View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Servicio de Difusión de la Creación Intelectual

PROCESSO DE SOFTWARE: UM ESTUDO DE CASO EM XP

Cristina Bona
crisbona@globo.com
Universidade Federal de Santa Catarina - UFSC
Rua Aristides Lobo, 520/311A - Agronômica
88025-510 - Florianópolis-SC, Brasil

Marcello Thiry
thiry@sj.univali.br
Universidade do Vale do Itajaí - UNIVALI - Campus São José
Núcleo de Computação
Rodovia SC 407, Km 4
88122-000 - São José-SC, Brasil

RESUMO

O presente trabalho busca fornecer, através de exemplo práticos e representações gráficas, uma estrutura capaz de orientar a aplicação de um processo de software. Um dos principais esforços dos pesquisadores envolvidos com a Engenharia de Software tem sido apresentar e abstrair modelos que descrevem processos de software. Estes modelos permitem que se compreenda o processo de desenvolvimento dentro de um paradigma conhecido. A existência de um modelo é apontada como um dos primeiros passos em direção ao gerenciamento e à melhoria do processo de software. Na última década, um novo segmento da comunidade de Engenharia de Software vem defendendo processos simplificados, também conhecidos como "processos ágeis", focados nas pessoas que compõem o processo e, principalmente, no programador. O trabalho apresenta a aplicação e avaliação do processo Extreme Programming (XP), o qual está entre os "processos ágeis" e que vai contra uma série de premissas adotadas por métodos mais tradicionais de desenvolvimento. Como produto final, além da aplicação do processo, são discutidos os seus pontos positivos e negativos.

Palavras-chave: Processo de Software, *Extreme Programming*, Processos Ágeis.

ABSTRACT

This work pretends to provide a structure capable of to guide the application of a software process using a practical example and graphical notations. One of the main researchers' efforts involved with the Software Engineering has been to introduce and to abstract models that describe software processes. These models allow the understanding of the development process inside a well-known paradigm. The existence of a model is pointed out as one of the first steps in direction to the management and to the improvement of the software process. In the last decade, a new group in the Software Engineering community has been defended the use of simplified processes also known as "agile processes", which are focused on people. This paper introduces the application of the Extreme Programming process, which is classified as one of the agile processes. This process is based on a different set of premises if compared with traditional development methods. As a conclusion of this work, after the application itself, it will be pointed out the positive and negative aspects of the Extreme Programming.

Keywords: Software Process, Extreme Programming, Agile Processes.

1. INTRODUÇÃO

O impacto e a rápida evolução ao longo dos últimos 40 anos das tecnologias relacionadas com os sistemas de informação têm colocado sucessivos desafios às empresas. A dependência e demanda crescentes da sociedade em relação à Informática e, em particular, a software, tem ressaltado uma série de problemas relacionados ao **processo** de desenvolvimento de software: alto custo, alta complexidade, dificuldade de manutenção, e uma disparidade entre as necessidades dos usuários e o produto desenvolvido.

Empresas de software em todo o mundo empregam perto de 7 milhões de técnicos e geram anualmente uma receita de mais de 600 bilhões de dólares, com taxa de crescimento anual de mais de 25% nos últimos três anos. A indústria de software é vista atualmente como um dos segmentos mais promissores, com um enorme potencial futuro (CORDEIRO, 2000). Desta forma, desenvolver projetos de software eficientes é de fundamental importância para a indústria de software como um todo.

Os processos usados para desenvolver um projeto de software têm a maior importância na qualidade do software produzido e na produtividade alcançada pelo projeto. No entanto, não existe um modelo uniforme que possa descrever com precisão o que de fato acontece durante todas as fases da produção de um software; os processos implementados são muito variados, e as necessidades de cada organização diferem substancialmente (SILVA & VIDEIRA, 2001).

Além disso, na última década, um segmento crescente da comunidade de Engenharia de Software vem defendendo a existência de problemas fundamentais da aplicação sistemática e institucionalizada de processos de software convencionais (HIGHSMITH, 2002) e (BECK, 2000). Estes proponentes advogam processos simplificados, focados nas pessoas que compõem o processo, e principalmente no programador.

Neste contexto, o processo ***Extreme Programming*** (XP) está entre os denominados "processos ágeis" (HIGHSMITH, 2002), que vai contra uma série de premissas adotadas por métodos mais tradicionais de desenvolvimento. O XP consiste numa série de práticas e regras que permitem aos programadores desenvolver software de uma forma dinâmica e ágil, com mínimo de documentação.

Extreme Programming (XP) é uma metodologia leve, eficiente, flexível e de baixo risco para times pequenos e médios, que desenvolvem software com requisitos dinâmicos ou em constante mudança (BECK, 2000). A metodologia XP foi criada por Kent Beck, que no início dos anos 1990 pensava sobre caminhos melhores para o desenvolvimento de software.

XP é uma disciplina de desenvolvimento de software baseado em valores de simplicidade, comunicação, feedback e coragem. XP envolve o time inteiro para um trabalho de equipe com práticas simples, com feedback suficiente para capacitar o time a ver onde eles estão e a convergir para práticas em uma solução única (JEFFRIES, 2001).

Ainda de acordo com Jeffries (2001), os programadores trabalham em pares e em grupo, com projeto simples e código obsessivamente testado, melhorando o projeto continuamente para manter sempre as necessidades correntes e o sistema integrado. Os programadores escrevem todo código de produção em pares, e todos trabalham junto o tempo todo. Eles codificam de forma consistente para que todos possam entender e melhorar o código quando necessário.

O presente artigo mostra a aplicação e análise do processo XP no projeto "Gratificação de Incentivo a Docência (GID)" do Centro Federal de Educação Tecnológica de Santa Catarina (CEFET/SC). A seção 2 apresenta os valores e práticas do processo XP. A seção 3 detalha o ciclo de vida e as fases do XP. A seção 4 mostra a aplicação do processo XP. A seção 5 apresenta a análise dos resultados. Finalmente na seção 6 temos as considerações finais sobre este trabalho.

2. VALORES E PRÁTICAS DO XP

É importante ressaltar que, o XP segue um conjunto de valores, princípios e regras básicas que visam alcançar eficiência e efetividade no processo de desenvolvimento de software (BECK, 2000). Os valores são quatro: comunicação, simplicidade, *feedback* e coragem. Nestes valores, estão fundamentados alguns princípios básicos: *feedback* rápido, simplicidade assumida, mudanças incrementais, compreensão às mudanças e qualidade do trabalho. A partir destes princípios, foram definidas as doze práticas básicas que são adotadas pelo XP.

Os quatro valores do XP são:

1. **Comunicação**: É o primeiro valor do XP. Deve-se preferir sala de discussão (*chat*) a correio (*e-mail*), telefonemas a *chat*, conversar pessoalmente a telefonemas, trabalhar na mesma sala a ter salas isoladas, trabalhar em conjunto a revisar o resultado final (WUESTEFELD, 2001). Jeffries (2001) enfatiza que deve existir comunicação em todos os sentidos. O time deve escrever o código usando as mesmas convenções de formatação, nomeando convenções, e definindo os padrões de codificação. Desta forma, independente que quem escreveu o método é familiar e fácil de entender. A comunicação deve ser facilitada para todo o time do XP;
2. **Simplicidade**: É o segundo valor do XP. O objetivo é simplificar continuamente o software. É isso que sustenta a premissa de extremo, pois a simplicidade não é fácil. O processo em si também é adaptado, a cada dia, se houver como torná-lo mais simples. Simplicidade e comunicação estão diretamente relacionadas. Pois quando mais comunicação existe no time, mais claro fica de se ver o que realmente precisa ser feito e, mais seguro de ver o que não precisa ser feito. Assim, se faz o software mais simples quando é possível de se trabalhar;
3. ***Feedback***: É o terceiro valor do XP. Todo problema é evidenciado o mais cedo possível para que possa ser corrigido o mais rápido possível. Toda oportunidade é descoberta o mais cedo possível para que possa ser aproveitada o mais rápido possível (WUESTEFELD, 2001);
4. **Coragem**: Este valor somente tem peso se estiver combinado com os três primeiros valores (BECK, 2000). É preciso coragem para apontar um problema no projeto, pedir ajuda quando necessário, simplificar código que está funcionando, expor para o cliente que o prazo estimado para implementar determinado requisito não é suficiente, fazer alterações no processo de desenvolvimento. Ou seja, fazer a coisa certa mesmo que não pareça o mais correto naquele momento.

As doze práticas do XP são:

1. **Jogo de Planejamento** (*Planning Game*): Rapidamente determina o escopo das próximas versões combinando a prioridades do negócio e estimativas técnicas. Como prática, estrutura o plano e atualiza o plano.
2. **Pequenas versões** (*Small releases*): O time XP coloca rapidamente um sistema simples em produção, e o atualiza freqüentemente em ciclos bastante curtos.
3. **Metáforas**: Guiam todo o desenvolvimento e a comunicação com o cliente utilizando um "sistema de nomes" e uma descrição do sistema.
4. **Projeto simples** (*Simple design*): O sistema precisar ser projetado o mais simples possível satisfazendo os requisitos atuais.
5. **Teste**: Os times XP focalizam a validação do software durante todo o processo. Os programadores escrevem primeiro os testes, e só então continuam o desenvolvimento que

deve atender os requisitos destes testes. Os clientes escrevem testes para validar se os requisitos estão sendo atendidos.

6. ***Refactoring***: Os times procuram aperfeiçoar o projeto do sistema durante todo o desenvolvimento, mantendo a clareza do software: sem ambigüidade, com alta comunicação, simples, porém completo.
7. **Programação em pares** (*Pair Programming*): Todo código produzido é feito em duplas, ou seja, dois programadores trabalhando juntos na mesma máquina.
8. **Propriedade coletiva** (*Colletive ownership*): Todo o código pertence a todo o time. Então, qualquer um pode alterar qualquer código em qualquer tempo.
9. **Integração contínua** (*Continnuous integration*): Integra e constrói o sistema de software várias vezes por dia. A todo o momento, uma tarefa é completada.
10. **Semana de 40-horas** (*40-hour week*): Como regra, não se deve trabalhar mais que 40 (quarenta) horas na semana. Programadores exaustos cometem mais erros.
11. **Cliente dedicado** (*On-site customer*): Os times XP tem "o cliente" real, disponível todo o tempo, que determina os requisitos, atribui as prioridades, e reponde as dúvidas.
12. **Código padrão** (*Coding standards*): Todos os programadores escrevem o código da mesma forma, de acordo com regras que asseguram a clareza e a comunicação através do código.

A maioria das regras XP causa polêmica à primeira vista e muitas não fazem sentido se aplicadas isoladamente. É a sinergia de seu conjunto que sustenta o processo XP, encabeçando uma verdadeira revolução de metodologias ágeis. Cada prática é uma parte simples. A riqueza está na interação das partes.

3. O CICLO DE VIDA E AS FASES DO XP

O projeto ideal em XP é aquele que inicia por uma curta fase de desenvolvimento, seguida de anos de produção e refinamentos simultâneos e finalmente encerra quando o projeto não faz mais sentido (BECK, 2000).

O ciclo de vida e as fases do processo XP são abordagens muito discutidas por diversos autores que adotam essa metodologia. O ciclo de vida XP é bastante curto e, à primeira vista, difere dos padrões dos modelos de processo convencionais. Entretanto, esta abordagem faz sentido somente em um ambiente onde as mudanças de requisitos do sistema são fatores dominantes (OSHIRO, 2001). No caso extremo, os requisitos podem mudar no meio da versão, para atender funcionalidades mais importantes do que as definidas no planejamento original. A seguir são apresentadas as fases do XP:

- **Exploração**: O objetivo da fase de Exploração é entender o que o sistema deve fazer, bem o suficiente para que possa ser estimado (WAKE, 2002). Sendo que, o cliente escreve e administra histórias (*story cards*) e o programador estima as histórias. Encerra quando todas as histórias necessárias para a próxima fase – fase de planejamento - tenham sido estimadas. A fase de Exploração começa com as regras de negócio. O cliente escrevendo cartões de história que descrevem o que o sistema precisa fazer. Em paralelo as histórias dos clientes os programadores estão experimentando ativamente as diferentes tecnologias e as possíveis configurações, explorando as possibilidades para a arquitetura do sistema. Leva-se de uma a duas semanas testando a arquitetura, mas deve-se testar de três a quatro maneiras, ou seja, diferentes pares podem testar diferentes tecnologias e comparar ou, dois pares podem testar a mesma tecnologia e comparar as diferenças emergentes;

- **Planejamento**: O objetivo da fase de Planejamento é definir a menor data e o maior conjunto histórias que serão realizadas na primeira versão. Esta definição é feita de acordo com estimativas entre cliente e programadores (BECK, 2000). Assim que as histórias são coletadas, o Jogo de Planejamento é conduzido. O cliente decide quais histórias são vitais e que devem fazer parte da primeira versão. Desta forma, pode-se elaborar uma lista priorizada das histórias. Se houver uma boa preparação durante a fase de exploração é necessário apenas alguns dias para a fase de planejamento;
- **Iteração para primeira versão**: Os compromissos são divididos para serem executados em iterações que duram de uma a quatro semanas. Para cada história executada naquela iteração é produzido um conjunto de testes funcionais (BECK, 2000). A primeira iteração, mostra como a arquitetura do sistema irá se comportar. Então, as histórias devem ser escolhidas de forma que representem força para criar todo o sistema. A pergunta chave para ser trabalhada nesta fase é: Qual a coisa de maior valor para o time para ser trabalhada nesta iteração? (BECK, 2000). Conforme Reis (2000), uma seqüência de ciclos iterativos conduzem o desenvolvimento, que concentra o projeto, a codificação, os testes e as versões do produto. Normalmente, há alguns ciclos iterativos antes da fase de produção. No final de cada iteração o cliente terá completado a execução de todos os testes funcionais e, no final da última iteração, o sistema estará pronto para entrar em produção;
- **Produção:** Existem alguns processos para avaliar se o *software* realmente está pronto para entrar em produção. Pode-se implementar novos testes para provar se o sistema está estável o suficiente para entrar em produção (BECK, 2000). Testes são freqüentemente aplicados nesta fase. Pode ser necessário apenas ajustar o desempenho. E o melhor momento para realizar estes ajustes é no final da versão. Pois, se tem mais conhecimento do projeto, assim como existe a possibilidade de realizar estimativas reais de sobrecarga de produção do sistema. E provavelmente, este teste poderá ser realizado na máquina de produção;
- **Manutenção:** A fase de manutenção é o estado normal de um projeto XP. Deve-se simultaneamente produzir novas funcionalidades, manter o sistema existente rodando, substituir membros do time que partem e incorporar novos membros ao time (BECK, 2000). Nesta fase, pode-se tentar fazer *refactorings* maiores, os quais, nas versões anteriores, causaram certo receio. Pode-se testar novas tecnologias que se pretende incorporar nas próximas versões, ou migrar a tecnologia que está em uso para versões mais atualizadas. O cliente pode escrever novas histórias que tragam para o seu negócio grandes conquistas. A estrutura do time provavelmente será alterada, ou seja, voltada para a produção. Talvez seja necessário um *help-desk*, e já não se tenha à necessidade de uma grande quantidade de programadores. Entretanto, se deve ter cuidado com a rotação da posição de todos os programadores, o time pode ser mudado gradualmente. Isto é importante, tanto para comunicar a estrutura de todo o projeto, quanto os detalhes de planejamento e implementação, e que somente pode ser feito através do contato diário entre o time;
- **Fim do Projeto:** Beck (2000) considera que "Morrer bem é tão importante quanto viver bem. Isto é uma verdade para o XP como para as pessoas". Quando não mais existir novas histórias, é o momento de finalizar o projeto. É o momento de escrever algumas páginas (de 5 a 10 páginas) sobre a funcionalidade do sistema, um documento que auxilie no futuro a saber como realizar alguma alteração no sistema. Uma boa razão para finalizar o projeto é o cliente estar satisfeito com o sistema e não ter mais nada que consiga prever para o futuro. Toda a equipe que trabalhou no sistema deve ser reunida para reavaliação. Aproveite a oportunidade de analisar o que pode ter causado queda no sistema e o que fez o projeto

avançar. Assim, o time saberá melhor o que fazer no futuro, o time executará tarefas de formas diferentes da próxima vez.

4. APLICAÇÃO DO XP

A escolha da utilização do processo XP no sistema GID se deu pelo fato que, o XP se destina a projetos nos quais o cliente não necessita de uma documentação formal, o time envolvido deve ser pequeno e o prazo de execução dura poucos meses (BONA, 2002). Além disso, o XP trabalha em ambientes dinâmicos, ou seja, os requisitos podem sofrer freqüentes mudanças.

O trabalho foi iniciado com a fase de exploração, cujo objetivo foi compreender o que o sistema precisava fazer, bem o suficiente para que pudesse ser estimado. Essa estimativa foi preliminar, e partiu das primeiras histórias escritas pelo cliente. Em paralelo as histórias dos clientes, foram avaliadas as diferentes tecnologias e exploradas as possibilidades para a arquitetura do sistema. Nesta etapa, foi utilizado também, o diagrama de atividades da UML (OMG®, 2001).

Durante a fase de planejamento, foi especificado o projeto, as iterações e o dia-a-dia. Foi elaborada uma estimativa com base nos cartões de história, na metáfora e em uma solução simples (ASTELS et al., 2002). O objetivo desta fase foi estimar o menor tempo e o maior número de histórias para a primeira versão (BECK, 2000). O projeto também poderia ser replanejado se uma alteração significativa, repassada pelo cliente ou pelos desenvolvedores, fosse identificada.

4.1 Descrição do sistema

O aplicativo foi construído para calcular a Gratificação de Incentivo a Docência (GID) e teve como base o regulamento elaborado pelo Comitê de Avaliação Docente do CEFET/SC. Este regulamento estabelece os critérios e procedimentos de avaliação e desempenho docente para a implementação da GID.

Para efeito de pontuação, os professores são classificados em grupos. Conforme o grupo, existe um critério para avaliação, ou seja, uma seqüência de procedimentos e cálculos que o sistema executa. O centro do sistema é o gerenciamento dos pontos obtidos pelo professor, que é proporcional ao seu desempenho docente. A principal finalidade do sistema foi gerar informações para os recursos humanos.

Neste momento, identificou-se a necessidade de um diagrama que mostrasse de forma genérica o fluxo do negócio. Neste sentido, foi escolhido o diagrama de atividades que ilustra o fluxo de atividades. Torna-se importante ressaltar que este diagrama é parte da especificação UML (OMG®, 2001). O diagrama completo pode ser visto em (BONA, 2002).

4.2 Composição e tarefas do time

Todos os contribuintes do projeto sentavam-se juntos como membros de um time (JEFFRIES, 2001). O time incluiu um especialista de domínio - "o cliente" - que definia os requisitos, fixava as prioridades e guiava o projeto. O time possuía uma dupla de programadores, os quais também absorveram a tarefa de ajudar o cliente a definir os testes de aceitação e os requisitos. Além destes, existia um gerente que provia recursos, mantinha a comunicação externa e coordenar as atividades. Nenhum destes papéis foi necessariamente de propriedade exclusiva de um só indivíduo. Todos os membros do time contribuíram de todas as formas que puderam, conforme suas capacidades. Jeffries (2001) destaca que os melhores times não têm nenhum especialista, mas contribuintes gerais com habilidades especiais.

4.3 Metáfora

A metáfora foi usada para superar os problemas de conceituação e comunicação inicial com o cliente (ASTELS, 2002). Quando se começou a pensar sobre o sistema, se achou que ele era direto o suficiente para se adotar simplesmente uma metáfora ingênua, um sistema de nomes baseado no domínio. Entretanto, o cliente em determinado momento, da fase de exploração, comentou: "Pensei inicialmente em utilizar uma planilha do Excel para realizar os cálculos. Pois, a entrada dos dados é facilitada e consigo visualizar os critérios de avaliação e, rapidamente tenho o resultado". Então, este comentário passou a ser a metáfora para o sistema da GID. A principal vantagem da metáfora foi que o time conseguiu, rapidamente, pensar em uma interface para o usuário.

4.4 Histórias do usuário

Os cartões de história foram elaborados pelo cliente. Todas as histórias de interesse foram anotadas e discutidas pelo time (ver figura 1). Inicialmente, houve uma pequena dificuldade do cliente em elaborar uma história. Entretanto, não existiu resistência. Partiu-se com as histórias centrais e, posteriormente, adicionou-se algum detalhe.

É importante ressaltar que um cartão de história é apenas um lembrete de uma conversação com o cliente (ASTELS, 2002). Uma história não contém todos os detalhes que são necessários para codificar o comportamento. Para isto, foram realizadas conversas diretas com o cliente.

CARTÃO DE HISTÓRIA E TAREFA

Data:	23/04/2002	Tipo de Atividade		Nova:	X	Dificuldade	Valor:
Número da História		001 – Calcular GID				Prioridade do Usuário:	
Referência Anterior:			Risco:			Estimativa do Técnico:	
Descrição da Tarefa:		Calcular a GID com base no grupo que o professor está. O grupo é especificado pelo regime de trabalho e número de horas/aula do professor, num total de 5 grupos. Sendo que para os grupos I-II-II calcular a GID, para o grupo IV dar 48 pontos (calcular somente se o professor solicitar) e o grupo V não recebem GID.					
Notas:		Os professores dos grupos I-II-III são atribuídos um total de pontos multiplicando-se 80 pontos vezes o número de professores de cada grupo.					

Acompanhamento da Tarefa:

Data	Estado	Para Realizar	Comentário
23/04	Questionar	OK	Quais informações guardar mensal/semestral
30/04		OK	Guardar mensal: horas/aula e semestral: avaliação quantitativa e participação projetos

Figura 1: Cartão de história e tarefas

4.5 Estimativa, priorização e planejamento

Assim que as histórias foram coletadas, o cliente as classificou por prioridade: alta, média, baixa. As histórias sinalizadas como "alta" eram imediatamente necessárias, o sistema não teria validade sem elas. As histórias marcadas como "média" eram requeridas, mas a sua ausência poderia ser contornada por algum tempo. As histórias sinalizadas como "baixa" seriam interessantes, mas apenas após a conclusão das outras histórias.

Posteriormente, o time de programação estimou as histórias pontuando em semanas, se uma história consumisse mais de três semanas o programador retornava a história para que o cliente a dividisse em histórias menores. Definidas as estimativas e prioridades, foram designadas as iterações. Decidiu-se por duas iterações (ver Tabela 1).

É importante observar que os cartões de história formaram a funcionalidade básica do aplicativo. Desta forma, optou-se por constituir a primeira versão do sistema. Provavelmente, outras versões serão necessárias para aperfeiçoamento e complemento de recursos.

Tabela 1: Dados dos cartões de história

História	Prioridade	Estimativa	Iteração
001 - Calcular GID	alta	2 semanas	1º
002 - Coletar dados	Alta	1 semana	1º
003 – Exceções do cálculo	média	1 semana	1º
004 – Gerar pontos	média	2 semanas	1º
005 – Imprimir resultados	média	1 semana	2º

Logo após, a realização das prováveis estimativas, as histórias foram divididas em tarefas. Com o objetivo de maximizar o trabalho, foi elaborada uma lista simplificada de tarefas. Esta decisão foi tomada para facilitar o trabalho do time, uma vez que, existia apenas um par de programadores.

4.6 Teste de aceitação

Para validar cada característica desejada, o cliente realizou os testes de aceitação manualmente por meio da interface gráfica do usuário (GUI – *Graphical User Interface*). Foi assentado que a GUI era suficientemente simples para que os testes de aceitação fossem executados à mão. Além disso, um teste de simulação da GID foi construído na planilha do Microsoft Excel, cujo objetivo era mostrar que os resultados esperados estavam corretamente implementados.

Para Jeffries (2001), os melhores times XP valorizam os testes do cliente da mesma forma que consideram os testes do programador. Uma vez que os testes funcionem, o time deve manter esta versão como correta.

4.7 Desenvolvimento orientado por teste

O time realizou o "desenvolvimento orientado por teste", trabalhando em ciclos muito pequenos, onde foram acrescentados os testes. Primeiramente, se escreveu o teste que especificou e validou o comportamento desejado e, em seguida, foi criado e desenvolvido o código que o implementou. Desta forma, o time produziu o código com grande parte dos testes cobertos, o que foi um grande avanço.

A construção dos testes foi iniciada com a definição de "*procedures*". O objetivo do primeiro teste foi examinar a pontuação docente obtida através da fórmula para calcular à carga horária. O teste seguinte foi modelado para validar os pontos obtidos através da relação de responsabilidades docentes, conforme o número de alunos. Um terceiro teste foi elaborado para a avaliação quantitativa das aulas ministradas, limitando pontos por assiduidade e atribuições didático-pedagógicas. O último teste realizado para atender a história "001 – Calcular GID", foi atribuir pontos pela participação em projetos de pesquisa utilizando a fórmula conforme o grupo do professor.

Cada teste, inclusive o código, foi desenvolvido de forma incremental, incluindo um teste de cada vez e fazendo com que ele fosse validado. Esse padrão de desenvolvimento se repetiu em toda a aplicação. Embora, tenham sido realizados testes somente para as histórias e tarefas que o time considerou como cruciais para o sistema.

Neste contexto, Jeffries (2001) enfatiza que o retorno dos testes realizados é muito importante no desenvolvimento do sistema, pois um bom resultado exige bons testes. Desta forma, o sistema da GID foi construído.

O sistema entrou em produção após a execução de todos os testes de aceitação realizados pelo cliente. Desta forma, um projeto pequeno e simples foi desenvolvido no modelo do XP.

É importante ressaltar que alguns requisitos surgiram à medida que o desenvolvimento progrediu. Um requisito que se pode destacar, foi à habilidade de simular o cálculo do professor com dados fictícios. Neste caso, não foi necessário alterar muito o planejamento. A nova história foi incorporada adicionando-se mais uma iteração. A seção seguinte apresenta a análise dos resultados, onde são descritos os pontos positivos e negativos do XP.

5. ANÁLISE DOS RESULTADOS

Com a aplicação dos passos descritos anteriormente, procedeu-se à avaliação do processo XP com o objetivo de produzir subsídios para abstrair os pontos positivos e negativos.

5.1 Pontos Positivos

Pode-se considerar que XP é adequado em projetos onde os clientes não sabem exatamente por onde iniciar o desenvolvimento e que a probabilidade de mudarem de idéia é grande. Uma vez que o software é rapidamente produzido e que o projeto de desenvolvimento recebe *feedback* rápido, apenas parte do planejamento é realmente perdido. Então, se o projeto é feito para atender as variações internas e de requisitos, o XP parece o mais adequado.

Isto encurta o ciclo para entrega de uma versão, sendo uma das forças primárias do XP. Realmente, isto é muito positivo: os clientes vêem rapidamente por aquilo que eles pagaram, embora incompleto, e os desenvolvedores permanecem mais interessados e na trilha certa.

Outra importante característica e que também é um dos principais *slogans* do XP é: sempre faça a coisa mais simples que poderia possivelmente funcionar. O pensamento que Beck (2000) mostra, ao longo do seu livro, que a simplicidade deve ser sempre maximizada. Percebeu-se que, a busca da simplicidade no sistema GID trouxe redução de tempo para o seu término, quando comparado a um projeto complexo.

A integração contínua é outra idéia muito interessante para desenvolver um projeto. O fato de o XP trabalhar com pequenas versões e que todo o código é testado, dá uma visão confortável de progresso e mantém os desenvolvedores entusiasmados tendo as tarefas à mão.

5.2 Pontos Negativos

Um problema importante de implementação com XP é que ele é um modelo difícil de ser usado quando se pretende vender serviços para outras companhias. Isto fica evidente quando o cliente deseja um levantamento explícito de requisitos ou, pelo menos, queira um preço fixo e o tempo de produção estimado. Convencer o cliente a contratar um serviço sem estas informações é muito difícil.

Em relação ao custo de mudanças ser baixo com XP, existem controvérsias. McBreen (2003), afirma que não existe nenhum dado real, para uma afirmação ou para a outra, que estime o custo de mudanças em software moderno. Realmente, estimar o custo de uma mudança de requisitos não é fácil porque depende do impacto que irá refletir no sistema.

A metodologia XP tem uma propensão para tentar convencer por choque. Nega muitas práticas que foram tentadas por décadas, algumas consideradas eficientes como a documentação e o levantamento de requisitos. A análise do problema, por exemplo, nunca é mencionada como algo para ser realizado. Os clientes devem ter uma boa idéia das "histórias" que eles precisam escrever inicialmente. A intenção de integrar o cliente no processo, escrevendo histórias, pode exigir muito esforço do cliente.

Neste contexto, destaca-se também o problema da ausência de uma fase de engenharia de requisitos mais específica. É importante pensar sobre o problema. Isto não significa dizer que se

deva passar tempo excessivo com requisitos e com planejamento, especialmente se eles mudam com freqüência. Porém, uma idéia global do problema a ser resolvido e um caminho para a solução acordados são sempre boas vantagens.

5.3 Considerações sobre XP

O XP parece ser adequado para projetos onde o ritmo durante o desenvolvimento é rápido. Quando o processo de desenvolvimento amadurece e menos funcionalidade é solicitada, não existe nenhuma especificação sobre como o XP se comporta em uma manutenção reduzida no desenvolvimento. Este problema pode não parecer óbvio, então pode-se esclarecer que, pouca ou nenhuma documentação significa "o código" e que o conhecimento do projeto deve estar confiado "nas pessoas". Infelizmente, estes recursos não são sempre confiáveis. É comum alguns desenvolvedores deixarem o projeto (isto é previsto em XP), mas ocasionalmente todos ou a maioria dos desenvolvedores deixam o projeto. Então, como ninguém sabe explicar como o código funciona, a barreira para a entrada de um novo desenvolvedor é grande. Porém, se existir documentação e que esteja corretamente mantida, mesmo que informal, pode ajudar a reduzir este problema. Esse é um problema que o XP evita mencionar.

Boehm (*apud* Ambler, 2002) mencionou que um projeto documentado ajuda um perito externo a diagnosticar problemas. Entretanto, Beck (*apud* Ambler, 2002) discorda, e expõe que um perito externo passa tempo considerável diagnosticando projetos e acaba incomodando as pessoas como detalhes pouco significativos e não técnicos na documentação.

O XP confia muito na possibilidade de aplicar a técnica de *refactoring* e *retrofitting* (reajustar) no código existente com o objetivo de aumentar a funcionalidade. Isto pode funcionar se as mudanças forem simples. Neste sentido, destaca-se que 'simples' é um valor percebido e não um objetivo, quando o assunto é desenvolvimento de software. Pequenas mudanças podem causar sérios impactos e um planejamento adequado de pré-codificação poderia ajudar a resolver este problema.

Em relação aos testes, se uma grande parte do projeto exigir implementação de interface com o usuário, pode ser um problema para os desenvolvedores criarem testes automatizados. Obviamente, os testes podem ser trabalhados manualmente, embora mais incômodos. Porém, o XP conta com desenvolvimento orientado por teste; problemas com teste podem significar desenvolvimento lento.

McBreen (2003) critica a posição de que um bom time XP é pelo menos seis vezes mais produtivo que um time de engenharia de software tradicional. Além disso, os programadores do XP são considerados todos profissionais capacitados e com muita disposição para executar uma tarefa. Realmente, pode-se dizer que alguém não capacitado ou não disposto não deve fazer parte do time. Entretanto, em algum momento, um desenvolvedor poderá passar por uma fase ruim ou não estar disposto a executar determinada tarefa que lhe foi designada. Em um grupo, isto comumente pode acontecer. Uma solução para tarefas que ninguém quer, ou que estão sendo mal executadas é realizar uma votação ou criar normas de negociação. Mas, isto pode gerar conflito com a idéia de livre escolha e de propriedade que o XP sugere.

É importante observar que a integração contínua não é uma idéia que surgiu com o XP e todos processos modernos que adotam ciclos de vida como o modelo em espiral, o RAD (*Rapid Application Development*) ou o modelo iterativo, empregam a filosofia de integração contínua.

6. CONSIDERAÇÕES FINAIS

Neste trabalho foram apresentadas algumas reflexões fundamentais relativas à importância do processo de software, principalmente como base para o levantamento de requisitos, para o planejamento de projeto e para o apoio ao desenvolvimento. Foram apresentados também, os

fundamentos teóricos do processo XP, bem como os conceitos relacionados à Engenharia de Software. Por fim, foi aplicado e demonstrado o processo XP, mostrando todo o ciclo evolutivo, desde a concepção até a entrega da primeira versão. Esta aplicação gerou subsídios para avaliar o processo nos pontos positivos e negativos.

Esse trabalho espera ter deixado duas contribuições principais. A primeira delas está relacionada à exemplificação textual e gráfica da aplicação do processo. A segunda contribuição está relacionada à busca constante de um modelo de processo que possa ser aplicado de forma "produtiva" em um ambiente de desenvolvimento de softwares. Uma das barreiras na aplicação de determinadas metodologias está relacionada ao excesso de recursos e controles que a mesma requer, pois acaba demandando um tempo excessivo à construção e atualização de artefatos, gerando custos. Considerando esse tipo de dificuldade, a comunidade de Engenharia de Software busca alternativas, como às metodologias ágeis (XP) e as metodologias práticas ou intermediárias (ICONIX). A definição de um processo ideal, que seja produtivo e que proporcione garantia de qualidade ao software produzido, ainda é um desafio a ser enfrentado e vencido pelos pesquisadores da Engenharia de Software. Enfim, existe um longo caminho a ser percorrido na busca do estado da arte para o processo de software.

REFERÊNCIAS BIBLIOGRÁFICAS

AMBLER, Scott W. **The Agile Edge: Duking it Out. Software Development**. Canadá, v.10, n.7, p.53-55, jul. 2002.

ASTELS, David; MILLER, Granville; NOVAK, Miroslav. **Extreme Programming Explained**: Guia Prático. Rio de Janeiro: Ed. Campus, 2002.

BECK, Kent. **Extreme Programming Explained**: Embrace change. Reading, Massachusetts: Ed. Addison-Wesley, 2000.

BONA, Cristina. **Avaliação de Processos de Software: Um estudo de caso em XP e ICONIX**. Florianópolis, 2002. 122f. Dissertação (Mestrado em Engenharia de Produção) – Programa de Pós-Graduação em Engenharia de produção, UFSC, 2002.

CORDEIRO, Marco A. **Bate Byte 100 – Foco no Processo**, ago, 2000. Disponível em: <http://www.pr.gov.br/celepar/celepar/batebyte/edicoes/2000/bb100/foco.htm>. Acesso em: 20 jun. 2002.

HIGHSMITH, Jim. **Does Agility Work? Software Development**. Canadá, v.10, n.6, p.30-36, jun. 2002.

JEFFRIES, Ron. XP Magazine Contents: What is Extreme Programming?. **XProgramming.com – an Extreme Programming Resource**, nov, 2001. Disponível em: <http://www.xprogramming.com/xpmag/index.htm>. Acesso em: 11 mar. 2002.

MCBREEN, Pete. **Questioning Extreme Programming**. Indianapolis: Ed. Person Education, 2003.

OMG®. Object Management Group **- Unified Modeling Language Specification (2.0)**, 2001. Disponível em: <http://www.omg.org>. Acesso em: 18 jul. 2002.

OSHIRO, Adriane et al. Extreme Programming, um novo modelo de processo para o desenvolvimento de software, jul, 2001. Disponível em: <http://www.icmc.sc.usp.br/~percival/>. Acesso em: 03 abr. 2002.

REIS, Christian. A commentary on the Extreme Programming development process, ago, 2000. Disponível em: <http://www.async.com.br/~kiko/papers/xp/>. Acesso em: 04 abr. 2002.

SILVA, Alberto M. R.; VIDEIRA, Carlos A. E. **UML, Metodologias e Ferramentas Case**. Lisboa: Centro Atlântico, 2001.

WAKE, William C. Extreme Programming Explored. Reading, Massachusetts: Ed. Addison-Wesley, 2002.

WUESTEFELD, Klaus. Xispê – Extreme Programming, 2001. Disponível em: <http://www.xispe.com.br/index.html>. Acesso em: 18 mar. 2002.