E-Revista Facitec
Editorial, Artigos, Resumos e Resenhas
www.facitec.br/erevista

O PAPEL DO USUÁRIO NO EXTREME PROGRAMMING

THE USER'S ROLE IN EXTREME PROGRAMMING

Por:

Welder Maurício de Souza

E-Revista Facitec, v.2 n.1, Art.2, julho. 2008.

http://www.facitec.br/erevista/index.php?option=com_content&task=view&id=9&Itemid=2

O PAPEL DO USUÁRIO NO EXTREME PROGRAMMING

THE USER'S ROLE IN EXTREME PROGRAMMING

Resumo

O desenvolvimento de software tem passado por profundas mudanças no decorrer dos últimos trinta anos. Durante este tempo, a indústria de software priorizou a melhoria de processos e ferramentas, esquecendo-se do lado humanista envolvido em projetos de sistemas. No presente ensaio, discute-se o papel do demandante de desenvolvimento de software, especificamente, em relação ao extreme programming, destacando-se a prática da metáfora como recurso de comunicação com a equipe de desenvolvimento. Com base na literatura e em observações empíricas, são discutidas as principais características dos demandantes e, também, as atitudes tomadas por analistas de sistemas que podem afetá-lo, provocando diversos tipos de reações. Por fim, são apresentadas ações para melhorar esta relação.

Palavras-chave: usuário, desenvolvimento de sistemas, extreme programming.

Abstract

Software development has passed by deep changes in the last thirty years. During this time, software industry improved tools and processes, missing the human factors of system projects. This present essay, discusses user's paper in software development, specially related with extreme programming point of view, emphasizing metaphor practice as a communication resource with the development equip. Based on literature and empirical observations, are discussed the major features about users and taken some attitudes from system analysts that can affect him too, generating many kinds of reaction. In the end, some actions are proposed to take better to this relationship.

Keywords: users, system development, extreme programming.

INTRODUÇÃO

O processo de desenvolvimento de software tem passado por profundas transformações nestes últimos trinta anos. Embora as técnicas e ferramentas tenham evoluído, tal como a *Unified Modelling Language* (UML) e as linguagens orientadas a objetos, existe um elemento que tem sido marginalizado, o usuário.

O usuário tem se colocado como um elemento paradoxal neste processo. Ele contrata o desenvolvimento de software, mas os programadores não querem vê-lo por perto, já que não sabe se expressar e muda constantemente de idéia, dificultando o desenvolvimento. Por outro lado, existe, atualmente, um cenário específico referente às novas metodologias de desenvolvimento, conhecidas como metodologias ágeis, que estão aproximando novamente os usuários das equipes de desenvolvimento de sistemas.

Este ensaio descreve como o usuário, aqui denominado de demandante, quem contrata uma equipe de desenvolvimento, reage em relação ao ambiente de desenvolvimento de software. Especificamente, mostra seu relacionamento com uma equipe que usa a metodologia de desenvolvimento ágil *extreme programming.* Além disso, são abordados os seguintes aspectos:

- As características de usuários e analistas de sistemas, com base em revisões bibliográficas e em observações empíricas realizadas em análise de sistemas. Também, são apontados procedimentos e ações que geram confiança entre demandantes e desenvolvedores;
- A metodologia de desenvolvimento ágil *extreme programming*;
- O emprego da metáfora (prática XP) e sua eficácia como ferramenta de comunicação entre desenvolvedores e demandantes de software.

O COMPORTAMENTO DA INDÚSTRIA DE SOFTWARE

Segundo Franco (2007), a indústria de software é nova, se comparada ao setor de manufatura. A partir das décadas de 50 e 60 é que o termo "software" foi popularizado, durante o surgimento dos primeiros computadores pessoais (MEIRELLES, 1994). Paralelamente, surgiu uma demanda por projetos de software que não foi acompanhada pelo desenvolvimento de ferramentas e processos de desenvolvimento. Segundo Sebesta (2003), as linguagens de programação se encontravam em suas primeiras gerações. Trabalhavam com uma estrutura cheia de saltos intercalados dentro do código e com termos de difícil entendimento até mesmo entre os programadores, gerando a primeira crise do software no final da década de 60.

Durante este período até o início da década de 90, observaram-se muitos problemas relacionados aos projetos de software. O Instituto Standish Group (ISG) (1994), em pesquisa que envolveu 365 companhias de desenvolvimento e 8.380 aplicações analisadas, chegou aos seguintes resultados: 31% dos softwares foram cancelados antes do termino do projeto; 53% dos softwares foram finalizados com custos elevados e com um prazo muito maior do que aqueles combinados no início do projeto (Tabela 1).

Tabela 1: Fatores que influenciam os riscos de um projeto de software.

Fatores críticos para um projeto de software	%
1. Falta de especificação dos usuários	12,8
2. Requisitos incompletos	12,3
3. Mudança de requisitos	11,8
4. Falta de apoio executivo	7,5
5. Tecnologia imatura	7,0
6. Falta de recursos	6,4
7. Expectativas irreais	5,9
8. Objetivos obscuros	5,3
9. Tempo irreal	4,3
10. Tecnologia nova	3,7
11. Outros	2,3

Fonte: ISG, 1994.

Os dados apresentados na tabela 1 apontam que a falta de especificação dos usuários é o elemento de maior risco em um projeto de software. Se for considerado que os elementos dois e três da referida tabela têm como participante o usuário, pode-se afirmar que os riscos são elevados a 36,9% de todos os fatores que representam riscos no projeto de desenvolvimento de software.

Apesar desta informação, a indústria de software priorizou a melhoria de ferramentas e processos nos projetos. A criação da UML é um exemplo. Segundo Jacobson (2000), na metade da década de 70 existia uma grande quantidade de métodos de desenvolvimento orientados a objetos, paralelamente, as aplicações a serem construídas se tornavam mais complexas. Exatamente nesta época, três metodologias entre as existentes se destacaram: o *Objectory* de Booch, o OOSE de Jacobson e o OMT de Rumbaugh. Na década de 1990, indústrias como a Digital Equipment, Hewlett-Packard e Intel, entre outras, contribuíram como colaboradoras para o surgimento da UML, já que a padronização se tornava um fator fundamental para essas empresas.

Ao mesmo tempo em que o ferramental evoluía, evoluíram também as normas para a área de desenvolvimento de software. O Instituto de

Engenharia Elétrica e Eletrônica (IEEE) definiu um conjunto de boas práticas, materializadas por meio do livro SWEBOK, que apresenta as seguintes áreas de conhecimento (Quadro 1).

Requerimentos de software
Desenho de software
Construção de software
Testes de software
Manutenção de software
Gerência de configuração de software
Gerência de engenharia de software
Processo de engenharia de software
Métodos e ferramentas de engenharia de software
Qualidade de software

Quadro 1 – Áreas de conhecimento do SWEBOK
Fonte: SWEBOK (2004).

As áreas de conhecimento dentro do SWEBOK estão altamente inter-relacionadas e, dentro de cada área, existe uma coleção de subáreas que devem ser abordadas em um projeto de desenvolvimento. Neste conjunto de boas práticas do SWEBOK, existem basicamente duas áreas relacionadas diretamente com o usuário: requisitos e qualidade de software (SWEBOK, 2004).

Em relação aos requisitos de software, Pohl (1994) afirma que o levantamento deles é afetado pelos seguintes aspectos:

- Métodos: análise estruturada ou orientada a objetos;
- Ferramentas: ferramentas para a representação dos requisitos;
- Aspectos sociais: o ambiente de trabalho da equipe de requisitos;
- Competência: diferença de capacitação das pessoas envolvidas no processo de requisitos;
- Restrições econômicas: limitador de recursos humanos, tempo, entre outros.

Observa-se que Pohl não inclui o usuário como elemento que afeta o levantamento, portanto a preocupação incide sobre os elementos "internos" do processo de desenvolvimento de software, ou seja, analistas, programadores e ferramentas.

A preocupação da indústria de software com as ferramentas, métodos e normas é conseqüência dos seguintes fatores: a) é uma indústria nova; b) está em pleno desenvolvimento de métodos e normas que automatize os processos e garanta qualidade (FRANCO, 2007).

Entre as boas iniciativas criadas pela indústria, destacam-se os casos de uso. Segundo Cockburn (2005), os casos de uso (técnica texto), assim como os casos de uso (diagrama da UML) são elementos importantes para auxiliar o analista a capturar as necessidades de negócio de uma aplicação a ser desenvolvida. Wazlawich (2004) lembra que, para um processo de análise efetivo, é necessário utilizar um bom método de trabalho, já que nenhum analista de sistemas capta com perfeição um problema de sistemas no primeiro contato com o usuário.

Segundo o autor, o resultado de um processo de análise malfeito é um software que funciona, mas que não corresponde às necessidades estabelecidas pelo usuário. A qualidade no processo de análise é fundamental, já que um erro encontrado na análise possui um custo ao ser corrigido; no projeto tem outro maior; na implementação, outro maior ainda (WAZLAWICH, 2004).

A ANÁLISE DE SISTEMAS

Segundo Pressman (2006), análise de sistemas define as metas de investigação e a especificação da solução do problema a partir dos requisitos levantados para a criação e implementação de um software. Larman (2004) enfatiza que a análise possui como característica a investigação do problema e dos requisitos, em vez de uma solução. Wazlawich (2004) afirma que na fase de análise serão buscadas as primeiras informações sobre o sistema a ser desenvolvido. Nesta fase, assume-se pouco conhecimento do analista sobre o sistema e, portanto, existe uma grande interação com o usuário.

Lima (2006) realizou um estudo para demonstrar que um processo de análise bem estruturado conduz a uma visão mais ampla do sistema.

Em seu trabalho de conclusão de curso em sistemas de informação, comparou o processo de construção de um sistema denominado "tarefura" com e sem o emprego de técnicas de análise.

Segundo Lima, os resultados foram bastante significativos. Entre as funcionalidades básicas (requisitos funcionais), com ou sem a técnica, todas foram implementadas. Mas, quando se analisaram os requisitos diretamente relacionados ao funcionamento da aplicação, observou-se que o idealizador do software não atentou para uma série de detalhes. No geral, a diferença em termos quantitativos destes itens (requisitos não-funcionais) ficou, respectivamente, em dezoito (sem técnica) contra vinte e cinco (com o uso da técnica de caso de uso).

METODOLOGIAS ÁGEIS E EXTREME PROGRAMMING

Atualmente, as metodologias de software encontram-se divididas, dicotomicamente, em metodologias tradicionais, tais como: o modelo cascata, RAD, prototipação, entre outros; e as metodologias conhecidas como metodologias ágeis, tais como: *extreme programming*, *SCRUM*, entre outras.

O que causou esta divisão é o histórico que os demandantes de desenvolvimento possuem acerca do processo de desenvolvimento de software. Segundo Ambler (2004, p. 21):

> A situação atual do desenvolvimento de software encontra-se aquém da ideal. Os sistemas são invariavelmente entregues com atraso ou com o orçamento estourado, isso quando são efetivamente entregues. Freqüentemente, eles não atendem aos requisitos de nossos clientes e precisam ser desenvolvidos novamente. Nossos clientes ficam descontentes com esses problemas e não apresentam disposição nem para confiar em nós nem para trabalhar conosco, pois se decepcionaram demais no passado.

Como observado por Ambler, além dos usuários se sentirem vítimas do processo de desenvolvimento, os programadores também sentem a repercussão de um processo de desenvolvimento que tem colocado os processos e ferramentas diante dos atores. Ambler (2004) afirma que os

programadores têm sido forçados pelas empresas a trabalharem 50, 60 e até 70 horas semanais no intuito de manter os cronogramas de projetos.

Dessa forma, programadores e usuários observam que algo está errado no processo de desenvolvimento, colocando um contra o outro. De um lado, uma equipe ou programador lotado de documentação que, segundo Ambler (2004), serve como defesa caso "algo aconteça de errado" durante o processo de desenvolvimento, restando aos usuários olhar o desenvolvedor como um profissional não confiável, já que ele não consegue programar o que é pedido e não cumpre prazos.

Para enfrentar estes tipos de problemas, um grupo de 17 desenvolvedores de *software* formou, em 2001, a aliança de desenvolvimento de *software*, que publicou os seguintes princípios: indivíduos e interações valem mais do que ferramentas; um software funcional vale mais do que uma documentação extensiva; a colaboração do cliente vale mais do que um contrato assinado e a mudança de planos vale mais do que seguir um plano (AGILEMANIFESTO, 2001).

Os princípios apresentados no parágrafo anterior são antagônicos com o que é pregado pelas metodologias tradicionais. Elas priorizam a prescrição de mudanças de requisitos durante o projeto, evitando-as (COCKBURN; HIGHSMITH, 2001a, 2001b). Ao contrário, as metodologias ágeis focam o humanismo existente no processo e as constantes mudanças.

Basicamente, uma metodologia pode ser considerada ágil a partir do momento que coloca usuário, programador e software em primeiro lugar. Um exemplar deste tipo de metodologia é o *extreme programming* (XP), que, segundo Teles (2004), é baseado em um conjunto de "boas práticas" de desenvolvimento. São elas:

- Cliente presente: relaciona-se à presença do cliente na fase de preparação e em cada iteração do sistema, onde ele planeja o sistema com a equipe XP. Acrescenta-se que alguns artefatos são produzidos no intuito de facilitar a comunicação entre o usuário e os programadores, tais como: cartões de tarefas, cartões de

classe responsabilidade, conhecidos com CRC (TELES, 2004; AMBLER, 2004).

- Jogo do planejamento: relaciona-se ao planejamento no XP, feito com o uso de iterações (intervalo de semanas fixas) e *releases*, pequenas entregas do software. O cliente participa escrevendo cartões aos programadores no início e durante o projeto (TELES, 2005; JEFFRIES, 2000).
- *Stand up meeting* (reunião em pé): realizada diariamente entre os participantes do projeto, inclusive, se possível, com a participação do cliente. Ela é importante para que todos sejam informados do andamento do projeto e também para que toda a equipe possa trocar idéias referentes a problemas existentes.
- Programação em par: atividade de codificação do software com os dois programadores trabalhando juntos. O trabalho em par faz com que o código seja conhecido por mais de um programador, fazendo com que um auxilie o outro quando ocorrerem dúvidas (TELES, 2004).
- Código coletivo: refere-se ao conhecimento coletivo do código, utilizando-se as práticas de reuniões em pé e a programação em par. Desta forma, código do programa se torna de conhecimento de todos.
- Código padronizado: prática na qual a equipe de programação adota um padrão de codificação, independente do tipo de padrão (TELES, 2004). Essa prática facilita o andamento do projeto, já que a equipe não fica dependente de um programador para alterar um determinado trecho do código.
- *Design simples*: significa que o projeto tem que ter coerência com o modelo mental especificado pelo usuário. O desenvolvimento do sistema durante as interações tem que ser bastante objetivo, buscando a solução mais simples para a implementação de cada requisito ou estória do usuário (TELES, 2005).
- Desenvolvimento orientado a testes (testar antes de codificar): prática que sugere que um par de programadores deve criar testes antes, para certificar o que realmente deve ser implementado. Essa prática também facilita a verificação de erros, guiando o programador na construção correta do código (TELES, 2004).
- Refatoração: relaciona-se à modificação constante de um código, mesmo que este não contenha erros de programação. Busca-se o máximo de simplificação e padronização. Essa prática facilita o entendimento e as futuras manutenções de um código (TELES, 2004; BECK, 2001).
- Integração contínua: à medida que funcionalidades do sistema são codificadas, é necessário que sejam colocadas em produção,

com a aplicação de testes de unidade e testes funcionais (TELES, 2004; JEFFRIES, 2000).

- *Releases* curtos: entrega de partes funcionais do sistema. Ao contrário das metodologias tradicionais, que só disponibilizam o sistema totalmente construído ao final do projeto, no XP, o usuário recebe partes para que possa usar até que o software seja totalmente finalizado (TELES, 2004).
- Metáforas: conjunto de simbologias e termos de linguagens que são comuns a usuários e desenvolvedores, ou seja, ao invés de serem utilizados artefatos e linguagem técnica para se comunicar com o demandante, podem-se utilizar analogias, planilhas e ilustrações que permitam que ele entenda o que está sendo transmitido pelos programadores e vice-versa (AMBLER, 2004).
- Ritmo sustentável: os programadores devem trabalhar 40 horas semanais. A adoção de horas-extras não favorece a velocidade de desenvolvimento da equipe (TELES, 2004), porque o trabalho ganho com as horas-extras será perdido nas semanas posteriores, devido ao cansaço e à falta de disposição dos programadores.

DESENVOLVIMENTO

Nos tópicos anteriores foram apresentados aspectos referentes ao histórico da indústria de software, análise de sistemas e do *extreme programming* (XP). Esses tópicos são importantes para o entendimento do ensaio que se segue.

O ponto fundamental do ensaio refere-se ao usuário, especificamente, aquele que é o demandante por software. Neste caso, será realizada uma análise baseada na literatura e em observações empíricas.

Sabe-se que existem vários problemas relacionados com o levantamento de requisitos. Segundo Oberg (2000), são eles:

- Requisitos não são óbvios e podem ser retirados de diversas fontes;
- Não são expressos de forma clara em palavras;
- Existem muitos níveis de requisitos;
- O número de requisitos pode se tornar não gerenciável, se não existe controle;
- Os requisitos estão relacionados a vários artefatos produzidos, dificultando sua rastreabilidade;
- Os requisitos têm valores e contexto próprios;

- Existem muitos interessados em um mesmo requisito;
- Requisitos podem ser sensíveis ao tempo;
- São mutáveis;
- São influenciáveis pelo conhecimento tácito do usuário;

Sommerville e Kontonya (1998) afirmam que, além das dificuldades relacionadas aos requisitos, existem fatores humanos que influenciam neste processo, tais como: personalidade e status envolvidos, objetivos dos indivíduos dentro da organização e influência política.

Apesar dos autores acima apontarem os problemas e fatores humanos relacionados ao levantamento de requisitos, o autor deste trabalho sintetiza a seguir esses mesmos fatores, contextualizados mediante o papel do analista de sistemas e demandantes de software (usuário).

Em relação aos usuários, são apresentadas as seguintes características:

- Conhece totalmente o negócio a ser modelado;
- Não sabe se expressar corretamente;
- Expressa termos inerentes ao negócio a ser modelado;
- Não conhece o processo de desenvolvimento de software;
- Não emprega a devida importância a uma entrevista;
- Possui "medo" por passar informações importantes a um estranho (analista);
- Fica "entusiasmado" com a solução apresentada;

A primeira característica dos usuários: "conhece totalmente o negócio a ser modelado", é importante devido ao conhecimento empírico produzido pela experiência do usuário em relação ao seu ambiente de trabalho, tal como apontado por Oberg (2000). Portanto, um demandante de software, mesmo que não se expresse corretamente, sempre terá uma visão mais ampla sobre o seu negócio do que um analista de sistemas.

Analisando este aspecto em relação ao *extreme programming*, verifica-se que a equipe de programação comporta-se como um grupo de analista de sistema, ouvindo atenciosamente o usuário, usando metáforas, discutindo as idéias e simplificando o que o cliente deseja para o primeiro *release*, sem se preocupar em prever futuras funcionalidades,

como acontece em metodologias tradicionais (AMBLER, 2004; COCKBURN; HIGHSMITH, 2001a). Logo que o usuário estiver com a primeira versão operacional do sistema, irá verificar se os conceitos empregados no desenvolvimento do software estão alinhados com o que foi especificado, produzindo *feedback* imediato com a equipe de desenvolvimento.

O segundo aspecto do demandante de software: "não sabe se expressar corretamente" refere-se à dificuldade que o usuário tem em expressar como funcionam os processos dentro da empresa ou negócio relacionado a ele.

Existem alguns motivos para que isto aconteça, entre eles podem ser destacados: a) o demandante tem como prioridade o negócio e não o do sistema; b) não possui conhecimento sobre a granularidade dos processos e seus níveis (OBERG, 2000). Para comprovar este fato, observa-se que durante uma entrevista o demandante alterna entre funcionalidades e frases do tipo: "agora lembrei, não se esqueça que o relatório tem que ser impresso somente às quarta-feiras"; "... estamos falando de vendas, mas me lembrei agora que em compras deve-se registrar o funcionário que a registrou".

Os dois pontos apresentados anteriormente apontam um outro problema: o usuário está preocupado com o sistema computacional, enquanto que o analista precisa entender a lógica do negócio. Este tipo de comportamento do demandante pode ser verificado quando expressa as funcionalidades em sentenças, taisl como: "na tela de venda colocar um botão para iniciar uma nova venda, uma grade para os itens...". Esse tipo de enunciado não ajuda o analista a entender o que é "vender produto" para uma determinada organização empresarial.

Em outros casos, os próprios analistas produzem este comportamento, analisam o sistema por meio dos termos "botões" e "telas". Esta abordagem pode até funcionar, mas não consegue capturar os objetivos da organização empresarial e as necessidades dos processos existentes internamente.

No contexto do *extreme programming* estas dificuldades podem ser contornadas usando-se algumas práticas, tais como: metáfora, cliente presente e código coletivo. Para isto, devem-se considerar alguns artefatos rápidos, tais como: os cartões de tarefas, CRCs e narrativas de caso de uso (AMBLER, 2004), que facilitam tanto o entendimento do negócio, das interfaces e diminuem a latência entre idealização de um conceito (idéia proposta por um demandante) e sua implementação.

Nas metodologias tradicionais, essa latência é muito alta, ou seja, somente após a concepção das idéias gerais do sistema e proposta de arquitetura é que serão implementadas as primeiras funcionalidades (LARMAN, 2004; SCOTT, 2003). Essa abordagem faz com que o demandante tenha problemas ao mudar requisitos, mesmo porque ele já assinou algum contrato, baseado no que foi especificado nos casos de usos.

O terceiro aspecto relacionado aos usuários, refere-se aos termos utilizados pelos mesmos nas sessões de análise de sistemas. Neste caso é necessário preparar um glossário. Às vezes, dentro de uma organização, criam-se conceitos ímpares, que, fora dela, não possuem significado, introduzindo ruídos nos conceitos utilizados para implementar um sistema. Seja o exemplo: "... na nossa empresa preparamos o produto a ser entregue pelo vendedor". Neste caso, o que significa "preparamos"? Para uma organização, preparar é "construir"; para outra, é "fazer o acabamento do produto final", gerando dois sistemas diferentes a partir do mesmo conceito.

Para este caso, o emprego de metáforas auxilia na busca de uma solução, já que produz muitas analogias, facilitando o entendimento do conceito pela equipe de programação (JEFFRIES, 2000; TELES, 2004; BECK, 2001). Um bom exemplo do emprego de metáfora é o desenvolvimento de banco de dados. Ao invés de fazer modelos de entidade e relacionamento, que normalmente não são compreensíveis ao usuário, cria-se uma planilha eletrônica para construir um modelo para o demandante. Dessa forma, desenvolvedor e demandante verificam em

tempo de análise quais dados serão efetivamente usados, já que estão trabalhando em um modelo familiar aos dois.

Quarto aspecto: o usuário não conhece o processo de desenvolvimento de software. Este aspecto é importante por estar relacionado ao conceito de que software é intangível (PRESSMAN, 2006).

O demandante (usuário) não abstrai software como um analista de sistemas e conclui que fazer alterações e pedir novas funcionalidades não produzem impactos na estrutura do programa. Mas, algumas mudanças podem deixar um software inoperante durante meses, ou produzir uma série de erros inexistentes até então.

O emprego de metáfora (prática XP) pode mostrar ao demandante o tipo de estrutura contida em um software. Isso é importante por dois aspectos: a) mostra ao demandante a dimensão de um software; b) se certas alterações são possíveis.

Um software pode ser comparado, por meio de uma metáfora, com casas e prédios. Com esta metáfora é possível mostrar aos usuários a complexidade de algumas alterações realizadas em um software. Por exemplo, mover os móveis de lugar em uma casa é relativamente fácil. Ao contrário, mudar a posição física de uma parede é muito complexo. Neste ultimo caso, é necessário tirar os móveis do cômodo, desativar as chaves de energia, quebrar paredes, entre outros. Portanto, esta linguagem melhora a avaliação de risco realizada por um demandante.

Outro ponto forte do emprego de metáforas no processo de construção de um software está relacionado à metodologia empregada por uma equipe de desenvolvimento.

Neste contexto, pode-se comparar a forma de trabalho das metodologias tradicionais e ágeis. Nas tradicionais, o usuário, eventualmente, verá o progresso da construção e irá dar indicações do que deve ser corrigido, somente habitando a casa ao final da construção. Por outro lado, nas metodologias ágeis, especificamente no XP, o usuário irá pedir para construir algo no qual ele poderá morar rapidamente, tal como um quarto. Após a construção do cômodo, o dono estará

participando ativamente da construção dos outros ambientes, usando-os à medida que a construção da casa avança. Esta estratégia pode aumentar o prazo da construção, devido à possibilidade do dono sempre estar pedindo mudanças, mas estará ciente disso e não verá esse fato como um atraso de cronograma.

Quinto aspecto: os demandantes de software não empregam a devida importância às entrevistas. Por mais formal que o analista de sistemas seja, os demandantes não prestam a devida importância à análise por alguns motivos: a) a entrevista é efetuada durante o trabalho, ou seja, ele sempre dará prioridade para atender aos seus clientes e negócios; b) não compreende a importância da entrevista, logo não visualiza o porquê ser entrevistado, já que o analista sabe o que tem que ser feito.

Sexto aspecto: os usuários têm "medo" de passar informações importantes a um estranho. Este aspecto não se refere somente à insegurança gerada pela idoneidade de quem o entrevista. Também, existe "medo" em relação ao cumprimento do prazo do projeto ante a necessidade de se usar o software, da mesma forma que ouvir que o sistema está sendo desenvolvido e nunca ver nada de concreto. Como conseqüência, as implicações de produzir "medo" nos demandantes variam de pequenas discussões, agressões verbais e até o cancelamento do projeto (JONES, 2001).

Por outro lado, o sétimo aspecto: "entusiasmo" do demandante, mediante a possibilidade do software "resolver todos os seus problemas", mesmo que seja contrário ao aspecto anterior, tem que ser evitado, pois gera perda de escopo durante o desenvolvimento do sistema.

Após terem sido apresentadas as características cinco, seis e sete, devem ser analisadas em relação ao XP. O "medo" e o "entusiasmo" são reações que são apresentadas quando não existe certeza do resultado do desenvolvimento. O XP minimiza o "medo" e "entusiasmo", à medida que o demandante recebe aos poucos o sistema: prática de *releases* curtos. Além disso, o emprego de cartões de tarefas na análise faz com que o

demandante escreva somente o que tem que ser realizado, sem fazer com que passe por uma "prova de conhecimento sobre seu negócio". Como afirma Ambler (2004), análises prescritivas levam a um conjunto de requisitos que possivelmente não serão utilizados.

A seguir, são apresentadas as características de um analista de sistema ou equipe de desenvolvimento:

- Insiste em propor "melhores soluções";
- Comunica-se com uma linguagem técnica;
- Esquece que quem conhece o negócio é o usuário;
- Apresenta "artefatos" que confundem o usuário;
- Falta de preparo para se relacionar aos vários tipos de comportamento do usuário;
- Não tem um conhecimento mínimo do domínio a ser modelado.

A primeira delas, insistir em apresentar melhores soluções, tem como resultado dois aspectos distintos: a) ao propor soluções ao demandante, ser bem entendido, ou; b) ao propor soluções, não ser bem interpretado pelo demandante. Isto parece paradoxal, mas é exatamente o segundo caso que mais acontece na prática, devido ao analista de sistema mostrar ao demandante que ele não conhece o seu negócio tão bem quanto ele imaginava. Portanto, para resolver esta situação, nos momentos iniciais de uma análise, o desenvolvedor de sistema deve se comportar como um ouvinte e produzir fielmente o que está sendo pedido. Somente após ter sido estabelecido um diálogo comum e de confiança entre as partes, são plausíveis as propostas de soluções que agreguem valor ao negócio.

Ao analisar o XP neste sentido, verifica-se que a equipe trabalha como "ouvinte" do demandante, já que, segundo Teles (2004), o demandante escreve os cartões de tarefas (espécie de requisitos) aos programadores e explica à equipe o que está sendo pedido. Outra participação importante do demandante acontece na escrita dos testes funcionais (TELES, 2004; BECK, 2000).

O segundo aspecto, a comunicação do analista de sistemas com o usuário em linguagem técnica da informática (PRESSMAN, 2006), não é

considerado uma boa prática. Nos primeiros momentos de uma análise, não tem sentido falar de modelagem conceitual, diagrama de seqüência e mapas de dependência de requisitos ao usuário.

Terceiro aspecto: um analista de sistemas tem que ter em mente que a investigação na fase de análise deve produzir o entendimento do negócio do demandante (WAZLAWICH, 2004) e não mostrar ao demandante que ele é ignorante no seu negócio e em análise e projeto de sistemas. Neste caso, o emprego de metáforas (prática XP) colabora com a solução deste tipo de problema ao gerar modelos gráficos, planilhas eletrônicas e analogias, que são de fácil entendimento para ambos.

Quarto aspecto: "apresenta artefatos que confundem o usuário". Segundo Larman (2004), artefato é qualquer produto produzido durante um projeto de desenvolvimento. Tanto a técnica de casos de uso de Cockburn (2005) e os de diagramas de casos de uso da UML são exceções deste caso, já que são ferramentas empregadas em metodologias tradicionais que auxiliam o demandante a entender o que será desenvolvido.

No caso do *extreme programming*, as narrativas de caso de uso, cartões de tarefas, cartões CRC (AMBLER, 2005), são ferramentas simples que auxiliam o demandante a passar corretamente as especificações para a equipe de desenvolvimento, já que ele confecciona diretamente alguns destes artefatos.

Por outro lado, a apresentação de artefatos mais técnicos, tais como, documento de arquitetura, diagramas de classes, não irá impressionar nenhum demandante. Mostrar estes artefatos pode servir como argumento para dizer que a equipe de desenvolvedores ficou perdendo tempo com o desenho de diagramas (AMBLER, 2005).

Quinto aspecto: "falta de preparo para se relacionar aos vários tipos de comportamento do usuário". Este aspecto é raramente trabalhado nas metodologias de desenvolvimento de software. Em algumas metodologias mais clássicas, como o método cascata, o usuário participava no início do desenvolvimento e na implantação (FRANCO, 2007). A interação entre

este e o método era mínima, produzindo softwares que não representavam o que o demandante havia especificado. Mesmo hoje, no processo unificado e no *extreme programming*, melhorias no relacionamento entre demandante e desenvolvedor devem ser introduzidas como premissa e não como um favor. E, neste quesito, o XP tem conseguido grandes êxitos ao desmistificar o processo de projeto de software aos usuários, trazendo-o junto ao desenvolvimento do projeto (JEFFRIES, 2000; BECK, 2000; TELES, 2004).

Na realidade, observa-se que, não obstante as metodologias, os analistas de sistemas têm um engessamento aos procedimentos formais de análise (engenharia de requisitos) (SOMMERVILLE; KONTONYA, 1998), concentrando-se na aplicação dos métodos e esquecendo-se da figura humana do demandante. Para estes casos, outros aspectos têm que ser levados em consideração: a) não se devem tratar demandantes distintos ou grupos de demandantes da mesma forma; b) ouvir o demandante em um primeiro momento, para entender sua cognição, seu *modus operandi*. Isto é importante para se promover empatia entre as partes.

Outro ponto que tem que ser destacado refere-se à contribuição positiva da confiança mútua em momentos difíceis do projeto. Ou seja, se em um determinado momento a confiança e empatia forem grandes, é capaz do demandante pedir a opinião do analista ou da equipe. Em outros momentos, quando o analista de sistema perceber ansiedade causada por problemas pessoais, é prudente este adiar a entrevista ou escrita de cartões de tarefa, conversar com o usuário, estabelecer confiança e continuar novamente a entrevista. Possivelmente, os usos de técnicas psicológicas poderiam contribuir muito para a qualidade da relação demandante-analista.

O último aspecto relacionado aos analistas de sistema tem sido a falta de domínio em relação aos negócios analisados. É importante um conhecimento prévio do que deverá ser analisado, ou seja, estudar, fazer uma revisão bibliográfica do assunto e conhecer tecnologias relacionadas ao negócio. É uma prática que apresenta bons resultados. O demandante

percebe que o analista pesquisa, apresenta, sabe discutir o assunto e, claro, estabelece com isso uma relação de confiança.

Ainda em relação ao aspecto anterior, mesmo que os programadores utilizem XP, um conhecimento prévio do assunto acelera o entendimento do modelo. Para alguns modelos, as metáforas são suficientes, mas em outros casos, tais como, sistemas que utilizam cálculos matemáticos, algoritmos complexos, interfaces gráficas diversas, é interessante o conhecimento prévio do assunto.

CONSIDERAÇÕES FINAIS

O demandante (usuário) é o principal interessado em um processo de desenvolvimento de software. Durante muito tempo, a indústria de software, composta por médias e grandes empresas de informática, se preocupou com métodos e normas do processo de desenvolvimento. Isso trouxe grandes melhorias aos processos, já que, há poucos anos, não existia nem mesmo uma linguagem padrão de projetos. Mas, mesmo assim, um elemento ainda continuava sendo marginalizado: o usuário.

Este artigo apresentou algumas características relacionadas aos usuários e aos analistas de sistemas, cuja relação é muito importante durante um processo de desenvolvimento, tal como é verificado na metodologia *extreme programming*. O analista é o responsável por ser a interface entre toda a equipe de desenvolvimento e o demandante. No caso do XP, a equipe trabalha em conjunto com o demandante e o mesmo' percebe que ele direciona a equipe.

Apresentaram-se várias características de comportamentos e atitudes que são comumente reproduzidas pelos usuários. Talvez esses comportamentos sejam respostas à forma como é apresentada a eles o processo de análise e projeto de sistemas, produzindo diversos tipos de reações.

Neste caso, apontaram-se também possíveis atitudes a serem observadas pelos analistas de sistemas, que, em alguns casos, devem

proceder como verdadeiros "psicoterapeutas", tamanha a diversidade de tipos de usuários encontrados em processos de analise de sistemas. Isto tem suma importância, porque uma abordagem inapropriada pode causar, em um primeiro momento, um desconforto entre ambas as partes.

Finalmente, tentou-se abordar a maioria dos aspectos relacionados a usuários e programadores, mas podem existir outros. No entanto, pode-se afirmar que as situações que foram abordadas neste ensaio são encontradas no cotidiano e vivenciadas pela maioria dos analistas, usuários e programadores. No que se refere à metodologia XP, ela resgatou o prestigio do usuário, já que é ele que investe em um projeto à espera resultados. Portanto, a relação analista-demandante deve ser permeada de transparência, clareza, linguagem comum e uma boa ligação afetiva.

REFERÊNCIAS

AGILEMANIFESTO. **Manifesto for Agile Software development.** (2001) <disponível em: http://www.agilemanifesto.org> Acessado em: 15/03/2008.

AMBLER, Scott W. **Modelagem ágil: práticas eficazes para a programação extrema e o processo unificado.** São Paulo, Bookman, 2004.

BECK, C. **Extreme programming explained: embrace changes.** Addison-Wesley. 2000.

COCKBURN, A; **Escrevendo casos de uso eficazes:** um guia prático para desenvolvedores de software. Editora Bookman, 2005.

COCKBURN, A; HIGHSMITH, J; **Agile software development: the business of innovantion.** IEEE publishing. (2001b) <disponível em: www.adaptivesd.com/articles/IEEEArticle1Final.pdf> Acessado em 23/03/2008.

COCKBURN, A; HIGHSMITH, J; **Agile software development: the people factor.** IEEE publishing. (2001a) <disponível em: www.adaptivesd.com/articles/IEEEArticle2Final.pdf> Acessado em 23/03/2008.

FRANCO, E. F. **Um modelo de gerenciamento de projetos baseado nas metodologias ágeis de desenvolvimento de software e nos princípios da produção enxuta.** Dissertação de mestrado. Escola politécnica da USP, São Paulo, 2007.

JACOBSON, I; Grady, B; Rumbaugh, J; **UML guia do usuário.** Editora Campos. 2000.

JEFFRIES, R; ANDERSON, A; HENDRICKSON, C. **Extreme programming installed.** Addison-Wesley. 2000.

JONES, C; **Conflict and Litigation between costumers and developers.** Software productivity research – Artamis corporation. 2001. Disponível em: <http:// www.spr.com/news/ConflictandLitigationArticle.pdf>. Acessado em 20/02/2008.

LARMAN, C. **Utilizando UML e padrões:** uma introdução à análise e ao projeto orientado a objetos e ao processo unificado. Editora Bookman. 2004.

LIMA, G. A. **Análise de eficiência da aplicação da técnica de caso de uso em processos de análise de sistemas.** Trabalho de conclusão de curso (Bacharelado em sistemas de informação). Instituto de Ensino Superior Social e Tecnológico de Taguatinga. Taguatinga-DF. 2006.

MEIRELLES, F. S. **Informática: novas aplicações com microcomputadores.** 2ª Ed. Makron Books, São Paulo, 1994.

OBERG, R; PROBASCO, L; ERICSON, M; **Applying requirements management with use cases.** Rational Sofware white paper. (2000) <disponível em: www.compgraf.ufu.br/alexandre/esof/use_cases.pdf > Acessado em 25/02/2008.

POHL, K; **The three dimensions of software requirements:** a framework and its application. Information systems; Vol. 19; n.3; p. 243-258. 1994.

PRESSMAN, R. S. **Engenharia de Software.** Editora McGraw-Hill. 6ª ed. 2006.

SCOTT, K; **O processo unificado explicado.** Ed. Bookman, 2003.

SEBESTA, R. W. **Conceitos de linguagens de programação.** 5ª Ed. Bookman, São Paulo, 2003.

SOMMERVILLE, I; KONTONYA, G. **Requirements engineering processes and techniques.** Ed. John Wiley and Sons, 1998.

STANDISH; **The caos report.** The Standish Group International. Inc. USA. 1994.

SWEBOK. **Guide to the software engineering body of knowledge.** Instituto de engenheiros elétricos e eletrônicos. 2004.

TELES, Vinícius Magalhães. **Extreme programming.** São Paulo: Novatec, 2004.

TELES, Vinícius Magalhães. **Um estudo de caso da adoção das práticas e valores do extreme programming.** Tese de mestrado, Rio de Janeiro, 2005.

WAZLAWICH, R. S. **Análise e projeto de sistemas de informação orientados a objetos**. Editora Campus. Publicação da Sociedade Brasileira de Computação. 2004.