design*.

A aplicação de testes automatizados é considerada fundamental na metodologia XP, pois eles dão suporte a outras técnicas utilizadas nessa metodologia, como o *refactoring* e o *colletive code ownership*. Várias linguagens de programação oferecem *frameworks* de testes automatizados. Por exemplo, no Java temos o *jUnit*, desenvolvido por Erich Gamma e Kent Beck para implementar testes de unidades, estruturando-os e executando-os automaticamente. O *jUnit* baseia-se na criação de uma classe de teste para cada classe a ser implementada. Cada classe de teste criada estende a classe *TestCase* do *jUnit*, e cada teste que deve ser feito será um método dessa classe de teste. Desta forma, a classe de teste *TestCase* provê uma série de métodos que permitirão confrontar os resultados obtidos com os esperados, identificando se o teste falhou ou não.

13.4.4.2 Testes de software na metodologia SCRUM

Como vimos, o SCRUM utiliza recursos claros para comunicação, como, por exemplo, o *task board* (quadro kanban). Esse instrumento permite que as tarefas e os objetivos fiquem mais claros para todos os membros da equipe.

Também existe uma grande ênfase a reuniões periódicas de alinhamento, que possuem objetivos específicos. Como exemplo podemos citar a *daily meeting*, que compartilha com todos os membros da equipe a situação atual do projeto e o andamento dos trabalhos. Essa reunião diária, cuja simplicidade permite à equipe focar-se no trabalho que realmente deve ser feito, pode contribuir para melhorar o gerenciamento dos testes quando comparado com metodologias tradicionais.

É bom lembrar que, como o SCRUM é um *framework* direcionado a boas práticas de gestão, é preciso que junto com essa metodologia também sejam adotadas técnicas complementares que contribuem para que as entregas atendam sempre aos padrões de qualidade.

13.4.4.3 Considerações complementares

É indiscutível a importância dos testes nas metodologias ágeis de desenvolvimento. Torna-se inviável entregar um software sem nenhum tipo de teste. Os testes concretizam a ideia de que prevenir é melhor que remediar, ou seja, o tempo gasto testando os componentes de software é economizado após a entrega das funcionalidades. Ainda existem algumas dúvidas e discussões sobre o papel que um analista de testes deve desempenhar ao utilizar uma metodologia ágil. Alguns chegam a achar que não existe espaço para um profissional com essa especialização em uma equipe que utiliza paradigmas ágeis. É bom lembrar que no processo tradicional os testes aparecem no fim do ciclo de desenvolvimento. Neste caso, o analista de teste é mais reativo. Quando esse profissional utiliza métodos ágeis, ele deixa de ser reativo para ocupar um papel fundamental no que diz respeito à interação dos desenvolvedores e demais integrantes do projeto.

13.5 Qualidade de software: é possível mantê-la usando uma metodologia ágil?

A indústria de software ainda enfrenta uma série de desafios relacionados a cumprimento de prazos, redução de custos e obtenção de qualidade, o que levou à necessidade de utilizar e estabelecer normas e modelos de qualidade. Essas recomendações são guias que orientam como aumentar, medir e garantir a qualidade de software, como, por exemplo, as normas ISO/IEC 12207 e ISO 15504, e o modelo CMMI (*Capability Maturity Model Integration*). No Brasil foi lançado, em 2003, o modelo de qualidade MPS.BR (Melhoria do Processo de Software Brasileiro), que também tem o objetivo de guiar a implantação e o uso de boas práticas da engenharia de software, considerando as características das empresas de software brasileiras, em especial as de pequeno e médio porte. É importante observar que todas essas recomendações orientam "o que fazer" para atingirmos qualidade no desenvolvimento de software, mas não explicam "como fazer". Essa característica fornece à empresa liberdade de escolha sobre o processo de desenvolvimento que será adotado.

Erros durante o desenvolvimento de um software sempre irão ocorrer, independente do paradigma adotado. Isso ocorre pelo fato de haver interação com pessoas e também ferramentas, que podem estar com *bugs* (erros no software), entre outras razões. Assim, considerando que as metodologias ágeis não são orientadas a documentação (justamente para colaborar com a desburocratização do processo de desenvolvimento), é comum refletirmos sobre a adequação de seu uso quando decidimos utilizar um modelo de qualidade.

Na engenharia de software, o custo gerado por um erro tende a aumentar na medida em que esse erro se propaga pelo ciclo de vida do software. Desta forma, quanto mais tarde se descobrir um erro, maior será o prejuízo. Quando pensamos em qualidade de software, devemos deixar claro que ela é obtida quando percebemos que o produto desenvolvido está em conformidade com os requisitos funcionais e não funcionais. Portanto, a ocorrência de não conformidades com os requisitos representa a falta de qualidade.

Também existem diversas perspectivas sob as quais podemos analisar a qualidade do software. Sob o ponto de vista do usuário final, software com qualidade é aquele que atende suas necessidades, mostrando-se fácil de operar, além de demonstrar ser confiável. Já para um desenvolvedor, software com qualidade é aquele que permite uma manutenção rápida para atender as necessidades

do cliente. No caso do cliente, a qualidade do software pode ser medida pelo valor que ele agrega à organização e pelo seu ROI (*Return of Investment*).

Certamente podemos incluir outros elementos a serem considerados para avaliar a qualidade de um software, como, por exemplo, o software ter ou não sido entregue no prazo e os custos previstos. De uma maneira resumida, podemos dizer que um fator decisivo para o sucesso de qualquer projeto de software é garantir um mínimo de qualidade, o que normalmente está diretamente relacionado com os investimentos feitos em prol do controle de qualidade, normalmente divididos entre atividades de prevenção de erros, avaliação e identificação de falhas.

13.6 Exercícios

1) Por que as metodologias ágeis têm sido apontadas como uma alternativa às abordagens tradicionais?

2) Quais são as principais diferenças entre métodos tradicionais e ágeis?

3) Quais são as diferenças relacionadas à participação do cliente nas metodologias ágeis e tradicionais?

4) Costuma-se dizer que o custo do projeto se mantém estável quando utilizamos metodologias ágeis. Isso se deve a quê?

5) Cite um aspecto positivo e outro negativo dos testes de software manuais.

6) Quais os tipos de teste comumente encontrados nas metodologias ágeis? Eles são comentados nas documentações relacionadas às metodologias que estão disponíveis?

7) Qual é o tipo de teste utilizado pela maioria das metodologias ágeis? Como podemos realizar esse teste?

8) Comente a aplicação de testes automatizados na metodologia XP.

14

Escolha da Metodologia Considerando as Características do Projeto

Objetivos

- *Apresentar critérios para escolha da metodologia ágil que melhor se ajuste às necessidades do projeto.*
- *Sugerir ferramentas de apoio nas diferentes fases de um sistema.*

14.1 Aspectos motivacionais

Segundo Abraham Maslow, famoso psicólogo americano, a motivação das pessoas é formada pelo conjunto de cinco necessidades básicas: fisiológicas, segurança, amor, estima e autoestima. Ainda segundo ele, as necessidades fisiológicas estão relacionadas com alimentação, moradia ou preocupação com a saúde. A segurança envolve a integridade física e também a estabilidade no emprego, uma vez que a preocupação com a remuneração também está ligada à segurança, pois garante a subsistência da pessoa. O amor, a estima e a autoestima também são importantes de serem observados, pois no contexto corporativo estão relacionados com a capacidade de realização e de reconhecimento dos demais companheiros de trabalho.

Conforme esses aspectos motivacionais, as organizações devem propor políticas pessoais e de cargos e salários, considerando preferencialmente a aquisição de conhecimento, objetivando aumentar a autoestima e o reconhecimento por parte dos demais colaboradores.

Giovanni Asproni, membro da *Agile Alliance*, destaca dez fatores motivacionais para profissionais da área de TI, em ordem crescente de importância: realização, possibilidade de crescimento, o trabalho em si, reconhecimento, avanço nos conhecimentos, supervisão técnica, responsabilidade, relacionamento com os pares, relacionamento com subordinados e salário.

Como sabemos, os métodos ágeis têm como essência básica o trabalho em equipe. Portanto, correlacionar motivação pessoal ao trabalho em equipe é fator básico de sucesso, pois colabora com o clima de organização da equipe, estimula uma liderança íntegra e a adoção de padrões de excelência e reconhecimento. As metodologias ágeis contribuem naturalmente para aumentar a produtividade dos desenvolvedores, em função do aumento da motivação dos integrantes do time. Isso ocorre pelos evidentes fatores que concebem autonomia, *feedback* e condições de completude rápida de tarefas, que contribuem para o aumento da satisfação e motivação dos integrantes do time. É importante compreender que as organizações não devem se limitar a aprender as técnicas propostas pelas metodologias ágeis que utilizam, mas também incorporar aspectos que as tornem reconhecidas como organizações ágeis e que aprendem.

14.2 Características do projeto

O uso das metodologias ágeis deve ser avaliado por cada organização, levando em conta suas vantagens e desvantagens, além do tipo de projeto em que a metodologia será aplicada. Considerando que os projetos possuem particularidades, na prática temos observado que o uso de todos os aspectos de uma única metodologia ágil, sem a utilização de contribuições advindas de outras metodologias, tem se mostrado raro. Também observamos o mesmo nas metodologias mais tradicionais, como o uso das boas práticas propostas pelo PMBOK, sendo difícil encontrar uma organização que siga dogmaticamente todos os seus preceitos.

É necessário, portanto, que realizemos uma avaliação criteriosa dos processos propostos pelas metodologias candidatas, objetivando criticar o uso desses processos pelo projeto, sempre tendo em mente a otimização de tempo, esforço e, principalmente, a garantia de atendimento às expectativas dos clientes.

Existe um consenso geral relacionado à necessidade de os processos de software se adaptarem a modificações de última hora, visando atender demandas específicas do cliente, considerando o mercado cada vez mais competitivo e dinâmico. Essa necessidade se justifica em função da imprevisibilidade dos sistemas de software, além da constatação de variadas habilidades dos desenvolvedores e de que documentos e planos ficam rapidamente desatualizados. O desempenho de qualquer metodologia adotada depende do ambiente em que será aplicada, das características do projeto, da maneira como será acolhida pela equipe de desenvolvimento, entre outros fatores.

14.3 Aplicabilidade das metodologias ágeis

Como foi apresentado nesta obra, as metodologias ágeis possuem diferenças entre si, embora compartilhem inúmeras características em comum, como, por exemplo, o desenvolvimento iterativo e o foco na comunicação. Podemos considerar que a aplicação dos métodos ágeis pode ser vista sob diversas perspectivas:

- **Perspectiva do produto:** a utilização de metodologias ágeis mostra-se mais adequada quando os requisitos ainda estão sendo elaborados e mudam rápido.
- **Perspectiva organizacional:** a aplicação desses paradigmas pode ser definida considerando três aspectos da organização, sendo a cultura, as pessoas e a comunicação. Aspectos culturais dizem respeito ao fato de refletirmos sobre o apoio que a organização oferece

durante a negociação do projeto. A organização deve ser responsável pela promoção das decisões tomadas pelos desenvolvedores. As pessoas também são importantes, devendo demonstrar confiança. A equipe pode ser composta de poucas pessoas, mas todas competentes. Com relação à comunicação, é importante que a organização tenha um ambiente que facilite e estimule uma rápida comunicação entre seus integrantes.

Além dessas perspectivas, também devemos considerar outros fatores para a escolha das metodologias ágeis. Um fator importante que sempre deve ser observado é o tamanho do projeto, pois sabemos que à medida que o projeto aumenta, a comunicação face a face, tão estimulada pelos paradigmas ágeis, torna-se mais difícil. Considerando esse aspecto, chegamos a uma conclusão importante: os métodos ágeis mostram-se mais adequados para projetos que envolvem um grupo reduzido de integrantes (até 40 pessoas).

Apesar de ser possível determinar a aplicação de metodologias ágeis pela análise das perspectivas mencionadas, é necessário utilizar análises mais sofisticadas quando optamos pelo uso de métodos ágeis específicos. Como exemplo, podemos citar a família de métodos Crystal, que provê caracterizações relacionadas à seleção de métodos para um projeto, baseando-se no tamanho do projeto, sua criticidade e prioridade. Também podemos observar na metodologia DSDM o chamado "filtro de aplicabilidade", utilizado para esse propósito.

Apesar dos exemplos citados, a maior parte das metodologias ágeis não apresenta recomendações e instrumentos específicos que ajudam a definir sua aplicabilidade a um projeto. Talvez isso ocorra porque, nas diretrizes de alguns métodos ágeis, como o DSDM ou o FDD, sua aplicabilidade seja direcionada para qualquer projeto de desenvolvimento, independente de suas características. De uma maneira geral, podemos usar o bom-senso e analisar as características de cada metodologia para determinar critérios de seleção relacionados à sua aplicabilidade.

Sabemos que o uso de metodologias ágeis tem sido amplamente documentado como exemplo de sucesso, principalmente considerando equipes pequenas (com menos de dez desenvolvedores). Também existem registros das boas contribuições desses paradigmas em cenários de constantes mudanças de requisitos. Contudo, a aplicabilidade dessas metodologias em alguns cenários específicos ainda gera discussões e falta de consenso por parte de quem as utiliza. Embora já tenham sido descritas como casos de sucesso, podemos citar como exemplo o polêmico uso de metodologias ágeis em larga escala, considerando times com mais de vinte integrantes. Também existem resistências ao uso desses paradigmas autogerenciáveis por organizações que possuem cultura de comando e controle de seus processos.

14.4 Métodos de escolha

Um dos métodos mais citados para escolha da metodologia ágil adequada às necessidades da organização é aquele baseado na análise de risco inerente ao uso da metodologia ágil para determinado projeto. Esse método consiste em diversos passos, resumidos a seguir:

1) Como primeira base para um critério de escolha, a análise de risco deve ser aplicada a áreas específicas que estão associadas à metodologia escolhida.

2) Com base no passo 1, verificar se trata-se de um projeto que pode utilizar metodologias ágeis ou não. Se for, ir para o item 4.

3) Se estivermos neste item, é porque o projeto não se mostrou adequado ao uso de metodologias ágeis, o que indica a existência de partes do projeto com valores de risco diferentes. Portanto, a equipe de desenvolvimento pode, caso seja possível, adaptar uma arquitetura de projeto que permita o uso de métodos ágeis ou utilizar uma metodologia tradicional.

4) Neste passo devemos definir as soluções para os riscos mapeados e iniciar o design do projeto.

Observe que esta técnica não está limitada à escolha de um método ágil ou outro, pois na maioria das organizações escolhemos as metodologias pelas vantagens e desvantagens que oferecem. Normalmente utilizamos a abordagem baseada na análise de riscos quando desejamos integrar aspectos de determinadas metodologias ágeis a processos tradicionais.

A aplicabilidade de uma metodologia ágil pode ser analisada sob vários aspectos. Entre eles, o que mais se destaca é considerar que requisitos que mudam constantemente ou surgem no meio de uma análise prejudicam a condução do projeto por metodologias tradicionais, ou seja, fica inviável planejar o desconhecido. Muitas vezes, ao utilizarmos metodologias tradicionais, alguns projetos de software acabam estourando prazos e custos justamente pelo fato de novos requisitos surgirem no meio da implementação, conforme apresentamos anteriormente. Em metodologias tradicionais, novos requisitos implicam novos projetos de sistemas, por isso devem ser planejados novamente, gerando outros prazos e, consequentemente, custos.

Seguindo esse raciocínio, fica óbvio que, ao nos depararmos com projetos cujas características de requisitos são "mutantes", uma boa estratégia seria a escolha de metodologias ágeis. Mas, como sabemos, nem tudo são "flores". Os critérios de escolha de uma metodologia ágil incluem variáveis e dimensões, como equipes e ferramentas de apoio (demonstradas a seguir), que devem ser consideradas fundamentais para seu funcionamento.

14.4.1 Equipes

Embora as metodologias ágeis abordem em suas características a possibilidade de atuarem em grandes projetos de sistemas e com grandes equipes de desenvolvedores, como citado anteriormente, às vezes os melhores resultados são obtidos com times pequenos, os quais devem possuir algumas qualidades, tais como:

- A equipe deve ser coesa em suas atitudes e decisões;
- Os membros da equipe devem ser confiantes e ter comprometimento ("vestir a camisa");
- Pessoas competentes mesmo que em pequeno número;
- Alto grau de comunicabilidade.

Em linhas gerais, consegue-se uma equipe de qualidade por meio de experiências em desenvolvimento e em vários tipos de projeto ao longo de um período. Convém lembrar que existe uma maneira de treinar os indivíduos conhecida como PSP (*Personal Software Process*), que utiliza um conjunto de *scripts* (roteiros) para melhorar o desempenho de um indivíduo com relação a projetos de sistemas. A PSP foi criada em 2000 por Watts S. Humphrey (um dos criadores do CMM) e disponibilizada em um livro de sua autoria (*Introduction to the Personal Software Process*).

Essa ferramenta possibilita ao desenvolvedor, entre outras coisas, conhecer-se, medir-se, analisar-se e aprender a conduzir seus projetos. Com o uso de um conjunto de métricas, é possível identificar onde o desenvolvedor está com mais dificuldade, facilitando o diagnóstico do que deve ser trabalhado. A PSP ajuda o desenvolvedor a produzir melhor e com muito mais qualidade, por meio de níveis, a saber:

- Nível 0 = Fundamentos de medidas e formatos de relatórios
- Nível 1 = Planejamento e estimativas de tamanho e tempo
- Nível 2 = Controle pessoal de qualidade de projeto
- Nível 3 = Extensão a projetos maiores

O profissional desenvolvedor deve atender a um determinado conjunto de requisitos para passar ao outro nível imediatamente acima.

Nível 0: nesse nível o desenvolvedor começa registrando seus tempos de codificação, os erros encontrados, os defeitos e descreve seu padrão de codificação, o qual deve obedecer às regras dos padrões existentes da engenharia de software (*Design Patterns*). Após registrar essas informações em formulários apropriados, conforme a Figura 14.1, deve ainda propor um melhoramento de todo o processo.

PSP - Log de Defeitos Diário				
Desenvolvedor			Data Início	
Projeto				
Data	**ID Requisito**	**Descrição do Erro/Defeito**	**Tipos**	**Tempo Conserto/h**
			Documentação Syntax Build Interface Função Definições	

Tabela 14.1 – Modelo de log de defeitos do PSP.

Nível 1: uma vez que as medidas de desempenho já foram definidas, esse nível ajusta o tamanho e o tempo gasto para desenvolver um determinado projeto. A ideia é ajustar o desenvolvimento e poder assumir as entregas de software com mais certeza, organizando e dividindo melhor os trabalhos. Nesse nível também é possível realizar os relatórios de testes de cada fase, independente da metodologia utilizada, além de estabelecer um cronograma de desenvolvimento.

Nível 2: nesse nível o indivíduo aprende a lidar com seus erros, utilizando uma lista de verificação (checklist) para revisar seus códigos. Com isso é possível ajustar seu nível de evolução com relação à qualidade durante o desenvolvimento.

Nível 3: este é o último estágio da PSP e recomenda a divisão de projetos grandes em partes menores para poderem ser tratadas pelo nível 2. Assim, o desenvolvimento passa a ser incremental sob o ponto de vista do desenvolvedor.

No caso de uma equipe de desenvolvimento que já possui técnicas e altos níveis de PSP, pode-se utilizar outro conjunto de roteiros conhecido como TSP. Diferentemente da PSP, a TSP (*Team Software Process*) aborda uma melhoria contínua no desempenho de uma equipe de projeto. Um projeto de software conduzido por regras da TSP, entre outras qualidades, requer:

a) que os desenvolvedores estabeleçam um conjunto de metas com que todos concordem;

b) que os papéis de cada um sejam preestabelecidos;

c) que o ambiente de trabalho seja adequado;

d) que um processo de trabalho seja comum a todos.

Além disso, a TSP exige que os desenvolvedores cumpram suas metas e mantenham o respeito à comunicação aberta entre todos os membros do time.

14.4.2 Ferramentas de apoio

Durante o desenvolvimento de um projeto de software, a equipe pode utilizar ferramentas (conhecidas como CASE) para agilizar e documentar suas iterações em todas as fases de desenvolvimento, podendo ser gratuitas ou pagas, como, por exemplo, a ferramenta CASE para elicitar requisitos denominada RequisitePro, da empresa IBM©, ou a Enterprise Arquitect da empresa Sparcs©. A Tabela 14.2 apresenta uma breve lista de ferramentas proprietárias, livres ou de código aberto que podem ser utilizadas para auxiliar o processo de desenvolvimento.

Ferramenta	Descrição
Agilefant	Indicada para gerenciar projetos de software. Reúne perspectivas de organização do trabalho diário ou em longo prazo de produtos, além do planejamento de lançamento e gestão de portfólio de projetos.
Agilo for Scrum	É uma ferramenta simples, baseada na web, indicada para apoiar o processo SCRUM. Baseia-se em *Ticket Tracking*. Projetada e desenvolvida para ser utilizada pela equipe, pelo *SCRUM Master*, o *Product Owner* e todas as partes interessadas e envolvidas no projeto.
Project Planner Auto	Aplicativo que calcula automaticamente os planos de projeto, com base em um cronograma predefinido.
ClearWorks	Flexível e poderoso software de gerenciamento ágil de projetos que trabalha com SCRUM, Extreme Programming, entre outras. É também um software de BPM (*Business Process Management*) que fornece governança do ambiente empresarial do processo para melhorar a agilidade e o desempenho operacional.
Digaboard	Ferramenta de gerenciamento de projeto de âmbito visual que complementa kanban e/ou SCRUM. Uma ferramenta web que apresenta suas informações em forma de quadro.
Earned Value Add-In	Uma ferramenta para ajudar os gerentes de projeto a avaliar e visualizar projetos usando análise de valor agregado.

Ferramenta	Descrição
FDD Project Management Application	FDDPMA é um aplicativo baseado na web que gerencia projetos de software. É um projeto open source. Ideal para gerenciar seus projetos de FDD.
FireScrum	Software open source (OSS) que suporta o gerenciamento de projetos ágeis usando SCRUM.
GanttProject	Ferramenta desktop multiplataforma para acompanhamento de projetos e gestão. Funciona na plataforma Windows, Linux e Mac OSX, é gratuita e seu código é open source.
IceScrum	Ferramenta para as equipes de desenvolvimento utilizando SCRUM ou outros processos de desenvolvimento ágil.
IWebJTacker	Ferramenta para rastrear um projeto ou ferramentas de garantia da qualidade e serviços.
JAM Circle	É uma plataforma de aplicativos usando kanban.
JTrac	Aplicação web que pode ser facilmente personalizada adicionando campos personalizados e drop-downs para monitoramento de projetos.
KanbanFX	Aplicação Java de um quadro do tipo kanban.
Onepoint Project	Software de código aberto voltado à gestão de recursos, planejamento, acompanhamento de progresso, monitoramento e controle.
Open Workbench	Aplicativo desktop de código aberto que fornece programação de projetos robusta e funcionalidade de gerenciamento.

Tabela 14.2 - Quadro de ferramentas de apoio.

14.5 Exercícios

1) Quais fatores são primordiais no desempenho de uma metodologia adotada?
2) O que são ferramentas CASE?
3) Descreva os níveis da PSP.
4) Para que serve a TSP?
5) Podemos considerar que a aplicação dos métodos ágeis pode ser vista sob diversas perspectivas. Quais são elas?

Anotações

15

Academic Project Support Office (APSO)

Objetivo

- *Apresentar uma nova proposta de metodologia para desenvolvimento de projetos em instituições de ensino superior.*

15.1 Introdução

Este capítulo apresenta um novo modelo de processo de desenvolvimento, idealizado para ser usado por instituições de ensino superior, baseado em estudos consagrados no ambiente corporativo e adaptações de metodologias de gestão de projetos, sobretudo as consideradas ágeis. Os resultados obtidos são fruto de um projeto de pesquisa dos autores conduzido no Núcleo de Inovação da Faculdade de Tecnologia "Dom Amaury Castanho" - FATEC de Itu, cujos resultados até o momento indicam que o modelo proposto é factível, podendo orientar sua implantação em outras instituições de ensino, sugerindo ainda sua aplicação sob a estrutura de um núcleo de inovação externo.

15.2 A pesquisa em instituições de ensino

Diferente do que ocorre em outros países, no Brasil ainda estamos longe de obter resultados satisfatórios relacionados ao empenho e apoio a instituições de ensino que desejam fazer pesquisa. Como exemplo, podemos citar o "*The Global Information Technology Report 2010-2011*", divulgado

pelo Fórum Econômico Mundial[2], no qual o Brasil ocupa a 56ª colocação. Considerando o relatório de 2007-2008, o Brasil subiu três posições, mas ainda está abaixo de países como Chile (39ª), Barbados (38ª) ou Porto Rico (43ª) em termos de aplicação do conhecimento científico. É uma posição extremamente desconfortável, se considerarmos que estamos dentre as maiores economias do mundo. Considerando apenas o resultado dessa pesquisa, a necessidade de direcionar uma atenção maior ao incentivo à pesquisa é evidente, principalmente quando se trata da importância do desenvolvimento de sistemas/produtos competitivos.

Conceitualmente, a pesquisa acadêmica tem a preocupação de expandir o nível de conhecimento existente, e a pesquisa tecnológica demanda a adequação prática da teoria, extremamente requisitada pelo meio empresarial. Portanto, a implantação de um ambiente que incentive a produção científico-tecnológica, orientada ao desenvolvimento de soluções para os problemas gerados com o uso da tecnologia, se justifica, pois contribui para estabelecer um processo de ensino-aprendizagem entre o corpo docente e discente e especificar relações com a comunidade empresarial da região, considerando suas potenciais demandas.

É importante lembrar que, ao formar profissionais, a instituição de ensino tecnológico concretiza a razão de sua existência na sociedade. A oferta de projetos por esse tipo de instituição de ensino superior deveria ser intensa, motivada pelo grande fluxo de ideias, propostas de pesquisas e pela própria demanda do mercado que, pressionado a acompanhar as inovações, necessita de apoio para a viabilização de suas ideias.

15.3 O projeto na instituição de ensino

Valeriano (2001) afirma que o gerenciamento de projetos vem despertando inusitada atenção em muitos países, atingindo áreas como o trato da formulação de trabalhos escolares sob a forma de projetos. Ainda segundo ele, nos EUA, no Japão e em muitos países da Europa, um trabalho de campo de Biologia, por exemplo, é conduzido com a participação dos alunos, como se fosse (e como é, de fato) um projeto: com objetivos, atribuições, resultados esperados, cronograma, orçamento, controle etc.

Ao refletirmos sobre o uso de processos formais de gestão de projetos em uma instituição de ensino, convém também observar que a atividade de projetar é diferente da atividade de pesquisar. O *Project Management Institute* (PMI) define em seu PMBOK[3] que um projeto é "um empreendimento único que deve apresentar um início e um fim claramente definidos e que, conduzido por pessoas, possa atingir seus objetivos respeitando os parâmetros de prazo, custo e qualidade" (PMBOK, 2009). Já a pesquisa científica "é a realização concreta de uma investigação planejada, desenvolvida e redigida de acordo com as normas da metodologia consagradas pela ciência" (RUIZ, 1986, p. 48). Essas definições podem ser sinérgicas e complementares, mas é importante considerarmos a diferença conceitual entre as atividades inerentes à investigação científica e os processos burocráticos relacionados ao seu controle, muitas vezes repudiados pelos cientistas por ocupar um tempo precioso que poderia ser usado para a investigação científica propriamente dita.

[2] Fonte: http://www3.weforum.org/docs/WEF_GITR_Report_2011.pdf
[3] Project Management Body of Knowledge

Por outro lado, mesmo considerando que a preocupação com a documentação e a gestão de projetos acadêmicos possa ser vista como uma atividade enfadonha, ela sempre se mostrou necessária, haja vista a antiga prática de patrocínio de projetos acadêmicos por entidades externas. Quando isso ocorre, normalmente os projetos requerem a assinatura de um instrumento jurídico específico, que pode ser sob a forma de um convênio, contrato, acordo de cooperação, termo de cooperação, entre outros. Tal formalismo objetiva disciplinar as ações previstas, que devem estar em conformidade com leis federais, decretos, portarias, normas regulamentares internas, estatutos, regimentos, resoluções, instruções etc.

Além disso, para projetos acadêmicos patrocinados por determinados órgãos de fomento, é necessário haver um controle da gestão financeira dos projetos, que deve observar normas rígidas de acompanhamento, controle da execução orçamentária ou controle patrimonial. Também podemos citar o princípio da anualidade da execução de recursos alocados ao projeto, como, por exemplo, disposições da Lei Orçamentária Anual (LOA). Observando essas características e necessidades, torna-se natural pensar na aplicação de boas práticas de gestão desses projetos ou no uso de uma estrutura organizacional complementar destinada ao apoio de projetos nas instituições de ensino que realizam pesquisas.

15.4 Escritório de projetos: por quê?

Atualmente existe uma tendência nas organizações de adotar o conceito de projetos para diversos tipos de atividades, uma vez que podemos aplicá-lo nas mais variadas situações. Questões relacionadas à gestão de projeto são cada vez mais importantes na economia moderna. O termo "gestão" ou "gerenciamento de projetos", juntamente com o conceito de escritório de projetos, tem sido amplamente difundido entre as organizações, seja no ambiente acadêmico ou empresarial, trazendo o desafio de uma aplicação eficaz e resultados satisfatórios. Kerzner *et al.* (2006), ao abordar a evolução da gestão de projetos com o passar do tempo, evidenciou que ela não é mais percebida como um sistema de interesse exclusivo do plano interno das organizações. Trata-se agora de uma arma competitiva que representa níveis crescentes de qualidade e agrega valor aos interesses dos clientes.

Devemos considerar também que a gestão de projetos é uma disciplina que ainda não está completamente compreendida e estabelecida. Para ilustrar este fato, podemos nos remeter ao relatório do *Standish Group*[4], que divulgou os seguintes dados (*Chaos Report* 2009): 24% dos projetos analisados falharam (foram cancelados ou nunca usados), 44% dos projetos mudaram (atrasaram, estouraram o orçamento e/ou reduziram o escopo) e 32% dos projetos foram bem-sucedidos (dentro do prazo, dentro do orçamento e com escopo completo). Apesar de apresentar um resultado ainda ruim e preocupante, o mesmo *Standish Group* observou que os indicadores melhoraram ao longo do tempo, sendo atribuído às seguintes razões: a) tendência de utilização de projetos menores e menos complexos, com consequente diminuição do custo médio; b) desenvolvimento de ferramentas de controles de progresso de projetos e aumento da utilização de processos de gerenciamento; c) capacitação dos gerentes de projetos, resultando em um melhor gerenciamento de projetos; d) uso de estruturas como as constituídas em escritórios de projetos ou PMO (*Projects Management Office*).

Esses dados ilustram a importância atribuída à gestão de projetos atualmente, que tem proporcionado uma nova visão para as organizações, pois cada vez mais empreendem esforços em busca de uma estrutura capaz de atender a demanda com excelência. O próprio *Standish Group* enfatiza que essa excelência tem sido conquistada por empresas que utilizam modelos de escritório de projetos.

[4] Organização especializada em avaliações de risco (www.standishgroup.com).

Crawford (*apud* REIS, 2007, p.19) diz que o escritório de projetos é um "escritório", que pode ser físico ou virtual, formado por profissionais de gestão de projetos que suprem as necessidades em gerência de projetos de uma organização. O escritório de projetos se comporta então como um provedor de serviços e processos necessários para gerenciamento de projetos, envolvendo planejamento, organização, controle de escopo, prazo e custo. Existem diversos modelos de escritórios de projetos, e a maioria baseia-se na metodologia proposta pelo PMI e em fundamentações apresentadas por autores como Kerzner, Menezes, entre outros.

Neste livro, estudamos cinco modelos de PMO, mencionados a seguir: *Authonomous Project Team* (APT), *Project Support Office* (PSO), *Project Management Center of Excellence* (PMCOE), *Program Management Office* (PrgMO) e *Chief Project Officer* (CPO).

15.5 Academic Project Support Office

Os estudos iniciais conduzidos pelos autores culminaram com a criação do *Academic Project Support Office* (APSO), denominação escolhida para identificar um novo modelo de escritório de projetos voltado para necessidades de um cenário acadêmico. A seguir são apresentadas considerações resumidas sobre a escolha e adaptação de metodologias de gestão de projetos, além de uma visão geral dos processos criados.

15.5.1 Escolha e adaptação de metodologias

Sabemos que o desenvolvimento de projetos *ad hoc*, ou seja, que não segue nenhuma metodologia específica, em geral produz resultados insatisfatórios, principalmente se considerarmos sistemas complexos. Por outro lado, modelos de gestão de projetos tradicionais podem não se mostrar adequados para todos os casos, uma vez que sua filosofia pode contribuir para "engessar" projetos que não precisam, na prática, de todo o formalismo solicitado.

Partindo deste raciocínio, pesquisas foram realizadas objetivando identificar as metodologias de desenvolvimento de projetos que melhor se adaptam às necessidades acadêmicas, com foco em modelos ágeis de gestão. Convém ressaltar que uma das características das metodologias ágeis que mais chamam a atenção é sua ausência de formalismo em documentos de projeto. Para muitos este é um fator que contribui para a agilidade desejada, mas ao mesmo tempo é alvo de críticas relacionadas à falta de documentações importantes. Assim, um dos desafios encontrados foi a busca de uma dosagem processual adequada entre o que parece bom e o que é imprescindível.

Estudando as diferenças de estrutura organizacional entre instituições de ensino e organizações com fins lucrativos, muitas das quais inspiraram os principais modelos de escritório de projetos existentes, ficou clara a necessidade de desenvolver um modelo e metodologia próprios, considerando o cenário acadêmico estudado. Para servir de base a essa adaptação, o modelo convencional de escritório de projetos escolhido foi o PSO (*Project Support Office*). Conceitualmente, esse modelo possui uma estrutura de apoio técnico e administrativo, além de pregar o uso de ferramentas e serviços que auxiliam o planejamento, programação, mudanças de escopo e gerenciamento de custos. No PSO, cada recurso envolvido (externo ou interno) é alocado no projeto, dependendo da sua natureza e de sua estrutura contratual. Esse modelo serviu de base para a incorporação e o uso de outras práticas advindas do estudo de metodologias ágeis, como será visto a seguir.

Além de nos basearmos no PSO, e reconhecendo a importância de utilizarmos documentações suficientes para obter subsídios necessários para o desenvolvimento dos projetos, adotamos nos projetos piloto do escritório de projetos da faculdade uma visão que agrega contribuições das seguintes metodologias: Processo Unificado, SCRUM e FDD (*Feature Driven Development*). Basicamente consideramos a divisão de atividades proposta pelo Processo Unificado, dividida nas fases de iniciação, elaboração, construção e transição. Também do Processo Unificado utilizamos modelos do documento de visão e do documento de casos de uso, considerados importantes principalmente quando relacionados ao desenvolvimento de software.

Observe que, mesmo estando disponível, existe uma flexibilidade quanto ao uso de tais documentos, podendo ou não ser utilizados. Essa decisão vai depender da natureza e complexidade do projeto, o que concebe um maior grau de flexibilidade aos processos. Durante a fase de construção, em que, por definição, ocorre o desenvolvimento do projeto propriamente dito, propusemos a utilização da metodologia SCRUM para realizarmos o acompanhamento. Essa metodologia também foi adaptada para necessidades do cenário acadêmico, sendo utilizados como base apenas dois documentos de especificação chamados *Product Backlog* e *Sprint Backlog*. A Figura 15.1 apresenta um resumo das principais fases e documentos utilizados pela metodologia adaptada.

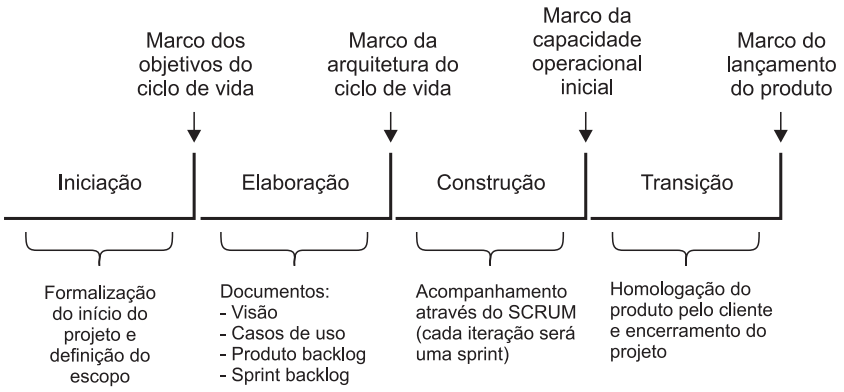


Figura 15.1 - Principais fases da metodologia adaptada. Fonte: Arquivo pessoal do autor.

Outro exemplo de adaptação realizada diz respeito às reuniões diárias (de no máximo dez minutos) propostas pela metodologia SCRUM. No caso do escritório de projetos da faculdade, substituímos essa reunião diária por uma reunião semanal, considerando a indisponibilidade dos alunos e professores envolvidos. Contudo, caso seja necessário, o SCRUM *Master*, que desempenha um papel de responsabilidade técnica na condução do projeto, deve proporcionar mecanismos informatizados de comunicação entre essas reuniões, para que possa ser consultado ou para colaborar na solução de problemas críticos.

A atualização do *task board*, que é um quadro utilizado para o acompanhamento das iterações, que deveria ser feita durante essa reunião diária, passou a ser feita a qualquer momento, considerando a disponibilidade dos integrantes do projeto. Outro aspecto adaptado diz respeito aos "*post-it*" utilizados no *task board*, que contêm informações sobre as atividades que estão sendo executadas. No caso da proposta adaptada, utilizamos elementos complementares inspirados em registros da metodologia FDD, como a indicação de andamento de completude da tarefa, conforme ilustra a Figura 15.2.

ID Requisito:					Duração (hs):				
Tarefa:									
Descrição:									
Responsável:									
Andamento (% concluída):									
0	20	30	40	50	60	70	80	90	100

Figura 15.2 - Registro de tarefa a ser realizada, utilizado no task board de acompanhamento de projeto. Fonte: Arquivo pessoal do autor.

Product Backlog	To Do	Doing	Done
Sprint Backlog	Unplanned Items	Impediments	To Discuss

Figura 15.3 - Visão geral do task board de acompanhamento de projeto. Fonte: Arquivo pessoal do autor.

Pelas informações contidas nesse registro, aliadas ao seu posicionamento no *task board*, torna-se possível a qualquer um observar o andamento do projeto, de maneira clara e intuitiva. A Figura 15.3 ilustra uma visão geral dos principais elementos utilizados no *task board*. De maneira resumida, o *task board* possui a seguinte dinâmica: a coluna reservada para o *Product Backlog* contém os "*post-it*" das atividades totais que o projeto deve realizar para ser concluído. A coluna do *Sprint Backlog* relaciona atividades de uma iteração específica. Durante uma iteração, integrantes do projeto escolhem atividades presentes na coluna "*To Do*" e a transferem para a coluna "*Doing*", indicando que estão em andamento e sob sua responsabilidade. Quando concluem a atividade, transferem para a coluna "*Done*".

O *task board* também pode ter adicionalmente uma coluna chamada "*To Verify*", indicando que existem atividades concluídas que já podem ser testadas. A coluna "*Unplanned Items*" é utilizada para indicar atividades que devem ser executadas, mas que não haviam sido previstas inicialmente. A coluna "*Impediments*" contém atividades que estão paralisadas por alguma razão, sendo alvo de atenção do gestor responsável. Adicionalmente, pode-se usar a coluna "*To Discuss*" para evidenciar atividades cuja execução demanda discussão técnica. A mudança das atividades através das colunas é feita pelos integrantes de projeto, e ao final de uma semana se reúnem para discutir o andamento da iteração, que termina quando todas as atividades do *Sprint Backlog* forem concluídas. A partir daí, nova iteração se inicia, com novos "*post-it*" advindos do *Product Backlog*. Informações complementares sobre as características do *task board* foram exploradas no capítulo 11.

15.5.2 Processos desenvolvidos

Processos foram concebidos durante o estudo para serem utilizados pelo escritório de projetos da faculdade, resultando na definição de diagramas que contemplam macroprocessos, Figura 15.4, e subprocessos, Figura 15.5. O propósito desse esforço foi perceber a maneira como o escritório do projeto deve trabalhar em conjunto com os principais envolvidos. Após estudar os processos criados, modelos de documentações, como, por exemplo, para submissão e aprovação de projetos, também foram definidos.

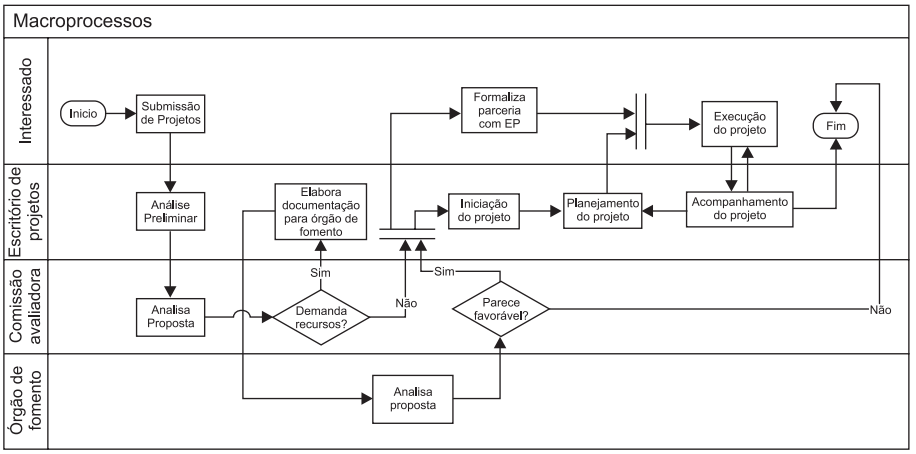


Figura 15.4 – Macroprocesso do escritório de projeto da faculdade. Fonte: Arquivo pessoal do autor.

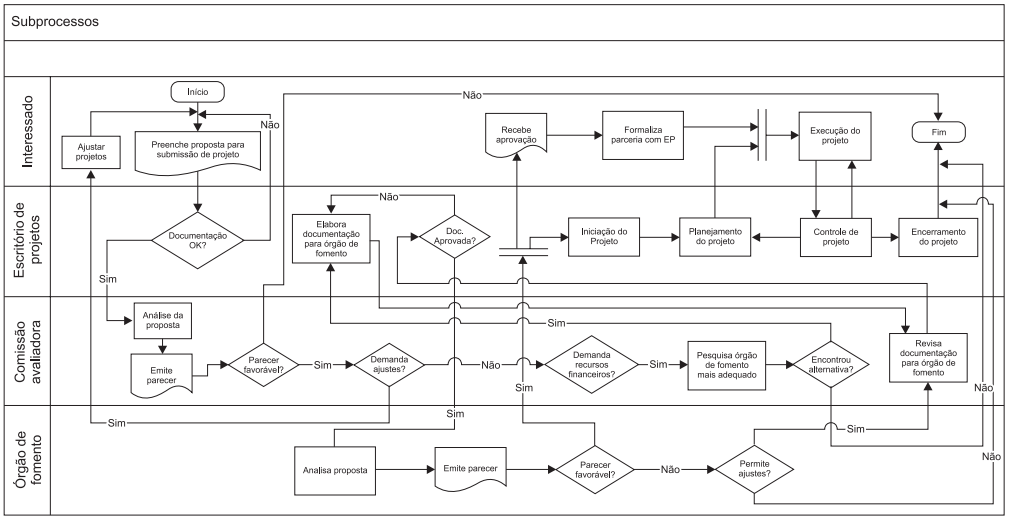


Figura 15.5 – Subprocessos do escritório de projeto da faculdade. Fonte: Arquivo pessoal do autor.

Conforme podemos perceber pela análise dos processos definidos, existem basicamente quatro atores que podem interagir: o interessado em executar um projeto (docentes, discentes, empresas, órgãos públicos etc.); o escritório de projetos propriamente dito, com suas responsabilidades sobre o processo; uma comissão de avaliação dos projetos submetidos; e eventuais órgãos de fomento envolvidos.

Considerando que a aprovação de projetos de interesse da instituição de ensino deve ser fruto de uma participação colaborativa de membros da comunidade acadêmica, propusemos que essa tarefa seja delegada por integrantes de outros grupos de pesquisa ou núcleos que podem ser instituídos (caso já não existam) em uma faculdade. Esses núcleos representam um órgão colegiado interdisciplinar e independente de apoio acadêmico, de caráter consultivo, deliberativo e educativo, que devem ser constituídos nos termos da legislação vigente sob o qual a faculdade se insere. Esse núcleo deve possuir regimento próprio e ser criado, entre outras finalidades, com o objetivo de

deliberar sobre demandas de projetos submetidas ou propostas pelo escritório de projetos da faculdade. Observando o processo definido, verificamos que os projetos submetidos ao escritório de projetos devem passar por uma pré-análise inicial, para posteriormente serem encaminhados a uma comissão de avaliação do núcleo de pesquisa, composta por integrantes de seus eixos de pesquisa.

15.6 Escritório de projetos X inovação tecnológica

A aplicabilidade observada de um modelo de escritório de projetos para a condução de projetos acadêmicos de uma unidade de ensino superior nos leva a crer que esse mesmo modelo também poderia ser utilizado por outras instituições de ensino. Várias dessas instituições possuem políticas de inovação, muitas vezes representadas por agências de inovação. Normalmente essas estruturas organizacionais têm a finalidade de apoiar a gestão da inovação, a transferência de tecnologia e a ampliação do capital intelectual, formando uma rede de gestores da inovação, facilitando o desenvolvimento de parcerias com o setor empresarial. Essa importante estrutura promove ações dirigidas ao relacionamento com as empresas, com o setor público e com instituições de ciência e tecnologia, estimulando o desenvolvimento de parcerias estratégicas para a pesquisa aplicada e a transferência do conhecimento.

Os escritórios de projetos das faculdades seriam então responsáveis pelo planejamento e acompanhamento de projetos envolvendo empresas da região ou pelas iniciativas de empreendedorismo advindas de eventuais *spin-off* (termo utilizado quando um projeto oriundo de uma instituição de ensino se transforma em uma empresa) ou conduzidas em parcerias com incubadoras regionais. Explorando melhor esta visão, convém salientar que o escritório de projetos seria parte integrante do núcleo de inovação, cujo foco de atuação principal pode ser resumido nas seguintes atividades: integração e comunicação, identificação e acompanhamento de riscos, identificação e acompanhamento de interdependências, soluções de questões críticas, entre outras.

15.7 Considerações finais

Com base na estrutura proposta e considerando as motivações apresentadas, concluímos que a utilização de escritórios de projeto por instituições de ensino superior estimularia o uso de boas práticas de gestão de projetos, concebendo, consequentemente, maior qualidade e condições favoráveis para sua execução. Neste cenário, o escritório de projetos pode desempenhar diversas funções, destacadas a seguir: integradora (do corpo docente, discente e comunidade empresarial), propositiva (de projetos e pesquisas), organizadora e de registro (por meio de banco de dados de projetos, relatórios, monografias, publicações), articuladora (colaborando com estágios supervisionados e trabalhos de conclusão de curso), comunicativa (divulgação interna e externa de dados e resultados em web sites e boletins periódicos) e capacitadora (de docentes, discentes, educadores e comunidade).

Fomentar o relacionamento empresa-escola por meio do escritório de projetos é uma iniciativa que pode gerar grandes contribuições para ambos os lados. Essa sinergia gera ações com foco no conhecimento, aprendizagem e na sociedade, pela possibilidade de condução de projetos que evidenciam a formação não apenas de um profissional, mas de cidadãos capazes de interagir, avaliar e transformar o meio em que se inserem, com postura crítica, ética e humanista. A estrutura proposta permitirá que o pesquisador se concentre mais nas suas atividades de pesquisa, tirando dele o peso dos aspectos burocráticos advindos de processos necessários para o sucesso do esforço empreendido.

16

Considerações Complementares sobre Gestão de Projetos

Objetivos

- *Apresentar considerações complementares sobre gestão de projetos, incluindo aspectos referentes ao planejamento de negócios, implementação de projetos, desenho organizacional dos projetos, fatores de sucesso e fracasso, liderança de projetos e capacitação dos envolvidos.*

Independentemente da metodologia de desenvolvimento escolhida, ágil ou tradicional, existem recomendações e preocupações que são universais. Este capítulo apresenta algumas reflexões e recomendações sobre temas que merecem atenção, independentemente do projeto e do paradigma de desenvolvimento utilizado.

16.1 Planejamento de negócios X implementação de projetos

O alinhamento de projetos exige um caminho coerente entre as estratégias determinadas pela empresa e as ações tomadas pelas equipes de projeto. Com relação às estratégias da empresa, podemos dizer que são obtidas por planejamento estratégico convencional, que pode incluir a criação ou correção das declarações de missão, visão e valores da empresa. Além disso, também existem elementos relacionados à revisão de cenários econômicos, à análise da concorrência, uma avaliação de riscos e oportunidades, bem como a articulação dos objetivos estratégicos da organização, os quais são o ponto de partida para qualquer tipo de projetos, sejam eles empreendimentos estratégicos específicos ou projetos relacionados ao lançamento de um produto, entre outros.

Uma vez que os objetivos estratégicos estejam identificados, o sucesso do alinhamento estratégico de projetos depende do estabelecimento de uma interface entre os objetivos estratégicos e o cenário específico de cada projeto. Sabemos que cada grande grupo de projetos tem suas peculiaridades. Projetos estratégicos, por exemplo, estão fortemente amarrados à missão, visão, valores e objetivos da empresa, e dependem bastante da coordenação da alta gerência para atingir suas metas. Já os projetos de produto dependem de se estabelecerem metas de produtos, monitorar oportunidades de mercado etc.

De uma maneira geral, os projetos têm sido tradicionalmente divididos em duas fases ou dois mundos. O primeiro é o "**mundo que vai da ideia ao início do projeto**", e o segundo é a passagem do projeto para as áreas funcionais ou unidades operacionais, que são aqueles que de fato "**farão o trabalho**". Portanto, o principal desafio em gerenciar uma organização que desenvolve projetos é fazer a ligação entre esses dois mundos e assegurar que os projetos recebam atenção e apoio suficientes da alta gerência. Considerando essa necessidade, as organizações têm atribuído essa tarefa a um profissional, muitas vezes conhecido como "*sponsor*" do projeto, o qual representa a ponte existente entre a organização e os projetos em andamento na empresa, tendo como principal atribuição cuidar e alimentar um projeto, de modo que ele receba recursos. Para exercer esse cargo, o profissional precisa ter algumas características, que podem variar dependendo da empresa. A seguir exemplificamos algumas dessas características:

- Ter um real interesse no projeto;
- Possuir conhecimentos e capacidades relacionados ao gerenciamento estratégico e de projetos;
- Ter a capacidade de influenciar outros executivos e grupos importantes;
- Possuir bom relacionamento com o gerente do projeto e sua equipe.

Existem diversas formas clássicas de patrocínio do projeto. A seguir vamos apresentar um resumo de algumas possibilidades, considerando a figura do *sponsor*.

- ***Sponsor* único:** uma abordagem que envolve o indivíduo que tem todas ou a maioria das características mencionadas anteriormente e que tem a única tarefa de patrocinar um determinado projeto.
- ***Sponsor* duplo:** dois patrocinadores são utilizados com frequência, particularmente quando há grande necessidade de uma pessoa para o gerenciamento tecnológico e outra para o gerenciamento tradicional.
- ***Sponsor* coletivo:** uma abordagem de grupo que faz sentido em algumas situações e pode ter a forma de um comitê consultivo, um conselho de projeto ou um grupo encarregado de cumprir as tarefas de patrocínio.

É importante observar que um gerente de projeto forte e bem posicionado politicamente pode precisar de pouco suporte de um *sponsor*. Por outro lado, um gerente de projeto menos maduro, com uma equipe recém-formada, pode precisar de um patrocinador. Cabe ao *sponsor* realizar algumas tarefas específicas, que variam ao longo do ciclo de desenvolvimento. A seguir exemplificamos onde o *sponsor* pode atuar, considerando algumas fases importantes do projeto.

16.1.1 Início do projeto

- Assegurar que estratégias, planos e controles do projeto estejam estabelecidos;
- Fornecer apoio para formação da equipe de projeto;
- Assegurar que o projeto seja adequadamente iniciado e que as iniciativas de trabalho em equipes sejam tomadas;
- Fornecer apoio político para o projeto no nível executivo;
- Orientar o gerente do projeto desde o início.

16.1.2 À medida que o projeto se desenvolve

- Participar das avaliações formais periódicas do projeto;
- Estar disponível para apoio e consultas;
- Acompanhar os relatórios de progresso do projeto;
- Envolver-se caso o projeto saia de seu curso.

16.1.3 Quando o projeto está chegando próximo ao fim

- Monitorar a transição do projeto para a operação;
- Estimular um rápido encerramento do projeto;
- Assegurar a documentação das lições aprendidas.

A abordagem do uso de *sponsor* para patrocinar projetos tem se mostrado uma forma eficaz de preencher a lacuna existente entre as estratégias organizacionais e a implementação de projetos. Essa abordagem começou como uma ajuda para projetos com problemas e evoluiu para uma estratégia formal. Contudo, estará longe de atingir sua meta se a empresa não estiver estruturada especificamente para o gerenciamento de organizações por projetos.

16.1.4 Alinhamento geral de projetos

Para que o patrocínio de projetos seja eficaz, os projetos de uma organização precisam ser embasados em técnicas, metodologias e sistemas empregados em toda a empresa, o que torna o trabalho mais fácil. Sabemos que o patrocínio de projetos é, afinal, uma atividade de tempo parcial, assumida em adição às atividades regulares do *sponsor*. Portanto, podemos dizer que as bases do alinhamento geral de projetos são:

- O gerenciamento dos *stakeholders*;
- A priorização de projetos;
- O gerenciamento de riscos;

- Uma avaliação equilibrada (*Balanced Scorecard*);
- Os sistemas de gestão em nível empresarial;
- O planejamento estratégico de projetos.

Outro aspecto importante diz respeito à priorização de projetos. Os projetos precisam ser sistematicamente classificados, de modo a destacar os de prioridade mais elevada. O pano de fundo para essa classificação são as clássicas categorias *Strenghts-Weaknesses-Opportunities-Threats* (SWOT), ou seja, forças, fraquezas, oportunidades e ameaças. Os fatores SWOT podem, por exemplo, mostrar que uma empresa deva preferir projetos que:

- Ajudem a abrir novos mercados;
- Aumentem a imagem da empresa;
- Desenvolvam *know-how* a ser utilizado em projetos futuros;
- Criem uma forte vantagem competitiva.

16.1.5 Gestão de riscos

Também devemos lembrar que projetos são negócios arriscados. Portanto, uma empresa deve ter processos para identificação, quantificação, desenvolvimento de respostas e controle do risco. Esses processos devem ser aplicados a todos os projetos da empresa, em uma linguagem que todos possam compreender. O risco tem três componentes básicos:

a) O evento ou fato que caracteriza o risco;

b) A probabilidade de que o evento realmente aconteça;

c) O impacto, medido em termos financeiros, do acontecimento do evento.

Como o gerenciamento de risco não é algo feito intuitivamente pela maioria dos gerentes de projeto, é preciso haver uma metodologia para cobrir os riscos desde o início até o término de um projeto.

16.1.6 Avaliação equilibrada

Uma técnica de avaliação gerencial denominada *Balanced Scorecard* (avaliação equilibrada) sugere que a saúde de um negócio pode ser calculada a partir de quatro ângulos diferentes: financeiro; cliente; processos internos; aprendizagem e crescimento. Admitindo que todos os projetos sejam desenhados para aumentar a saúde da empresa, essas áreas precisam ser examinadas em termos da contribuição que cada projeto traz para a empresa. Como os executivos são responsáveis pelo destino da organização, precisam avaliar e monitorar projetos em andamento e assegurar que eles estejam contribuindo para as quatro áreas críticas. Se as contribuições de um dado projeto forem concentradas, por exemplo, só no lado financeiro, então é necessário uma reavaliação para examinar como o projeto pode ser alavancado de modo que ele tenha um impacto mais amplo sobre a organização.

16.1.7 Avaliação equilibrada

Para que as estratégias e os projetos de uma organização se fundam, é necessário lidar com a questão do dinheiro, o que significa gerenciar:

- As estimativas iniciais;
- Os orçamentos detalhados dos projetos;
- As estimativas versus as despesas reais;
- O fluxo de caixa.

16.1.8 Alinhamento estratégico de projetos

Essa tarefa é idealmente cumprida por um banco de dados corporativo integrado, de modo que a informação flua livremente do nível de projeto para o nível empresarial e vice-versa. O alinhamento estratégico de projetos também envolve uma análise global dos recursos disponíveis versus recursos necessários, não apenas para recursos humanos, como também para recursos financeiros e materiais.

16.1.9 Planejamento estratégico de projetos

Tem por objetivo assegurar que cada projeto desenvolva e implemente as estratégias necessárias para garantir que seus objetivos sejam atingidos. Isso exige o desenvolvimento de um documento inicial do projeto, que inclua:

- Os objetivos do negócio;
- Os objetivos do projeto;
- O cronograma;
- O orçamento;
- As restrições e hipóteses que influenciem o projeto.

16.2 Desenho organizacional dos projetos

Também consideramos importante apresentar uma visão do desenho organizacional usado na gerência de projetos, com o objetivo de abordar algumas falhas que é possível encontrar nos modelos tradicionais. Como podemos observar na maioria das organizações tradicionais, as hierarquias existentes tendem a ser lentas, inflexíveis e deficientes ao tentar possibilitar um enfoque organizacional das atividades do projeto. Neste cenário, também é normal existirem barreiras que sufocam o fluxo das atividades necessárias quando projetos são implementados. A inadequada delegação de autoridade e responsabilidade para sustentar as atividades do projeto tem sido um problema comum que também pode ser observado. Portanto, uma modificação do desenho organizacional tradicional pode ser requerida para dar suporte às atividades do projeto, principalmente se considerarmos o uso de paradigmas ágeis.

16.2.1 Organização do projeto

A organização do projeto é um desenho temporário usado para indicar uma equipe interorganizacional que trabalha em conjunto para a gerência do projeto. É importante perceber que, na organização do projeto, o pessoal é proveniente das unidades funcionais da empresa, formando uma equipe de projeto sobreposta à estrutura organizacional existente.

16.2.2 Departamentalização tradicional

Entre os meios tradicionais mais comumente usados para descentralizar autoridade, responsabilidade e obrigação de prestar contas, podemos incluir os seguintes:

- **Departamentalização funcional:** as unidades organizacionais descentralizadas têm por base especialidades comuns, tais como finanças, engenharia, produção etc.
- **Departamentalização por produto:** as unidades organizacionais são responsáveis por um produto ou linhas de produto.
- **Departamentalização por consumidor:** as unidades descentralizadas são estruturadas em torno de grupos de consumidores.
- **Departamentalização territorial:** as unidades organizacionais são estruturas em bases geográficas. Por exemplo, área de mercado do Sudeste do Brasil, Nordeste etc.
- **Departamentalização por processo:** os recursos humanos e outros são baseados em fluxo de trabalho, tal como uma refinaria de óleo, por exemplo.

16.2.3 Organização matricial

Numa organização matricial existe uma divisão de autoridade, responsabilidade e obrigação de prestar contas entre a equipe do projeto e as unidades funcionais da organização. A organização matricial é caracterizada pela descrição específica dos papéis individuais e coletivos no gerenciamento do projeto. Encontramos nesse modelo a figura de uma interface projeto-função, em que são focados papéis complementares de autoridade, responsabilidade e obrigação de prestar contas. A Tabela 16.1 apresenta um exemplo de modelo que pode ser usado como guia para entender a interface relativa à autoridade - responsabilidade - obrigação de prestar contas dentro da organização matricial.

Gerente do projeto	Gerente funcional
O que vai fazer?	Como a tarefa será feita?
Quando se realizará a tarefa?	Onde se fará a tarefa?
Por que efetuar a tarefa?	Quem fará a tarefa?

Tabela 16.1 – Exemplo de divisão de autoridade.

16.2.4 Papéis do gerente de projeto

O papel do gerente de projetos nas organizações é muito diversificado. Em alguns casos, existe a possibilidade de determinada organização atribuir esse papel em regime *ad hoc*, por um ocupante de um cargo que possui outro título, como engenheiro, chefe de seção ou diretor geral. Essas pessoas acabam recebendo essa incumbência em regime de dedicação exclusiva ou acumulando essa tarefa com outras, sem desvincular-se de seu cargo original. Em outro cenário mais confortável, o papel de gerente de projetos é fixo e está instituído oficialmente. Portanto, a diversidade desse cargo é diretamente influenciada pela estrutura na qual ele está inserido e as disciplinas envolvidas no projeto. Seja qual for a configuração que tenha seu cargo, dentre os muitos papéis desempenhados pelo gerente do projeto, podemos destacar:

- Um estrategista em desenvolver o senso de direção para o uso dos recursos do projeto;
- Um negociador na obtenção de recursos para apoiar o projeto;
- Um organizador em reunir uma equipe e focar na gerência do projeto;
- Um líder para recrutar e controlar o planejamento e a execução dos recursos para dar suporte ao projeto;
- Um mentor em prover aconselhamento e orientação aos membros da equipe do projeto;
- Um motivador em criar um ambiente para que a equipe do projeto possa extrair seu melhor desempenho;
- Um controlador, que mantém o controle da eficácia com a qual os recursos estão sendo usados para apoiar os objetivos do projeto;
- Um diplomata, que cria e mantém alianças com os *stakeholders* para ganhar o apoio aos objetivos do projeto.

16.2.5 Mapeamento da organização do projeto

A escolha da estrutura organizacional do projeto depende das características do projeto e de restrições externas, como, por exemplo, uma política organizacional já existente. É difícil prescrever essas estruturas, pois o que é eficaz em certos casos dependerá muito do cenário em que está inserido. Com o objetivo de dar uma ideia de como esse mapeamento pode ser feito, descrevemos a seguir um possível alinhamento de uma organização com suas respectivas responsabilidades, com o uso de uma técnica conhecida como Mapa de Responsabilidade Linear.

Sabemos que o mapa típico da organização piramidal é, na verdade, uma simplificação da organização, pois embora descreva a estrutura da organização, não especifica os papéis individuais e coletivos. Portanto, é insuficiente para clarear as incontáveis relações entre os membros do projeto e os *stakeholders*. Além disso, não deixa claro como as pessoas deveriam, em tese, trabalhar juntas na execução do trabalho da empresa.

Uma alternativa para cobrir esses aspectos é a utilização do Mapa de Responsabilidade Linear (MRL), que exibe a junção dos pacotes de trabalho do projeto com pessoas na organização. A Tabela 16.2 apresenta um exemplo de Mapa de Responsabilidade Linear para as relações de autoridade-responsabilidade-obrigação de prestar contas, dentro de uma organização matricial.

Atividades	Gerente geral	Executivo de projetos	Gerente do projeto	Gerente funcional
Direção do projeto	2	1	5	3
Planejamento do projeto	4	2	1	3
Orçamento do projeto	4	6	1	3
Controle do projeto	4	2	1	3
Administração de gastos	2	4	3	1

Tabela 16.2 - Mapa de Responsabilidade Linear.

Legenda

1) Responsável
2) Supervisão geral
3) Deve ser consultado
4) Pode ser consultado
5) Deve ser notificado
6) Autoridade de aprovação

O MRL deve ser desenvolvido de forma cooperativa com os membros da equipe do projeto. Uma vez completado um MRL, pode tornar-se um "documento vivo", com as seguintes funções:

- Mostrar papéis formais e esperados de autoridade-responsabilidade-obrigação de prestar contas.
- Informar a todos os *stakeholders* os detalhes de como os pacotes de trabalho serão distribuídos no projeto.
- Contribuir para o comprometimento e a motivação dos membros da equipe.
- Fornecer um padrão para o papel que o gerente de projeto e os membros da equipe desempenharão ao monitorar o que as pessoas estão fazendo no projeto.

16.2.6 Considerações sobre autoridade

A autoridade é definida como um poder legal ou legítimo para comandar ou agir. Aplicada ao gerente, autoridade é o poder para comandar outros para agirem ou não agirem. Existem dois tipos básicos de autoridade:

- **Autoridade de *jure*:** é o poder legal ou legítimo para comandar ou agir na gerência de um projeto. É usualmente expressa na forma de uma política, descrição de posição, carta de nomeação ou outra forma de documentação. A autoridade de *jure* prende-se a uma posição organizacional.
- **Autoridade de *facto*:** é o poder trazido à gerência de um projeto em razão de conhecimento, aptidões, habilidades interpessoais, competência, especialidade, entre outros quesitos, de uma pessoa.
- **Poder:** é a posse de um papel organizacional que dá a um indivíduo extraordinária influência sobre outra pessoa, devido a características especiais como uma especialização, dedicação, rede de comunicações, alianças, habilidades interpessoais etc.

16.2.7 Falhas de autoridade e responsabilidade (A-R)

A maioria das falhas de A-R no desenho organizacional do projeto pode ser atribuída a um ou mais dos seguintes fatores:

- Falha ao definir a A-R legal dos principais participantes no desenho organizacional do projeto.
- Má vontade das pessoas em dividir a A-R nas questões relativas ao projeto.
- Existência de ambiente cultural que reforça a mentalidade de "comando e controle" do desenho organizacional tradicional etc.

16.2.8 Documentação da autoridade e responsabilidade

Uma documentação adequada deveria ser publicada com respeito às autoridades e responsabilidades dos principais participantes do projeto. A Figura 16.1 fornece um exemplo da natureza de tal documentação.

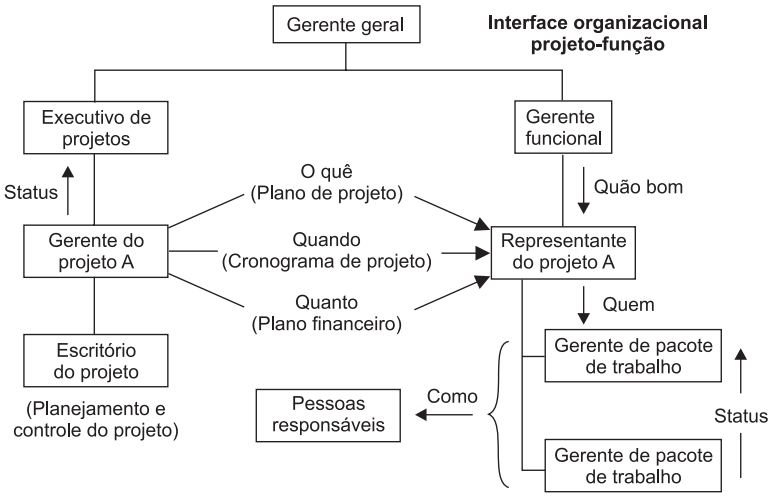


Figura 16.1 - Interface organizacional projeto-função.

Considerações sobre a Figura 16.1

Executivo de projetos: é responsável por:

- Dirigir e avaliar a atividade do gerente de projeto;
- Planejar, propor e colocar em prática a política de gerência de projeto;
- Assegurar a concordância do projeto com os compromissos contratuais.

Gerentes funcionais: são responsáveis por:

- Efetuar as tarefas de um pacote de trabalho no prazo e dentro do orçamento;
- Proporcionar políticas funcionais e guias de procedimento;
- Proporcional pessoal adequadamente apto;
- Manter a excelência técnica.

Gerentes de projetos: são responsáveis por:

- Desenvolver e manter os planos de um projeto;
- Estabelecer um cronograma do projeto e a direção financeira;
- Avaliar e informar a performance do projeto.

Gerentes de um pacote de trabalho: são responsáveis por:

- Estabelecer e manter os planos de um pacote de trabalho;
- Estabelecer um guia técnico de um pacote de trabalho;
- Estabelecer um cronograma detalhado de um pacote de trabalho e as verbas operacionais;
- Controlar e reportar a performance de um pacote de trabalho.

Representante do projeto: é responsável por:

- Planejar e controlar ativamente os esforços de sua organização no projeto.

16.2.9 Definição das obrigações de prestar contas

Prestar contas é um compromisso assumido por algo que agregue valor, seja por meio de um contrato ou em razão de uma posição de responsabilidade. Custos, cronograma e parâmetros de performance técnica são elementos em torno dos quais fluem as forças de autoridade-responsabilidade-obrigação de prestar contas. A Figura 16.2 ilustra uma maneira de retratar tais forças.

Figura 16.2 - Elementos que influenciam forças de autoridade-responsabilidade-obrigação.

16.2.10 Treinamento em gerência de projetos

Treinamento é um importante aspecto para elevar os níveis de conhecimento e mudar o comportamento dos indivíduos em projetos. O treinamento dá ao indivíduo maior capacidade para atuar em níveis mais altos de produtividade e fazer maiores contribuições ao sucesso do projeto. Treinamentos em habilidades interpessoais são tão importantes quanto o treinamento em conhecimento das funções da gerência do projeto. Treinamento de capacidade também é importante para as ferramentas de gerência de projetos. Essas ferramentas preenchem os requisitos de planejamento de tempo (ferramentas de cronograma), relatórios (gráficos), acompanhamento e processamento de custos (planilhas), acompanhamento de dados (base de dados) etc.

Observe que nem todas as pessoas requerem o mesmo tipo ou nível de treinamento dentro das áreas de conhecimento e habilidades. O treinamento pode estar voltado para a familiarização ou amplo conhecimento geral ou conhecimento detalhado do trabalho. A posição e a responsabilidade de uma pessoa determinam o nível de conhecimento requerido e os requisitos de treinamento. O treinamento também há de considerar se os indivíduos trazem conhecimento anterior inapropriado e que deve ser superado ou se o treinamento é a primeira exposição no assunto. Existem também treinamentos para reforçar conceitos e conhecimentos, assim como treinamentos avançados.

16.2.11 Categorias gerais das pessoas envolvidas em projetos

Em geral, existem quatro níveis de indivíduos que precisam de algum conhecimento de gerência de projetos. A Tabela 16.3 resume as categorias de pessoas envolvidas em projetos e o nível de conhecimentos requeridos para efetivamente cumprirem suas responsabilidades:

Responsabilidade	Papel	Conhecimento de gerência de projetos
Direção estratégica	Líderes executivos	Capacidades de gerência de projetos para suportar as metas estratégicas
Alocação de recursos	Gerente/patrocinador de líderes de projetos	Conhecimento das metas estratégicas da organização e *links* como projeto
Aplicação dos recursos	Líder do projeto	Conhecimento do projeto e dos processos
Utilização de um recurso	Integrante da equipe do projeto	Conhecimento detalhado dos componentes do trabalho do projeto

Tabela 16.3 - Categorias de pessoas.

16.2.12 Áreas de conhecimento e habilidades para os participantes do projeto

Sabemos que o aumento da competitividade tem impulsionado mudanças no ambiente empresarial, impulsionando as organizações a gerenciarem suas competências. O setor de recursos humanos cada vez mais tem se dedicado à gestão das competências empresariais, buscando envolver as pessoas nas estratégias empresariais, implantando modelos de remuneração atrelados às pessoas e não aos cargos e contribuindo para aumentar a formação educacional dos colaboradores. A seguir

faremos um resumo dos conhecimentos e habilidades que têm sido esperados daqueles envolvidos com projetos, começando com o estudo da Tabela 16.4, que ilustra os conhecimentos e habilidades esperados para alguns cargos comumente encontrados no desenvolvimento de projetos.

Conhecimentos/Habilidades	GE	PP	LP	PL	CP	IE
Planejamento estratégico	X	X				
Metas estratégicas da organização	X	X	X			
Informação do projeto (conhecimento e interpretação dos dados do projeto)	X	X	X	X	X	X
Planejamento de projetos		X	X	X		
Gerência da qualidade do projeto			X	X	X	
Planejamento e gerência do custo do projeto			X	X	X	
Ferramentas do projeto			X	X	X	X

Tabela 16.4 – Áreas de conhecimento e habilidades para os participantes do projeto.

Abreviaturas utilizadas na Tabela 16.4: GE (Gerente Executivo); PP (Patrocinador do Projeto); LP (Líder do Projeto); PL (Planejador do Projeto); CP (Controlador do Projeto); IE (Integrante da Equipe)

16.2.13 Entendimento do trabalho do projeto pelos profissionais de projeto

Praticamente todos os projetos são planejados e implementados em um contexto social, econômico e ambiental, portanto a equipe integrante do projeto deve compreender sua participação no projeto, levando em consideração seus contextos sociais, culturais, ambientais, políticos, físicos, entre outros.

Quando pensamos no ambiente cultural e social no qual o projeto está inserido, a equipe vai precisar entender como o projeto pode afetar as pessoas ou vice-versa. Já a visão do ambiente político ou internacional envolvido pode levar alguns membros do projeto a se familiarizarem com leis e costumes internacionais, considerando, por exemplo, diferenças de fuso horário, feriados, possibilidades de teleconferências, viagens, entre outras providências. Outro aspecto que podemos observar é que os profissionais de projeto podem se beneficiar do entendimento das características de um projeto à medida que o relacionam às suas responsabilidades.

16.2.14 Escritório de projetos (Project Office)

Neste item vamos explorar um pouco mais o conceito de escritório de projetos, já abordado em capítulos anteriores. Como foi mencionado anteriormente, a utilização do modelo de escritório de projetos está ocorrendo em diversas organizações para alcançar os benefícios da consolidação de muitas funções da gerência de projetos. A centralização das funções permite a uma organização ganhar consistência em práticas, assim como no uso de padrões comuns para itens como cronogramas e relatórios.

Um escritório de projeto é tipicamente iniciado quando desejamos reduzir os custos das funções da gerência de projetos em uma organização e aprimorar a qualidade da informação sobre o

projeto fornecida ao dirigente da empresa. Tem a função de grupo de apoio que provê serviços aos gerentes de projeto, dirigentes e gerentes funcionais que trabalham juntos em projetos. Um escritório de projetos pode também ser reconhecido com vários nomes diferentes, sendo os mais comuns:

- Escritório de apoio ao projeto;
- Escritório de apoio ao programa;
- Escritório da gerência de projeto;
- Escritório de apoio à gerência de projeto;
- Escritório do programa etc.

Na prática, o escritório de projetos é o que uma organização quer que ele seja. Pode ser tão simples como umas poucas pessoas preparando e mantendo cronogramas, até ter a participação de vários colaboradores realizando planejamento, informes, garantia de qualidade, coleta de informações de desempenho etc. O escritório de projetos é definido pelas necessidades de negócios da organização e cresce com essas necessidades. Devemos sempre lembrar que projetos são "**veículos de mudança**" pelos quais as empresas implementam suas estratégias. Sendo assim, investir na estruturação de um escritório de projetos pode propiciar uma vantagem competitiva e colaborar com a melhora do desempenho de seus projetos. Para entender melhor seus propósitos, vamos observar algumas funções desempenhadas pelo escritório de projetos na Tabela 16.5.

Área de trabalho	Serviços prestados
Apoio para o planejamento do projeto	Manter a metodologia e as variações dos processos padrão; armazenar e atualizar planilhas para o planejamento; manter a medida de progresso; disponibilizar consultas sobre estimativas de custos e prazos etc.
Auditoria do projeto	Processar listas de comprovação das "*milestones*"; apoiar a intervenção em um projeto devido a deficiências; manter um registro das ações corretivas.
Apoio para o controle do projeto	Manter um registro e o acompanhamento do controle de mudanças; manter as ações de controle de mudanças e os elementos de conclusão; efetuar sínteses e resumos de projetos etc.
Apoio para a equipe do projeto	Ensinar e dirigir as técnicas de gerência de projetos necessárias etc.
Desenvolvimento de habilidades para gerência de projetos	Identificar as demandas gerais de treinamento para o processo; manter políticas, procedimentos e práticas para a gerência de projetos; institucionalizar a gerência do projeto.
Recursos para a gerência de projetos	Avaliar as necessidades de ferramentas para os projetos; coordenar o treinamento em ferramentas das equipes de projetos; recomendar prioridades para os projetos novos etc.
Apoio executivo para projetos	Recomendar a alocação de recursos entre os projetos; revisar as avaliações de rendimento dos projetos; servir como consultoria de gerência de projetos para executivos.
Itens de ações	Estabelecer um registro e acompanhamento dos itens de ações; Distribuir relatórios para os *stakeholders*; manter cópias registradas das comunicações etc.

Tabela 16.5 – Funções desempenhadas pelo escritório de projetos.

16.2.15 Implementação de um escritório de projetos

Para iniciar e amadurecer a ideia de implantar um escritório de projetos, é necessário seguir, basicamente, os seguintes passos:

- Definir os serviços a serem prestados pelo escritório de projeto. Obter o acordo do dirigente e do gerente de projeto sobre os serviços.
- Definir as competências e os papéis para o pessoal do escritório de projetos.
- Definir e anunciar o início do escritório de projetos.
- Trabalhar com dirigentes e gerentes de projeto para entender suas necessidades e atendê-las.
- Refinar as habilidades e os papéis do escritório de projeto à medida que o envolvimento com seus clientes cresce.

16.2.16 Principais clientes de um escritório de projetos

São dirigentes, gerentes ou líderes de projeto, membros da equipe do projeto, gerentes funcionais, *stakeholders* (como recebedores dos produtos do projeto). A alta administração é o cliente natural de um escritório de projetos, uma vez que precisa de informações gerenciais do projeto para tomar decisões estratégicas. As equipes dos projetos são especialmente e temporariamente alocadas para os projetos, precisando receber todas as informações necessárias sobre o projeto, bem como reportar o andamento de suas atividades periodicamente ou sempre que solicitado. Os gerentes funcionais são os que demandam projetos para suas áreas, compartilhando recursos entre as atividades diárias que sua área desempenha e os projetos que estão sendo realizados.

16.2.17 Processos da gerência de projetos

Projetos são compostos de processos. Resumindo, podemos dizer que processo é uma série de ações que geram resultados. Desta forma, os processos dos projetos são realizados por pessoas e normalmente se enquadram em duas categorias:

- **Processos orientados ao produto:** especificação e criação dos produtos do projeto.
- **Processos da gerência de projetos:** descrição, organização e trabalho do projeto.

Durante todo o projeto, existe uma interação e uma sobreposição entre os processos da gerência de projetos e os processos orientados a produto.

- **Definição de processo de gerência de projetos:** paradigma segundo o qual podem ser levadas a efeito as funções administrativas de planejamento, organização, motivação, direção e controle, conforme mostra a Figura 16.3.

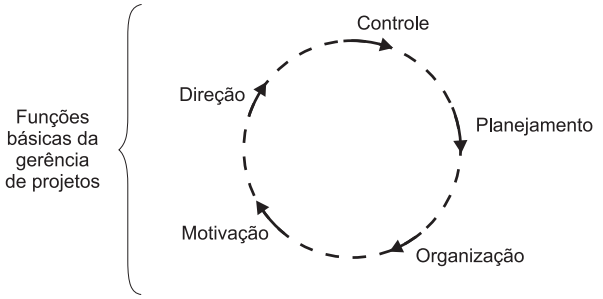


Figura 16.3 - Paradigma dos processos de gerência.

A seguir resumimos o propósito de cada função básica que integra a gerência de projetos:

Planejamento: pergunta básica ao pensarmos no planejamento do projeto: ***qual é o nosso alvo e por quê?*** Na execução do planejamento de um projeto, a missão da organização é usada como o ponto básico para determinar os objetivos, as metas e as estratégias do projeto.

Durante o processo de planejamento são estabelecidas as políticas, os procedimentos, as técnicas e documentações necessárias para dar forma à utilização prevista de recursos que levem a cabo os propósitos do projeto.

Organização: pergunta básica ao pensarmos na organização do projeto: ***o que está envolvido e por quê?*** No desempenho dessa função, determinam-se os recursos humanos e materiais e estabelecem-se os modelos desejados de autoridade e responsabilidades.

Motivação: pergunta básica ao pensarmos em estratégias para a motivação da equipe do projeto: ***o que provoca o melhor desempenho dos membros das equipes de projetos e das outras pessoas que lhes dão apoio?***

Direção: pergunta básica ao pensarmos nos envolvidos com a gestão do projeto: ***quem decide o que e quando?*** No cumprimento dessa função, os gerentes de projeto e outros administradores preparam-se para uma liderança direta sobre a supervisão e execução das decisões envolvidas no compromisso de recursos de um projeto.

Controle: pergunta básica ao pensarmos no controle do projeto: ***quem julga os resultados e mediante quais padrões?*** Nessa função, o administrador de projetos e outros administradores executam a monitoração, a avaliação e o controle do emprego de recursos que apoiam o projeto.

16.3 Fatores de sucesso e fracasso

É importante poder antecipar se um projeto terá sucesso ou fracassará. Em qualquer das duas situações, isso só pode de fato ser descoberto por intermédio das medidas de avaliação aplicadas quando o projeto tiver terminado. Contudo, certas providências podem evitar resultados negativos. No contexto da gerência de projetos, a palavra sucesso é empregada quando se quer exprimir que foi alcançada alguma coisa que se desejava, ou seja, a entrega do projeto acontece no prazo certo, dentro do orçamento, adequado aos objetivos, metas da empresa etc.

A palavra fracasso descreve a condição inversa, quando os objetivos esperados não são alcançados. No entanto, um projeto pode ser considerado aceitável quando os resultados relativos aos custos e aos prazos não tenham sido aqueles previstos. Para determinar o sucesso ou o fracasso, é necessário que, no decorrer do projeto, os padrões de desempenho sejam desenvolvidos e comparados aos resultados que vão sendo apresentados. Há alguns padrões gerais usados para julgar se um projeto alcançou ou não os objetivos esperados:

Alguns fatores que evidenciam sucesso

- O trabalho do projeto foi cumprido de acordo com o prazo e orçamento;
- Os resultados do projeto foram entregues ao cliente, que os considera adequados aos objetivos e às metas da empresa;
- Os *stakeholders* estão satisfeitos com o modo pelo qual o projeto foi administrado e com os resultados apresentados;
- Conseguiu-se lucro com o trabalho executado;
- O trabalho do projeto resultou em alguns avanços tecnológicos que prometem dar à empresa uma vantagem competitiva.

Alguns fatores que evidenciam o fracasso

- O projeto excedeu os custos e/ou a programação;
- O projeto não é adequado à missão, aos objetivos e às metas da empresa;
- Permitiu-se que o projeto fosse além do ponto em que os resultados seriam necessários para cumprir as expectativas do cliente;
- Foram aplicados processos inadequados de gerência de projetos;
- Foi feito um desenho errado dos padrões técnicos de desempenho do projeto;
- A alta administração não deu suporte ao projeto;
- Pessoas não qualificadas trabalharam na equipe do projeto;
- O projeto cumpriu as exigências, mas não resolveu as necessidades do negócio;
- Os *stakeholders* ficaram insatisfeitos com o andamento do projeto e/ou com os resultados obtidos.

Observar que cada projeto tende a ser único, podendo haver razões extras de sucesso ou de fracasso. Se tivermos consciência de tais fatores, aumentamos a probabilidade de sucesso e reduzimos as chances de falhar. Para ilustrar as preocupações que devemos ter com relação ao fracasso de um projeto, a seguir exemplificamos algumas questões-chave sobre este tema.

- A organização tem desenvolvido padrões de desempenho para o projeto, que possam ser usados como critério para determinar seu sucesso ou fracasso?
- A equipe de projetos, os gerentes e a alta administração compreendem quais são os fatores prováveis de sucesso e de fracasso?

- O que já foi realizado pelo projeto sofre uma revisão regular para determinar a presença dos fatores de sucesso ou de fracasso?

16.4 Liderança de projetos

Todo gestor é gestor de pessoas. Desta forma, é necessário observar aspectos relacionados à liderança que deve ser exercida sobre a equipe envolvida. Convém lembrar que a liderança está relacionada ao conceito de autoridade, que é o direito ou poder de fazer obedecer, de dar ordens, de tomar decisões, de agir etc. Para entender aspectos da liderança de projetos, é importante entender outros conceitos, como o de gerente e o de líder. Gerente é aquele que administra ou dirige um empreendimento. Já o líder é aquele que guia, vai ou está à frente de alguém, leva alguém em uma direção etc. Eis aí o motivo de tanta preocupação e estudo sobre liderança, sobre equipes de trabalho etc.

Como podemos observar, tanto a gerência quanto a liderança são necessárias em uma organização. Não há superioridade de uma sobre a outra: elas são complementares, e uma forma não substitui a outra. A gerência trabalha com complexidade, enquanto a liderança promove mudanças. O gerente planeja, orça, estabelece objetivos, cursos de ação e aloca recursos, enquanto o líder aponta uma direção, uma visão do futuro e a estratégia para conseguir a mudança. O gerente define os objetivos de seu empreendimento, cria uma organização e preenche as funções com pessoal qualificado, difunde o plano, distribui tarefas e controla o trabalho. O líder alinha pessoas, isto é, aponta a nova direção aos que entendem a visão do projeto e estão comprometidos com sua obtenção.

O gerente assegura o cumprimento de seu plano pelo controle da execução e resolução de problemas e conflitos. Para o líder, alcançar a visão requer motivação e inspiração. Consegue isso movimentando a equipe na direção certa, a despeito dos obstáculos à mudança, e fazendo apelo para as necessidades, os valores e emoções básicas humanas. O gerente de projeto pode ter, muitas vezes, uma especialização diferente da do executante. Por isso, pode julgar e valorizar seu pessoal apenas tendo em vista os resultados obtidos. Portanto, deve-se procurar reconhecer líderes potenciais e desenvolver a liderança nos gerentes, o que resultará em superpor as duas formas de autoridade: a de direito e a de fato, fazendo com que as organizações passem a desenvolver gerentes-líderes.

16.4.1 Líder de projeto

Vamos agora entender melhor as características de um líder de projeto, aquele que lidera ao longo do ciclo de vida do projeto, a fim de atingir os objetivos técnicos dentro do prazo e do orçamento planejados. Para liderar qualquer plano organizacional, exige-se tanto presença quanto processo, por isso os líderes, de uma maneira geral, devem possuir várias características importantes, descritas a seguir:

- Saber como agir e o que fazer;
- Ser visíveis aos membros da equipe (sua liderança é reconhecida) e estar à frente de tudo;
- Colocar-se à disposição para ouvir, debater e coletar informações dos membros da equipe, visando a tomada e execução de decisões;

- Ser capazes de dizer "mãos à obra!" no momento certo;
- Ser decididos e manter o registro de acompanhamento da elaboração e execução das decisões corretas;
- Buscar as melhores habilidades dos membros da equipe;
- Trabalhar para simplificar as coisas e evitar torná-las complexas;
- Ser justos e pacientes;
- Trabalhar com afinco para desempenhar bem o papel de líderes.

Quando enfocamos a liderança relacionada ao desenvolvimento de projetos, outras características do líder podem ser destacadas, conforme a relação a seguir:

- Desenvolve e vende uma visão para o projeto;
- Confronta com as mudanças operacionais e estratégicas do projeto;
- Constrói redes de troca de informações recíprocas, com os *stakeholders* mais relevantes;
- Desenvolve um ambiente cultural para a equipe do projeto, de modo a facilitar o comprometimento e a motivação dos membros;
- Estabelece a direção geral do projeto, em colaboração com os *stakeholders*;
- Entende as grandes questões que tendem a impactar o projeto e, depois, trabalha com os membros da equipe para acomodá-las;
- Torna-se símbolo do projeto e seu propósito;
- Torna-se o principal defensor do projeto em contato com os *stakeholders*.

Vamos agora enfocar as características de um gerente de projeto que, além de liderar, deve desempenhar outros papéis:

- Enfrentar a complexidade de desenvolver e implementar um sistema gerencial para o projeto;
- Manter uma visão geral do uso eficiente e eficaz dos recursos destinados ao projeto;
- Planejar e desenvolver as funções gerenciais de planejamento, organização, motivação, direção e controle dentro do contexto do sistema de gerência de projeto;
- Reprogramar os recursos, na medida do necessário, para manter o equilíbrio do apoio ao projeto;
- Monitorar a competência dos membros do projeto, para poder orientá-los, buscando o aprimoramento do conhecimento, habilidades e atitudes;
- Assegurar eficácia nos processos de comunicação que envolvem o projeto;
- Manter o acompanhamento, a fim de que os processos de avaliação do monitoramento e do controle sejam executados.

Portanto, os gerentes de projeto devem tanto liderar quanto gerenciar um projeto, mas para desempenhar tais papéis, exige-se competência nas seguintes habilidades:

- Ter compreensão geral da tecnologia envolvida no projeto;
- Ter habilidades interpessoais que promovam um ambiente cultural para a equipe e os *stakeholders* do projeto, de modo a refletir confiança, lealdade, comprometimento e respeito;
- Compreender os processos gerenciais e suas aplicações ao projeto;
- Ser capaz de tomar e implementar decisões acerca do projeto.

Observe que a competência em atuar tanto como líder quanto como gerente de projeto depende de conhecimento, habilidades e atitudes do indivíduo.

16.4.2 Ponto de vista da competência

É pelo ensinamento que os líderes conduzem seus colaboradores. Liderar não se resume a dar ordens e exigir obediência ou simplesmente ditar comportamentos específicos. O líder deve permitir que os outros saibam discernir sobre as ações que precisam ser tomadas para levar a organização para o fim desejado. Para que isso ocorra, um líder deve assumir a postura de um educador, propiciando meios para que os indivíduos se motivem para um fim específico. Líderes vencedores consideram educar um de seus principais papéis, pois desta forma atingem seus objetivos por meio das pessoas a quem ensinam. Isso exige experiência. Experiência, preparo técnico e senso de inovação são ingredientes que se misturam e se completam, concebendo competência ao líder. Para ilustrar as características desejadas e não desejadas de um líder, apresentamos na Tabela 16.6 um contraste entre líderes competentes e incompetentes.

Líder competente	Líder incompetente
Atitude positiva; reconhecimento.	Usa a sua posição de autoridade/título para dirigir pessoas - ele não compreende ou resolve o problema.
Interessado nos aspectos pessoais dos funcionários (situação familiar etc.), antecipa-se às preocupações (problemas) antes que elas se tornem evidentes. Modelo exemplar; decidido.	Não escuta, ignora ou rejeita todas as colaborações que não são politicamente aceitáveis; muda o escopo ou a direção à mercê da sua vontade, enquanto culpa os outros pelos erros.
Comunica claramente a visão da meta a ser alcançada, desafia e motiva os funcionários. Um gerente "orientado para os resultados".	Não pede ajuda; não estabelece um exemplo para seus seguidores; não conhece os aspectos técnicos do processo.
Tem uma visão do futuro do negócio, comunica essa visão às pessoas envolvidas no negócio e permite que os funcionários envolvidos façam contribuições em prol das metas do negócio.	Deixa os gerentes cuidarem do negócio, de maneira indisciplinada. Não aborda os problemas quando eles surgem. Preocupa-se somente com o resultado final. Não faz elogios, só críticas.
Demonstra confiança, fornece apoio e disposição para levar as culpas que porventura lhe forem imputadas e também para sofrer os eventuais desapontamentos; confia e apoia a equipe.	Não explica as razões de mudanças. Presta pouca atenção à implementação.

Líder competente	Líder incompetente
Ajuda os subordinados a estabelecerem a direção do trabalho e permite que eles cresçam rumo a essa direção. É um mentor, não um ditador. Sabe o que todos estão fazendo.	Não ouve as ideias dos outros. Não sabe criticar construtivamente. Espera somente a perfeição. Não reconhece ou elogia um trabalho benfeito. Desencoraja o pensamento criativo e ideias novas.
É sensível aos efeitos das decisões sobre todos os envolvidos. Enfatiza o trabalho em equipe. Reconhece as contribuições individuais e a dos grupos. Faz um esforço contínuo para relacionar-se bem com seus subordinados.	Totalmente concentrado na autopromoção. Não tem entusiasmo. Não consegue comunicar a visão ou as ideias.
Ouve os pensamentos e as ideias dos subordinados. Traz harmonia para departamentos que têm disputas históricas entre si. Não fica somente sentado dentro do escritório, também vai em campo verificar o trabalho.	Não é orientado para pessoas. Demonstra falta de interesse. Não tem uma visão geral de condução do trabalho e/ou não demonstra competência ou interesse para implementar uma.
Trata os colegas de trabalho como seres humanos, em vez de apenas uma peça qualquer do sistema.	Comunica-se por gritos, acenos e gestos, e traz no rosto um ar insatisfeito.

Tabela 16.6 – Líderes competentes x incompetentes.

16.4.3 Ética na gerência de projetos

Outro aspecto complementar ao da liderança é a ética, que na gerência de projetos abrange muitas áreas de conduta pessoal e profissional. Devemos considerar que podemos ter no quadro de integrantes de determinado projeto pessoas que trabalham em países diferentes, com aspectos culturais próprios e sistemas diversos de valores. Em alguns países, por exemplo, pode ser que culturalmente se espera e se exige o suborno para assegurar a continuidade de trabalho num projeto. Considerando essa possibilidade, percebemos que a ética não é universalmente compreendida em todos os ambientes nem há concordância a respeito dela. Em geral a ética pode depender da situação e ser aplicada livremente, ou pode ser rigorosa e exigente para o pessoal do projeto. Outra questão é que a falta de um código de ética, ou do treinamento em ética, também pode afetar o modo pelo qual o pessoal responde a situações de desafio. Também devemos considerar que a aplicação aleatória ou desigual de um código de ética pode prejudicar a ética praticada.

Atualmente, para todas as profissões reconhecidas é exigido um código de ética. Espera-se, desta forma, que os profissionais tenham liberdade de afirmar seus valores e que vivam de acordo com essas afirmações de conduta ética. Para ser eficaz, um código de ética para um grupo profissional precisa ser divulgado e aplicado. Os administradores de projeto também podem ter, dentro de suas organizações, um código de conduta que sirva de guia na maneira pela qual trabalham com os outros.

O mais conhecido código de conduta para administradores de projeto é o *Project Management Professional Code of Ethics* (código de ética do profissional de gerência de projetos), criado em 1983 pelo *Project Management Institute* (PMI) como parte de seu Programa de Certificação do Profissional de Gerência de Projetos (*Project Management Professional* [PMP] *Certification Program*). Com essa iniciativa, as pessoas que obtêm certificados profissionais de gerência de projetos devem concordar e aceitar o código de ética da PMP e apoiá-lo.

Quando um código de ética não é adotado por um indivíduo ou grupo de indivíduos, não ficam definidas as fronteiras da conduta ética. A variação por parte dos indivíduos, na prática e no cumprimento da conduta, chegará a um ponto em que não haverá prática coerente. A ética aleatória e que varia de acordo com a situação não apoia o profissionalismo nem cria nos demais uma confiança de que o grupo coletivo se atenha a quaisquer regras de conduta.

16.5 Capacitação dos envolvidos

Como indivíduos, não estamos a salvo do turbilhão constante de modificações que nos cercam. Por exemplo, a expectativa de vida, que não passava de 14 anos para os habitantes das cavernas, dobrou para 28 anos nos tempos do Império Romano, atingiu 36 anos no início do século XX, 70 anos na década de 1950 e hoje vai além dos 80 anos nos países mais desenvolvidos. Ao mesmo tempo, o número de horas da jornada de trabalho anual vem caindo:

No Brasil de 1913, eram de cerca de 3.016 horas no setor urbano. Em 1996, eram 1.372 na Holanda, 1.560 na Alemanha, 1.645 na França e 951 nos EUA. Nestes mesmos países, em 1891, a jornada estava entre 2.900 e 3.000 horas. Por problemas trabalhistas, está decaindo em todo o mundo. Os dois fatores citados (duração da vida e tempo dedicado ao trabalho) inverteram a relação tempo de trabalho/tempo disponível, o que leva os homens a procurar serviços de lazer.

Esse limitado tempo disponível precisa ser administrado! Mas por quem? Certamente que é pelo próprio indivíduo e não por uma coleção de entidades como clube, agência de turismo, academias de ginástica etc. Em todas as idades o tempo disponível deve ser programado, planejado, executado e controlado como uma empresa. Com muito maior razão, a profissão deve ser cuidadosamente tratada. A prática corrente, quanto à formação profissional, era fazer um curso que servisse para toda a vida, como qualificação profissional. Hoje, a vida útil de um curso é extremamente curta.

É imprescindível uma atualização constante, quando não a completa mudança de campo de atuação. O emprego, como o conhecíamos há alguns anos, está sendo substituído pelo trabalho. O treinamento dos trabalhadores está sendo assunto de importância estratégica para as empresas. Muitas até já criaram suas Universidades Empresariais como a Motorola University, a Disney University, a Sears University e a McDonald's Hamburguer University.

16.5.1 Desenvolvimento da equipe

Muitos estudos têm sido feitos para formar equipes eficientes e delas obter o máximo resultado. Esta é uma das importantes preocupações do gerente de projetos. Afinal, o sucesso ou o desastre muito devem à natureza da equipe, aos seus componentes e ao trabalho conjunto. A criação dessas equipes é, atualmente, um dos grandes desafios e consiste em transformar um conjunto de pessoas, de formação, especialização e história profissional diferentes em um conjunto voltado a alcançar um objetivo comum, executando trabalho cooperativo. A partir do início da formação de uma equipe, é de se esperar que o grupo deve passar por algumas fases de evolução, sob a coordenação do gerente.

Identificação da necessidade de treinamento

Requer que os gerentes seniores e de área reconheçam dois fatores críticos:

- O primeiro, que o treinamento é uma das maneiras mais rápidas de consolidar conhecimento;
- Segundo, que o treinamento deve ser realizado em benefício do pessoal menos qualificado da empresa para o aperfeiçoamento de sua eficiência.

Identificar a necessidade de treinamento tornou-se, de certa forma, mais fácil nos últimos dez anos devido aos estudos de casos dos benefícios proporcionados pelo treinamento em gestão de projetos. Esses benefícios podem ser classificados como **qualitativos** e **quantitativos**.

Benefícios quantitativos

- Menor tempo para o desenvolvimento de produtos;
- Decisões mais rápidas e qualificadas;
- Redução dos custos;
- Aumento das margens de lucros;
- Redução da necessidade de pessoal;
- Redução da burocracia;
- Melhoria da qualidade e confiabilidade;
- Redução da rotatividade de pessoal;
- Implementação mais rápida das "melhores práticas" etc.

Benefícios qualitativos

- Melhor visibilidade e foco nos resultados;
- Melhor coordenação;
- Melhor controle;
- Moral elevado;
- Aumento do apoio dos funcionários;
- Melhor relação com o cliente etc.

16.6 Exercícios

1) Como as estratégias empresariais normalmente são obtidas?
2) Quais são as características básicas que um *sponsor* do projeto deve ter?
3) Uma empresa deve ter processos para identificação, quantificação, desenvolvimento de respostas e controle do risco. Quais são os componentes básicos de um risco?
4) O que envolve o alinhamento estratégico de projetos e como deve ser cumprido?
5) Com relação à departamentalização tradicional, quais os meios tradicionais mais comumente usados para descentralizar autoridade e obrigação de prestar contas?
6) O que normalmente motiva a implantação de um escritório de projetos?
7) No contexto da gerência de projetos, quando as palavras sucesso e fracasso normalmente são empregadas?
8) Quais as características mais importantes que um líder de projeto deve ter?
9) Como podemos classificar os benefícios proporcionados pelo treinamento em gestão de projetos?

Anotações

17

Outras Metodologias Ágeis e Considerações Finais

Objetivos

- *Apresentar uma visão geral de outras metodologias ágeis não abordadas neste livro.*
- *Tecer considerações finais sobre os temas apresentados anteriormente.*

17.1 Metodologias ágeis não abordadas neste livro

Este capítulo é dedicado a algumas metodologias cujas especificações não foram exploradas nesta obra, mas que merecem serem citadas pela colaboração ao movimento ágil de desenvolvimento.

17.1.1 OpenUP - Processo Unificado Aberto

Trata-se de uma metodologia com abordagem iterativa e incremental para conduzir projetos de software. Foi criada em 2005 pela *Enterprise Foundation* em conjunto com a *IBM Rational Software*. Preza pela qualidade encontrada no modelo RUP, com a agilidade da metodologia *Extreme Programming* e com as melhores práticas de gerenciamento do método SCRUM. Valoriza e colabora com a equipe de projeto, trazendo inúmeros benefícios aos clientes em vez da formalidade desnecessária. Está dividido em quatro grandes áreas, sendo comunicação e colaboração, objetivo, solução e gerência, as quais atuam iterativamente por meio de ciclo de vida do projeto baseado em suas disciplinas:

- Análise e Projeto;
- Gerência de Configuração e Mudança;
- Implementação;
- Gerência de Projetos;
- Requisitos;
- Teste.

Maiores informações podem ser encontradas no endereço web www.eclipse.org/epf, que mantém atualizado o conteúdo sobre essa metodologia.

17.1.2 Pragmatic Programming (PP)

Criada em 2000, por Andrew Hunt e David Thomas, essa metodologia de desenvolvimento ágil é caracterizada por incluir um conjunto das "melhores práticas" de programação, procurando abranger o que há de melhor em outras metodologias ágeis. Suas práticas têm uma perspectiva pragmática de desenvolvimento iterativo com foco incremental. Utiliza em seus projetos uma bateria de testes rigorosos e procura conduzir projetos de sistemas centrados no usuário. Não contém processos, fases ou papéis, e sim dicas que focam dificuldades de programação do dia a dia. Considera-se que essas dicas podem ser utilizadas em qualquer método de desenvolvimento de software. Maiores informações podem ser encontradas no endereço web a seguir, inclusive as dicas abordadas:

http://www.codinghorror.com/blog/2004/10/a-pragmatic-quick-reference.html

17.1.3 Test Driven Development (TDD)

O *Test Driven Development* (ou Desenvolvimento Orientado a Testes) é uma técnica de desenvolvimento orientada a testes que busca antecipar a identificação e correção de falhas durante o processo de desenvolvimento de software. Foi proposta por Kent Beck, em 2003, com o objetivo de encorajar o desenvolvimento de códigos simples e que inspirem confiança. Baseia-se em pequenas iterações, que começam pela implementação de um caso de teste, depois pelo código necessário para executar o teste e terminando com o aperfeiçoamento do código criado, objetivando acomodá-lo a mudanças que foram feitas.

É importante ressaltar que o TDD não é um método para testar software, mas sim para construí-lo. Foi inspirado no conceito de "*test-first programming*" da XP. Essa metodologia é conduzida pelos chamados "testes de unidade", e também pode ser aplicada a testes de integração.

Também é bom não confundir com os testes de sistema (caixa-preta) ou testes de aceitação que são feitos por usuários finais. O TDD apresenta vários benefícios, como a eliminação do "medo" relacionado à alteração de algo que funciona, considerando que um novo problema pode ser introduzido. Essa metodologia também nos leva a produzir componentes de software mais desacoplados, aspecto que garante o isolamento dos testes.

17.1.4 Lean Software Development (LD)

Essa metodologia foi inspirada em metodologias desenvolvidas pela Toyota, que tinham por objetivo definir boas práticas para guiar processos industriais de linha de montagem. Tem como foco a eliminação de desperdício, a excelência na qualidade e também o aumento da velocidade dos processos. A filosofia Lean, proposta inicialmente, tem a finalidade de manter as coisas certas, no lugar certo e na hora certa. Oferece um conjunto de princípios que podem ser utilizados por qualquer organização, estimulando a cultura de trabalhadores generalistas (os que possuem diversas habilidades de interesse corporativo), além de possibilitar a diminuição de estoques. Em 2003, foi proposta a adaptação para o desenvolvimento de software, originando o termo *Lean Software Development*, por ocasião da publicação de um livro com o mesmo nome, por Tom e Mary Poppendieck.

17.1.5 Microsoft Solutions Framework (MSF) for Agile Software Development

Essa metodologia de desenvolvimento surgiu em 1994, como um conjunto de boas práticas compiladas pela Microsoft a partir de sua experiência na produção de software e em serviços de consultoria, evoluindo para um *framework* flexível para nortear projetos de desenvolvimento de software. O MSF 4.0 foi publicado em duas linhas diferentes: o *MSF for Agile Software Development* e o *MSF for CMMI Process Improvement*, que segue uma filosofia de desenvolvimento mais tradicional. O MSF ágil é muito mais do que um conjunto de atividades a serem seguidas, pois busca a criação de uma cultura organizacional que encoraje projetos bem-sucedidos. Ela se baseia no conceito de Mindset, uma coleção de valores que determina como os indivíduos vão interpretar as situações e reagir perante elas. Essa metodologia ágil prega a formação de equipes de pares, que não segue uma organização hierárquica top-down tradicional, mas um modelo para uma equipe colaborativa com responsabilidades compartilhadas.

17.2 Considerações finais

Considerando os aspectos apresentados nesta obra, podemos verificar que o desafio futuro das metodologias ágeis concentra-se em encontrar maneiras de eliminar pontos fracos observáveis, como, por exemplo, a falta de análises de risco. Essas evoluções não podem fazer com que as metodologias ágeis se descaracterizem, tornando-as pesadas.

Outro ponto é o desafio de implantar metodologias ágeis em grandes empresas, considerando seu aspecto inicial de ser proposta para pequenos grupos de desenvolvedores. Para este cenário, problemas de comunicação interna entre a equipe de desenvolvimento devem ser resolvidos, principalmente ao considerarmos a possibilidade de os colaboradores estarem geograficamente dispersos.

Talvez a ausência de publicações que relatem casos de sucesso envolvendo metodologias ágeis em projetos grandes e críticos esteja influenciando grandes corporações na hora de optar pelo uso dessas metodologias. Todas as metodologias de desenvolvimento de software apresentam vantagens e desvantagens, limitações e restrições. Contudo, percebe-se, por meio de resultados empíricos, que o uso de metodologias ágeis apresenta excelentes resultados em termos de qualidade, confiança, prazo de entrega e custo. Certamente estudos complementares são necessários, considerando projetos com características diferentes, equipes específicas etc., mas atualmente a adoção de metodologias ágeis tem sido uma tendência mundial.

Anotações

Bibliografia

BACK, K; FOWLER, M. **Planning Extreme Programming**. 1. ed. Addison-Wesley Professional, 2000.

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. **The Unified Modeling Language**: *User Guide*. EUA: Addison Wesley, 1999.

COCKBURN, Alistair. **Crystal Clear: A Human-Powered Methodology for Small Teams:** *A Human-Powered Methodology for Small Teams.* 1.ed. Addison-Wesley Professional, 2004.

CRISPIN, L.; HOUSE, T. **Testing Extreme Programming**. Addison Wesley, 2005.

CRUZ, T. **BPM & BPMS**: *Business Process Management & Business Process Management Systems*. São Paulo: Brasport, 2008.

DEBONI, J. E. Z. **Modelagem Orientada a Objetos com a UML**. São Paulo: Futura, 2003.

DELAMARO, M.; MALDONADO, J. C; JINO, M. **Introdução ao Teste de Software**. Campus, 2007.

HIGHSMITH, J. **Adaptive Software Development:** *A Collaborative Approach to Managing Complex Systems*. Dorset House, 1999.

_____. **Agile Project Management:** *Creating Innovative Products*. Addison-Wesley Professional, 2004.

HUMPHREY, W. S. **Introduction to the Personal Software Process**. 1.ed. Addison-Wesley Professional, 1996.

_____. **Introduction to the Team Software Process**. 1. ed. Addison-Wesley Professional, 1999.

MALDONADO, J. C. **Critérios Potenciais Usos**: *Uma Contribuição ao Teste Estrutural de Software*. **PhD thesis.** DCA/FEE/UNICAMP. Campinas, SP, July 1991.

MANTIS BUG TRACKER. MantisBT Group. GNU General Public License (GPL), 2000-2012.

PALMER, S. R.; FELSING, Mac. 2001. **A Practical Guide to Feature-Driven Development.** 1. ed. Pearson Education.

PRESSMAN, R. S. **Engenharia de Software**. Rio de Janeiro: McGraw Hill, 2002.

PROJECT MANAGEMENT INSTITUTE - PMI. *Guia PMBok: Um guia do conjunto de conhecimentos do gerenciamento de projetos.* 3. ed. Pennsylvania: Project Management Institute, 2004.

ROSENBERG, D.; STEPHENS, M. **Use Case Driven Object Modeling with UML Theory and Practice.** Apress, 2007.

SBROCCO, J. H. T. C. **UML 2.3:** *Teoria e Prática*. São Paulo: Érica, 2011.

SO/IEC 14764. **Software Engineering - Software Life Cycle Processes - Maintenance - International Standards for Business**, Government and Society, 2006

SOMMERVILLE, I. **Engenharia de Software**. São Paulo: Prentice Hall Brasil, 2003.

STAPLETON, J. **Business Focused Development, The DSDM Consortion.** 2. ed. Person Education, 2003.

VALERIANO, D. L. **Gerenciamento estratégico e administração por projetos**. São Paulo: Makron Books, 2001.

VALLE, T.; O., S. B. **Análise e Modelagem de Processos de Negócio**. São Paulo: Atlas, 2009.

Artigos e Dissertações

Carvalho, B. V. (2009) **Aplicação do Método Ágil Scrum no Desenvolvimento de Produtos de Software em uma Pequena Empresa de Base Tecnológica**. Dissertação de Mestrado - Universidade Federal de Itajubá.

CAVAMURA JR, L. (2008) **Aqua - Atividade de Qualidade no Contexto Ágil**. Dissertação de Mestrado - Universidade Federal de São Carlos.

Comparação Entre os Processos dos Métodos Ágeis: Xp, Scrum, Fdd e Asd em Relação ao Desenvolvimento Iterativo Incremental. E-Tech: Atualidades Tecnológicas para Competitividade Industrial, Florianópolis, v. 1, n. 1, p. 37-46, 1. sem., 2008.

Web Sites

ADAPTIVE SOFTWARE DEVELOPMENT. Disponível em: <www.adaptivesd.com>. Acesso em: janeiro de 2012.

AGILE ALLIANCE. *Manifesto for agile software development*. Disponível em: <http://www.agilemanifesto.org/>. Acesso em: 10 dez. 2011.

ATLANTIC SYSTEMS GUILD. *Template Volere*. Disponível em: <http://www.volere.co.uk/template.htm>. Acesso em: dezembro de 2011.

BOSTON GLOBE. Sysdeco Projeto. Disponível em: <www.dsdm.org>. Acesso em: fevereiro de 2012.

COCKBURN. Alistair. Disponível em: <http://alistair.cockburn.us/crystal>. Acesso em: janeiro de 2012.

DESIGNIT. Case de sucesso. Disponível em: <http://www.sitepoint.com/successful-development/>. Acesso em: janeiro de 2012.

DMV. Disponível em: <http://www.sparxsystems.com.au/downloads/pdf/dmv_csi_case_study.pdf>. Acesso em: dezembro de 2011.

DSDM. *Atern for Agile Project Management and Delivery*. <www.dsdm.og>. Acesso em: fevereiro de 2012.

EXTREME PROGRAMMING. Disponível em: <www.extremeprogramming.org>. Acesso em: dezembro de 2011.

ESCROW. Caso de sucesso. Disponível em: <http://www.agilelogic.com/files/ExtremeProgrammingPerspectivesCh30.pdf>. Acesso em: dezembro de 2011.

FEATURE DRIVEN DEVELOPMENT. FDD, FDDI, the FDD - trademarks of Nebulon Pty. Ltd. Disponível em: <www.featuredrivendevelopment.org>. Acesso em: janeiro de 2012.

ICONIX. *Software Engineering, UML and SysML training for Enterprise Architect and MagicDraw*. Disponível em: <www.iconixsw.com>. Acesso em: dezembro de 2011.

JAVA DEVELOPER JOURNAL. Disponível em: <http://www2.sys-con.com/itsg/virtualcd/java/archives/0511/Westra/index.html>. Acesso em: dezembro de 2011.

LSST. Disponível em: <http://iconixsw.com/Articles/LSST_case_study_20080825.pdf>. Acesso em: dezembro de 2011.

OBJECTMENTOR - Caso de sucesso. Disponível em: <http://www.objectmentor.com/resources/articles/Primavera.pdf>. Acesso em: dezembro de 2011.

SCOTTISH NATURAL HERITAGE E NEWELL & BUGDE. Disponível em: <http://www.snh.org.uk/pdfs/publications/commissioned_reports/F02CI01.pdf>. Acesso em: fevereiro de 2012.

SEI. *Software Engineering Institute* - Carnegie Mellon. Disponível em: <http://www.sei.cmu.edu/psp/psp.html>. Acesso em: março de 2012.

SEMA GROUP. Disponível em: <http://www.semagroup.com.au/>. Acesso em: fevereiro de 2012.

SIG. Disponível em: <http://www.sparxsystems.com.au/downloads/pdf/esri_lbs_delivery_case_study.pdf>. Acesso em: dezembro de 2011.

TANGIBLESOLUTIONS. Case de sucesso. Disponível em: <http://www.methodsandtools.com/archive/archive.php?id=19>. Acesso em: janeiro de 2012.

Marcas registradas

Java e todas as marcas registradas e logotipos baseados em Java são marcas registradas da Sun Microsystems, Inc.

IBM e ibm.com são marcas comerciais ou registradas da International Business Machines Corporation nos Estados Unidos e/ou em outros países.

UNIX é uma marca registrada da The Open Group.

Microsoft Windows, Microsoft Word, Microsoft Access, Microsoft Visual Basic são marcas registradas da Microsoft Corporation.

Todos os demais nomes registrados, marcas registradas ou direitos de uso citados neste livro pertencem aos seus respectivos proprietários.

Índice remissivo

Símbolos

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

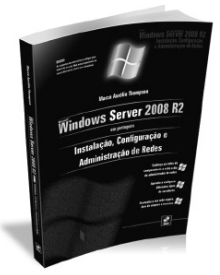
T

U

V

W

X



Windows Server 2008 R2 - Instalação, Configuração e Administração de Redes

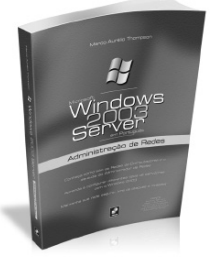
Autor: Marco Aurélio Thompson
Código: 3066 • 336 páginas • Formato: 17,5 x 24,5 cm • ISBN: 978-85-365-0306-6 • EAN: 9788536503066

Com linguagem simples e ilustrações que esclarecem passo a passo cada operação, este livro ensina desde os conceitos básicos até os mais avançados, para que estudantes e profissionais da área possam construir uma rede de computadores segura, de alta confiabilidade e fácil gerenciamento.
Traz uma visão geral do ambiente de redes e apresenta protocolos do TCP/IP, principais funções, infraestrutura com DHCP, WINS, DNS, diretivas de rede, Active Directory, servidor de arquivos e de fax, servidor de impressão e de web com o IIS, a virtualização com o Hyper-V, as diferentes formas de acesso remoto ao servidor, segurança da rede, como usar scripts de configuração com o PowerShell, além da descrição de diversas tarefas de administração da rede, com seus problemas mais comuns e as respectivas soluções.

Windows Server 2008 R2 - Fundamentos

Autor: Marco Aurélio Thompson
Código: 3240 • 192 páginas • Formato: 17,5 x 24,5 cm • ISBN: 978-85-365-0324-0 • EAN: 9788536503240

Estudantes e profissionais que desejam aprender os recursos fundamentais do Windows Server 2008 R2 encontram neste livro um conteúdo prático e exemplos passo a passo. Ele explica como transformar o micro em um laboratório para a prática de redes, criar a infraestrutura para uma pequena rede, as novidades e diferenças da nova versão, como decidir entre as opções de resolução de nomes, instalação profissional do Windows Server 2008 R2 e das funções e recursos e a administração remota do servidor.
Mostra as diferenças entre compartilhamento e servidor de arquivos, compartilhamento de impressora e servidor de impressão, como usar as diretivas de segurança, trabalhar com as ferramentas administrativas e o gerenciador de servidores.



Windows Server 2003 - Administração de Redes

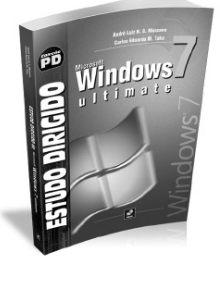
Autor: Marco Aurélio Thompson
Código: 9808 • 376 páginas • Formato: 17 x 24 cm • ISBN: 978-85-7194-980-5 • EAN: 9788571949805

Ensina a gerenciar o Windows 2003 Server em rede e mostra como realmente é o dia a dia do administrador.
Está organizado de forma didática, abordando conceitos básicos de redes, arquiteturas, protocolos e instalação da versão Server, os tipos de servidor que o Windows 2003 pode se tornar (controlador de domínio, servidor de arquivos, de impressão, DNS, WINS, DHCP, servidor Web (WWW e FTP) etc.), criação de uma Intranet, adotando uma política de segurança, além de dicas e macetes do Windows 2003 e orientações para certificação Microsoft.
É indicado aos profissionais e alunos da área de informática que desejam ingressar no lucrativo mercado de administração de redes.

Windows Server 2003 em português - Implementação e Administração

Autor: Eng. Francisco Baddini
Código: 9832 • 376 páginas • Formato: 17 x 24 cm • ISBN: 978-85-7194-983-6 • EAN: 9788571949836

Traz os procedimentos recomendados na implementação do Windows Server 2003. Trata desde os principais recursos de hardware existentes nos servidores até instalação, configuração, aplicações web, otimização e gerenciamento dos recursos da plataforma. Um capítulo especial aborda o Microsoft ISA Server 2000, com o qual é possível transformar o servidor numa poderosa solução de segurança. É destinado aos profissionais e alunos da área de informática que desejam implementar e entender como funcionam os principais recursos e produtos desenvolvidos para essa plataforma.



Estudo Dirigido de Microsoft Windows 7 Ultimate

Autores: André Luiz N. G. Manzano e Carlos Eduardo M. Taka
Código: 2663 • 176 páginas • Formato: 17 x 24 cm • ISBN: 978-85-365-0266-3 • EAN: 9788536502663

Com objetividade este livro mostra as principais características operacionais e a nova interface do Windows 7. Ensina a criar contas de usuário, gerenciar arquivos e pastas, dar direitos a usuários, além de algumas noções de Internet. Trata das terminologias básicas, CPU, memórias, periféricos, novidades da versão, área de trabalho, principais aplicativos e acessórios, Windows Explorer e desfragmentação de disco. Contempla assuntos importantes e atuais como segurança, tecnologia Bluetooth, Wireless, Windows Defender, Firewall e Windows Update.
É indicado a professores, alunos e autodidatas. Traz exemplos e exercícios didáticos para facilitar o aprendizado.

Microsoft Windows 7 Professional - Guia Essencial de Aplicação

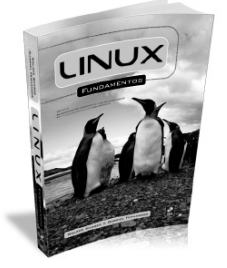
Autor: José Augusto N. G. Manzano
Código: 3035 • 296 páginas • Formato: 17,5 x 24,5 cm • ISBN: 978-85-365-0303-5 • EAN: 9788536503035

O Microsoft Windows 7 e seus recursos essenciais estruturam este livro prático e dirigido, que conceitua sistema operacional, explana a estrutura organizacional do Windows, instalação e configurações iniciais. Aborda acesso ao computador por portadores de necessidades especiais e descreve as ferramentas de produtividade e de sistema, como bloco de notas, calculadora, prompt de comando, impressoras etc., o programa Windows Explorer e recursos de multimídia. Inclui gerenciamento do sistema, bibliotecas e pastas do usuário.
Conectividade e acessibilidade à Internet, Windows Update, manutenção do sistema e outros recursos auxiliares, como o modo de compatibilidade com o Microsoft Windows XP, são assuntos tratados.

Estudo Dirigido de Windows Vista Ultimate

Autores: João Carlos N. G. Manzano e André Luiz N. G. Manzano
Código: 1871 • 208 páginas • Formato: 17 x 24 cm • ISBN: 978-85-365-0187-1 • EAN: 9788536501871

O Windows Vista é um sistema operacional totalmente inovador. O livro mostra as principais características operacionais e ensina a configurar a nova interface do Windows Vista Ultimate, criar contas de usuário, gerenciar arquivos e pastas, dar direitos aos usuários e ainda apresenta as principais mudanças que foram implementadas e algumas noções de Internet. A linguagem é simples e traz exercícios didáticos, possibilitando melhor acompanhamento dos assuntos abordados. É indicado a professores, alunos e autodidatas.



Linux - Fundamentos

Autores: Walace Soares e Gabriel Fernandes
Código: 3219 • 208 páginas • Formato: 17,5 x 24,5 cm • ISBN: 978-85-365-0321-9 • EAN: 9788536503219

Os conceitos e aspectos essenciais do Linux, sua instalação e gerenciamento básico são explanados de forma objetiva e prática. O livro aborda a história do programa, sua relação com o UNIX, estrutura, vantagens, aspectos práticos do dia a dia do administrador, as tarefas exigidas, ferramentas e comandos, gerenciamento de usuários e grupos, monitoramento do servidor, administração de serviços (daemons), execução da instalação de forma eficiente, os gerenciadores de boot comumente utilizados (GRUB e LILO) e dicas de segurança.
Detalha os sistemas de arquivos, a ferramenta Shell, protocolos, configuração, DHCP, SAMBA, NFS, compartilhamento de arquivos com servidores Windows, login remoto, SSH, roteamento, FTP, NTP, DNS e muito mais.

Montagem e Configuração de Computadores - Guia Prático

Autor: Renato Rodrigues Paixão
Código: 3196 • 304 páginas • Formato: 20,5 x 27,5 cm • ISBN: 978-85-365-0319-6 • EAN: 9788536503196

Indicado a estudantes, profissionais e entusiastas, o livro desmistifica o processo de montagem de um computador pessoal detalhando cada etapa, como configuração, testes de performance e interconexão numa rede local (LAN).
Aborda conceitos básicos de eletricidade e eletrônica digital, ferramentas e componentes, partes do PC, periféricos, ambiente de trabalho, aterramento e conexões, modelos de gabinete, motherboard, jumpeamento, overclocking e o processador.
Descreve pentes de memória, conexões a cabos, fonte de alimentação, HDD e FDD, controladoras, cabos internos e externos do gabinete, filtros de linha, estabilizadores de tensão e nobreaks. Ensina como ligar o PC pela primeira vez, configuração do setup e BIOS, janelas principais, boot do sistema, instalação do sistema operacional, placa de vídeo, impressoras, softwares básicos e de monitoramento de sinais, defrag e utilitários de desempenho.

Manutenção de Computadores - Guia Prático

Autor: Renato Rodrigues Paixão
Código: 3226 • 208 páginas • Formato: 20,5 x 27,5 cm • ISBN: 978-85-365-0322-6 • EAN: 9788536503226

Estudantes, entusiastas e profissionais da área encontram neste livro um conteúdo didático e objetivo sobre os cuidados necessários para a manutenção preventiva e corretiva de um PC.
De forma gradativa, aborda conceitos básicos de eletricidade, instrumentos de medida, principais componentes que integram um PC, microprocessadores, barramento, arquitetura interna, encapsulamentos, conexão do processador com a motherboard, memórias, chipsets, motherboards, drives (HDD, FDD, CD/DVD), conexão da BIOS, setup e controladoras de vídeo, como também aspectos práticos da manutenção, ferramentas e sobressalentes, ambiente de trabalho, cargas eletrostáticas, aterramento, conexões, filtros de linha, estabilizadores de tensão e nobreaks, cuidados com o PC, acessórios, limpeza, tipos de falha e solução de problemas.