Fernando Eduardo Bastos

Relacionamento entre Metodologias Ágeis (XP e Scrum) e o SW-CMM nível 2

Monografia apresentada a Escola Politécnica da Universidade de São Paulo para a conclusão do curso de MBA em Engenharia de Software

São Paulo
2003

Fernando Eduardo Bastos

Relacionamento entre Metodologias Ágeis (XP e Scrum) e o SW-CMM nível 2

Monografia apresentada a Escola Politécnica da Universidade de São Paulo para a conclusão do curso de MBA em Engenharia de Software

Área de concentração:
Engenharia de Software

Orientador:
Prof. Dr.: Reginaldo Arakaki

São Paulo
2003

RESUMO

Metodologias Ágeis e o Capability Maturity Model são dois modelos recentes que surgiram para endereçar a dificuldade de entregar produtos de software de alta qualidade. O CMM introduz cinco níveis de maturidade e dá em linhas gerais *o que* é necessário a ser feito para a organização , enquanto que as Metodologias Ágeis dizem *como* desenvolver software no nível de projeto. Juntos, esses modelos formam um framework para a estruturação do desenvolvimento de software à nível organizacional. Esta monografia tem como papel principal mostrar que é possível conciliar o CMM com as Metodologias Ágeis, mostrando relacionamento de suas práticas e uma das forma de combinar esses 3 modelos.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

CMM	Capability Maturity Model
Dod	U.S Department of Defense
KPA	Key Process Area
PTO	Software Project Tracking and Oversight
RM	Requirements Management
RUP	Rational Unified Process
SCM	Software Configuration Management
SEI	Software Engineering Institute
SPP	Software Project Planning
SQA	Software Quality Assurance
SSM	Software Subcontract Management
SW-CMM	Software Capability Maturity Model
XP	Extreme Programming

1 – INTRODUÇÃO e MOTIVAÇÕES

Objetivo: descrever o mundo atual no que diz respeita à fabricação de software, assim como demonstrar o que têm se feito para aprimorar este tipo de trabalho e falar um pouco sobre o que intuito desta monografia.

A indústria de software tem através de sua história uma grande quantidade de projetos que falharam. O sintoma mais comum da crise de software são os orçamentos e prazos acima do estimados, produtos cheios de defeitos e sistemas com sua complexidade aumentada. Para cooperar com a rápida mudança no ambiente de desenvolvimento de software, diferentes modelos foram tentados e alguns sucessos, conquistados. De alguns anos para cá algumas novas metodologias/modelos foram criados, entre eles o Capability Maturity Model (CMM), Extreme Programming (XP) e Scrum .

Extreme Programming (XP) é uma metodologia ágil para pequenas equipes de desenvolvimento de software, introduzida por Kent Beck [1] . Nos últimos anos, cresceu o interesse sobre este modelo, agitando o mercado de desenvolvimento de software. Ele é baseado em algumas práticas comuns de engenharia de software levado ao extremo assim como novas práticas criadas para agilizar o desenvolvimento de software. É composto basicamente de alguns valores, variáveis de controle , princípios e práticas.

O Scrum, como as outras metodologias ágeis, tem ganhado mais atenção pelo seu modo diferente de gerenciar o desenvolvimento de software. Ele é ágil na forma em que não há muitos detalhes definidos no seu padrão. Entretanto, o desenvolvimento é controlado usando equipes pequenas e incrementos pequenos para suportar uma comunicação rápida entre o cliente e o programador. Este padrão é uma instância da Modelagem Ágil da comunidade ágil, que é considerada um método para a programação apropriada para o mundo de requisitos voláteis da Internet e desenvolvimento de softwares web. Nesta monografia o Scrum será revisado e juntamente com o XP eles serão avaliados sobre a

perspectiva do SW-CMM, que é o padrão para as melhores práticas no desenvolvimento de software.

O Capability Maturity Model para Software (SW-CMM) tem as mesmas aspirações do XP, mas a sua origem é muito mais formal. Foi desenvolvido pelo Software Engineering Institute (SEI) da Carnegie Mellon University em uma resposta a um pedido do governo dos EUA para um método para certificar a qualidade dos desenvolvedores de software.

É comum falar que a indústria de desenvolvimento de software tem uma diferença bastante grande entre as melhores práticas e as práticas mais usadas. Uma das razões pode ser a falta de conhecimento dos processos e da engenharia de software como um todo. Esta monografia sugere que as metodologias ágeis (XP + Scrum) podem ser uma base madura para uma organização de desenvolvimento de software e que seria aperfeiçoado através das recomendações do CMM.

No próximo capítulo, será dada uma breve introdução sobre o XP, CMM e o Scrum. Para aqueles que já estão familiarizados com estes conceitos, pode-se começar pelo capítulo 3. Comparações entre o XP e o CMM já foram feitos [2] [3], mas o diferencial desta monografia é o acréscimo do SCRUM e um estudo de caso usando uma instância do RUP.

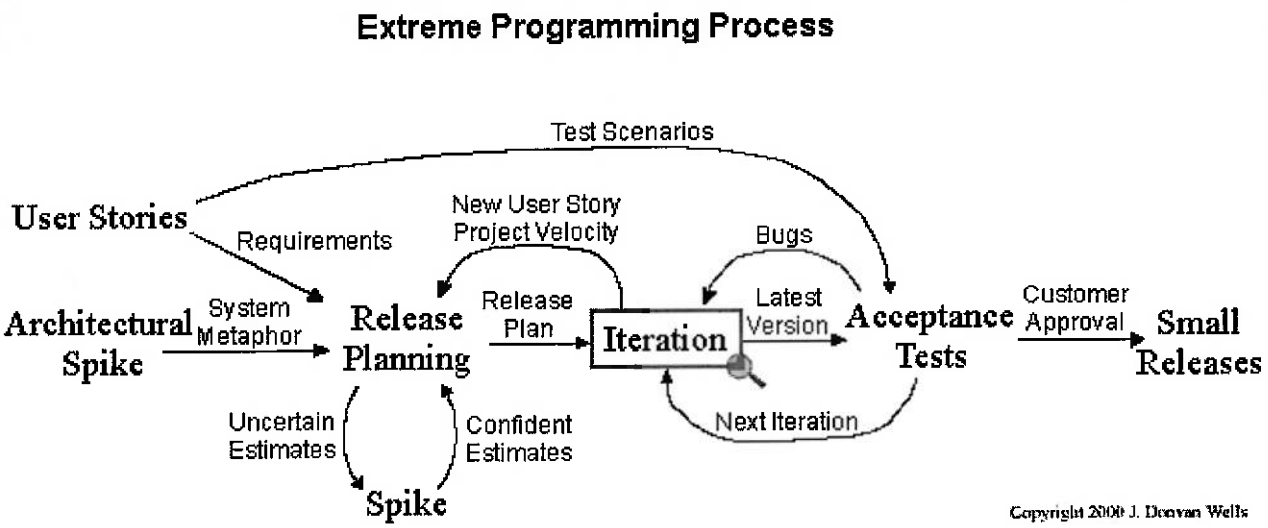
Conclusão: O que podemos concluir é que muito já foi feito para melhorar a qualidade do desenvolvimento do software, mas por diversas vezes tudo o que é feito/desenvolvido é jogado fora porque não atingiu o objetivo proposto ou porque o custo de instalação/manutenção é inviável.

2 – FUNDAMENTOS CONCEITUAIS

Objetivo: fazer uma descrição breve da teoria de Extreme Programming , Capability Maturity Model e Scrum. Mais detalhes sobre estes assuntos poderão ser encontrados na bibliografia anexa.

2.1 – Extreme Programming

Este capítulo fará uma sucinta apresentação do que é o XP (Extreme Programming).



A foto acima dá uma noção do ciclo de vida do XP, que é uma metodologia ágil para desenvolver software em ambientes de requisitos voláteis, gerando produtos de alta qualidade. É definido por valores, práticas princípios e variáveis de controle, que serão explicadas nos tópicos a seguir. Grande parte do texto deste capítulo, vem do livro Extreme Programming Explained[1] por Kent Beck.

Indivíduos e interações **mais que processos e ferramentas**
Trabalhar no software **mais que documentação extensiva**
Colaboração do cliente **mais que negociação de contratos**

Responder a mudança **mais que seguir um plano**

O XP tem duas premissas básicas:

- Aos programadores, o XP permite que eles trabalhem nos assuntos que realmente importam, todos os dias. Não serão encaradas situações sozinhas. E tomarão somente as decisões que lhes forem qualificadas;
- Aos clientes e gerentes, o XP entende que eles poderão tirar o máximo em programação durante a semana. A cada iteração eles serão capazes de ver progressos concretos e os objetivos sendo alcançados. Eles serão capazes de mudar a direção do projeto no meio do desenvolvimento sem ter aumentos exorbitantes de custos.

O XP é baseado nas seguintes opiniões:

- Programar pode ser divertido;
- Não se deve antecipar o futuro;
- É preciso feedback;
- Devem-se eliminar as perdas;
- Os papéis certos para as pessoas certas;
- Funciona somente para equipes pequenas;

2.1.1 – Valores

Kent Beck [1], um dos criadores do XP definiu quatro valores para o XP:

- Comunicação;
- Simplicidade;
- Feedback;
- Coragem;

Comunicação é facilitada através do pair programming, estimativas de tarefas, planejamento de iterações, etc. A meta da comunicação é prover um local onde as pessoas possam falar livremente sobre o projeto sem medo ou repressão.

Simplicidade pode parecer estranha, mas é comum na indústria de software geralmente complicar os desenvolvimentos dos projetos. O propósito do projeto não é mostrar que o desenvolvedor tem capacidade técnica, mas sim capacidade de entregar ao cliente coisas que possam agregar valor ao seu negócio. Não é necessário ficar recriando algoritmos,

muito menos sistemas de inteligência artificial para problemas que você pode resolver com algumas sentenças if, não é necessário também ficar usando EJB's ou XSLT, por exemplo, somente para deixar bonito e você poder colocar no seu curriculum. É sempre necessário escolher o melhor design, tecnologia, algoritmo e técnicas que irão satisfazer as necessidades do cliente para a iteração corrente do projeto.

Feedback é crucial e obtida através de testes de código, estória do cliente, pequenas iterações, entrega constante de artefatos, revisão de código, e outros métodos. Por exemplo, se você faz testes unitários constantemente, você estará constantemente recebendo feedback sobre a qualidade e produção de seus artefatos. Se o sistema está errado, você saberá rapidamente, ao invés de ter surpresas desagradáveis na noite anterior ao produto entrar em produção.

Coragem é tudo aquilo sobre ter coragem suficiente para fazer o que é correto. É normal em projetos, por exemplo, as pessoas terem medo de jogar código fora. O código pode estar muito mal escrito com o sistema tendo diversas falhas, mas mesmo assim o gerente e os desenvolvedores têm medo de jogar o código. Se você segue as regras do XP, esse problema não acontecerá. Quando você sabe que as mudanças que forem feitas não irão afetar o sistema, daí você terá confiança para refazer o código. Testar é a chave para a coragem.

2.1.2 – Princípios

- Prover feedback rápido;
- Assumir simplicidade;
- Fazer mudanças incrementais;
- Abraçar a mudança;
- Fazer trabalho de qualidade;

Prover feedback rápido é a idéia atrás de feedback. Quanto mais rápido você tem feedback, mais rápido você pode responder ao cliente e mais pode ser feito no processo de desenho do sistema. O feedback não acontece somente ao testar e integrar o sistema, mas também quando ocorrem as iterações curtas, dando ao cliente a capacidade de direcionar o projeto e manter o custo da mudança baixo, permitindo ao desenvolver assumir a simplicidade

Assumir simplicidade significa tratar cada problema como um problema simples até que se prove o contrário. Esta aproximação funciona bem, já que a maior parte dos problemas são de fácil resolução. Assumir simplicidade não significa fugir a parte de desenho, por exemplo , mas sim que você faça o desenho de somente aquela iteração. Como há um feedback rápido por parte do cliente com as mudanças de requisitos, não poderá ser gasto tempo refazendo o código que foi desenhado para ser resistentes a uma bomba atômica se o que você precisa na verdade é algo apenas que seja à prova d'água.

Fazer mudanças incrementais se encaixa perfeitamente com simplicidade. Não é preciso fazer grandes estudos para fazer o desenho, é sempre melhor fazer aos poucos. Por exemplo, imagine uma aplicação web que precisa ser refeita para melhorar a performance do sistema.

Ao invés de atacar o sistema inteiro, porque não fazê-lo incrementalmente, mudando as páginas que tem o maior tráfego primeiro, pegando o maior problema agora para minimizar o risco. Testando e fazendo o deploy aos poucos, decisões poderiam ser tomadas com o feedback ao vivo, coisa que não aconteceria se as mudanças não fosse incrementais.

Abraçar a mudança, quantas vezes não é ouvida a frase em que um gerente de projeto declara que os requerimentos já foram feitos, e o cliente não pode mudá-los? Esta é uma boa argumentação, mas o sistema não será feito para o cliente?

No XP, os desenvolvedores abraçam a mudança. Mudança é esperada, pois você está agregando valor ao negócio do cliente incrementalmente. O cliente tem tempo suficiente para te dar um feedback rápido e requisitar mudanças. Este processo envolve a qualidade do sistema assegurando que o sistema que está sendo construído é uma das necessidades do cliente. Os clientes estão felizes, pois eles podem direcionar o projeto na próxima revisão antes do projeto ficar desviar muito de suas necessidades atuais de negócio.

Fazer trabalho com qualidade é uma das tarefas mais prazerosas do desenvolvedor. É importante, por exemplo, na hora de se fazer os testes que toda linha de código seja testada, fazendo com que não existe o mesmo código em várias partes do sistema, deixando o mesmo mais fácil para fazer mudanças e continuar desenvolvendo.

Existem alguns outros princípios menos centrais que serão somente citados aqui: aprendendo para ensinar, pequeno investimento inicial, planejar para vencer, experimentos concretos, comunicação aberta e honesta, trabalhar com o instint , aceitar responsabilidades, "pegar leve", métricas honestas.

2.1.3 – Variáveis de Controle

Este tópico trará uma breve descrição das variáveis de controle do XP.

- Escopo: o que deverá ser feito, ou seja, as funcionalidades a serem implementadas;
- Qualidade: significa atender às expectativas do usuário em termos de funcionalidades do sistema;
- Custo: o investimento de pessoal, equipamentos e outros;
- Prazo: a duração do projeto;

- Modelo Tradicional:
  - Tempo
  - Escopo
  - Custo

  - Manipula-se a **Qualidade**

- Modelo XP:
  - Tempo
  - Qualidade
  - Custo
  - Manipula-se o **Escopo**

Conforme quadro acima, é bastante comum fixar-se o escopo entre outras coisas, ou seja, durante o projeto é provável que o cliente perca as chances de fazer ajustes no que diz respeito ao negócio, já que com o passar do tempo o mesmo vai aprendendo mais sobre o negócio e sobre como quer que o sistema seja feito, fazendo com que o próprio fornecedor assuma riscos, cobrando muito mais alto para se proteger e fazendo com que o cliente tenha uma falsa sensação de segurança.

Com isso, há um certo desgaste quando da renegociação de contrato, de um lado o cliente, querendo incorporar o seu aprendizado e querendo mais funcionalidades e do outro o fornecedor, querendo trabalhar menos, pedindo renegociação do contrato e levando a um inevitável desgaste entre fornecedor/cliente, e que perde no final é o cliente, que tem a qualidade de seu software sacrificada e acaba perdendo duas vezes, já que ele não obtém o que precisa e o que obtém costuma ter baixa qualidade.

No XP, com a premissa de que o escopo é variável, a qualidade de software é mantida durante todo o projeto, com as suas funcionalidades implementadas de forma simples, de maneira rápida e com a melhora qualidade possível. Na possibilidade de se fazer ajustes ao longo do projeto, as mudanças serão feitas sem nenhum desgaste no relacionamento cliente/fornecedor. O aprendizado do cliente é incorporado e pequenos ajustes serão feitos. Dessa forma, o cliente vai saber quanto vai gastar, quanto tempo vai

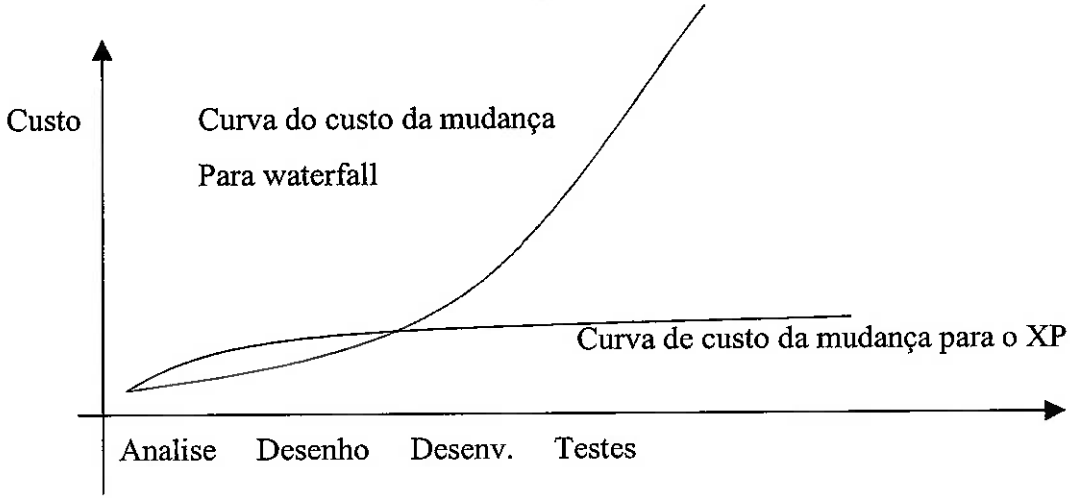
durar e o nível de qualidade. Acima de tudo, ele saberá quais as funcionalidades que irão gerar mais valor, já que elas serão implementadas de formas simples, de maneira mais rápida possível e com a melhor qualidade possível.

2.1.4 – Custo da Mudança

Este tópico trará uma visão já conhecida sobre o estudo de caso do custo da mudança de um software.

Uma séria de fatores ajuda no aumento do custo de mudança no decorrer do projeto de software. Um dos maiores é o excesso de documentação, já que as necessidades são informadas através de documentos, e no futuro sempre será preciso mantê-los atualizados. Com isso se gasta muito tempo com documentos, sobra pouco tempo para codificar e quando se codifica, descobrem-se falhas nos documentos.

Uma das constantes no universo da Engenharia de Software é que o custo de mudança de um sistema cresce exponencialmente com o passar do tempo.



Uma séria de fatores ajudam o XP a manter a curva de custo da mudança como o indicado no gráfico acima, como por exemplo:

- Design simples;

- Test first Design;
- Refactoring;

2.1.5 – Práticas

Este tópico trará uma breve descrição sobre as práticas que o XP emprega para que o desenvolvimento de um projeto seja ágil e funcione de acordo com as premissas do XP.

Pair Programming: é provavelmente a prática mais revolucionária no XP – e é usualmente a mais difícil de ser entendida pelos gerentes. Alguns podem pensar que é caro trabalhar em duplas, mas o quão caro pode ser você ter que substituir todo o sistema, pois está impossível de dar manutenção e cada mudança causa terríveis efeitos colaterais? Isso ocorre sempre, e geralmente não é bom que isso ocorra.

É conhecido pelos profissionais que trabalham com XP que pair programming funciona. Em primeiro lugar, melhora a comunicação entre os membros da equipe. Você pode, por exemplo, trabalhar com um desenvolvedor um dia e trabalhar no dia seguinte com outro desenvolvedor. Um outro aspecto bastante importante nesta prática é que você também pode trabalhar em dupla com alguém que tenha algum conhecimento que você não tem e que no final do projeto poderá ter, já que ao passar do tempo, com a comunicação entre ambos é inevitável que o conhecimento se dissemine entre todos os desenvolvedores.

A questão da qualidade também é outro fato bastante importante já que o feedback é constante entre os dois desenvolvedores, e ambos se asseguram que os testes estarão sempre sendo rodados, assim com o refactoring e os padrões de código sendo seguido de acordo com as regras estabelecidas.

Simple Design: O sistema deve ser desenhado da maneira mais simples possível durante todo o projeto. Complexidade extra é removida assim que descoberta. Simplicidade é a chave.

Os desenhos simples são aqueles que passam em todos os testes, não tem lógica duplicada, e expressa o propósito de todos os desenvolvedores. Este passo, caminha junto ao de small releases. Se a arquitetura não está sendo bem expressada e está sendo construída para antecipar o futuro, o sistema não será entregue no tempo adequado. A melhor forma de antecipar as necessidades de futuro do cliente será dando um sistema funcionando e receber o feedback dele. Muitos clientes não sabem exatamente o que querem até que eles tem algum artefato em mãos.

Small Releases: São entregas feitas ao cliente em um período médio de 1 a 3 semanas, fazendo com que valor seja agregado ao cliente o mais breve possível, ou seja, permite um feedback mais rápido por parte do cliente.

User Stories : User Stories podem ser definidas como uma versão mais simples de Use Cases . Geralmente eles são escritos em um cartão e somente o cliente pode escrevê-los e dar as prioridades necessárias para serem desenvolvidos. Os programadores serão responsáveis por estimar e depois desenvolver essas User Stories. Geralmente as estimativas são feitas por comparação, já que os desenvolvedores procuram por estórias semelhantes já implementadas, muitas vezes no início do projeto não há um histórico para comparação e nesse caso a estimativa é mais incerta. Com isso, as primeiras estimativas são mais determinísticas , não levam em conta incerteza, quase sempre são irreais e quase sempre são simples chutes.

No XP, é o cliente quem deve priorizar as estórias para que seja agregado valor ao negócio o mais rapidamente possível, já que ele irá se basear nas suas necessidades e nas estimativas dos desenvolvedores.

Segue um exemplo de um cartão para se escrever User Stories:

Date: _____ Type: New__ Fix__ Enhance__ Func Test:_____

Story #: _____ Priority User:____ Tech:_____

Prior Reference:_____ Risk:_____ Tech Estimate:_____

Task Description:

Notes

Task Tracking:

Date	Done	To Do	Comments

Customer on-site : o cliente estará sempre on-site (ou disponível) e fará parte do time. Esta pessoa deve ser um usuário e será ela que proverá as User Stories, responderá as questões e terá como prioridade direcionar as User Stories que deverão ser desenvolvidas.

System Metaphor: é uma forma do desenvolvedor falar sobre a estrutura do software de uma forma conveniente e que seja de fácil memorização. Ela dá a forma do sistema como um todo. Às vezes, ela pode ser mais voltada ao mundo físico, quando usa palavras como contas e objetos e outras vezes pode ser mais abstrata como janelas e células. A idéia é que o cliente e o desenvolvedor tenham a mesma visão do sistema e sejam capazes sobre o sistema em um dialeto comum.

The Planning Game: é usado para planejar releases, iterações e tarefas (tasks). Serve também para ter certeza que está sendo trabalhado nos valores mais importantes durante o decorrer do tempo. Não serve para predizer o futuro. São nessas reuniões que o cliente escolhe quais serão as user stories que deverão ser implementadas neste release.

No XP existe Release Planning, que é quando o cliente recebe um orçamento dos desenvolvedores, seleciona as estórias prioritárias que possam ser implementadas dentro do orçamento. O cliente pode trocar as estórias durante o release.

Dentro de um release existem várias Iterações, e no Iterarion Planning e não pode haver troca de funcionalidades durante uma iteração e a velocidade é medida através da quantidade de estórias que a equipe consegue implementar em uma iteração.

De posse das estórias a serem implementadas, os desenvolvedores respeitam as prioridades dos clientes, já que o projeto é dividido em releases e iterações. Existe um livro específico para cuidar do planejamento no XP – PlanningXP[7].

Test first design: os testes são especificados antes do código. Isto faz com que os desenvolvedores planejem os seus trabalho usando o ponto de vista do cliente (lembrando que os testes são escritos pelos clientes).

Unit Test é o nome dado à verificação feita sobre cada parte do código, e cada código possui um unit test associado a ela. Existem ferramentas para automatizar o código, e quando um novo código é inserido no sistema, todos os testes são executados.

Acceptance Teste é o teste que o cliente especifica. São testes de aceitação para cada funcionalidade de uma iteração. O sistema deve passar nestes testes ao término da iteração.

Coding Standards: todo o time deve-se aderir à um conjunto de padrões para o desenvolvimento de código. Deve-se chegar neste padrão através de um consenso, e não pode haver reformatação para atender um ou outro gosto individual.

É necessário, por exemplo, que em um coding standard tenha regras para o uso de threads, construtores da linguagem, inexistência de lógica duplicada, etc. O coding standard deve ser muito mais que um guia de como e onde colocar vírgulas ou aspas; ele deve mostrar estilo e práticas comuns que o seu time deve seguir.

Collective Ownership: os módulos não pertencem a somente um desenvolvedor. Todo desenvolvedor tem o direito de modificar qualquer módulo no projeto. O código pertence ao time inteiro, e todos os programadores estão familiarizados com todo o código.

Refactoring: significa fazer mudanças estruturais que preservam a semântica. É usado para melhorar o desenho do sistema, deixar o software mais fácil de entender, facilitar o acesso aos bugs, evitar duplicação de código, métodos longos e comandos do tipo "Switch".

Continous Integration: é um conceito fundamental. Para que esperar até o fim do projeto para ver como as coisas irão funcionar. Pelo menos uma vez por dia o sistema deve ser totalmente montado e testado, incluindo as últimas mudanças. Fazendo isso sempre, será mais fácil de identificar quais mudanças derrubaram o sistema, e os ajustes poderão ser feitos diariamente, ou seja, não será necessário que se tenha uma pilha de modificações para daí você começar a resolvê-las.

40 hour week: a regra é, não mais de uma semana consecutiva de trabalho com horas-extras. Se for necessárias horas-extras para acabar a iteração ou release, uma semana é aceitável, mas se for necessário mais do que uma semana o processo está sendo mau executado e a iteração precisa ser re-planejada.

2.2 – Capability Maturity Model (CMM)

O CMM é um conjunto de regras ou uma estrutura que descreve os elementos chaves de um processo de software eficaz. É um caminho de melhoramento evolucionário (cinco níveis de maturidade) para organizações de software mudarem de um processo de software imaturo, ad hoc, para um processo maduro, disciplinado. O CMM foi desenvolvido e melhorado por centenas de experts em engenharia de software sob a supervisão direta do SEI. Grande parte do texto abaixo teve como referências aulas de Rosely Sanches[3].

A maturidade dos processos de software de uma organização influência na sua capacidade de atingir metas de custo, qualidade e cronograma.

O CMM teve o início de seu desenvolvimento em 1986 quando o governo federal dos EUA solicitou um modelo de maturidade de processo, em junho de 1987 houve uma liberação de uma breve descrição do modelo, e em setembro de 1987 a versão preliminar do questionário de maturidade.

Em junho de 1991 saiu a primeira versão do CMM e em 1993 o modelo de maturidade evoluiu para um modelo completamente definido, usando o conhecimento adquirido das avaliações de processo de software e de extensivo retorno das indústrias e do governo.

O SEI tem a seguinte premissa básica:
A qualidade de um software produto é profundamente determinada pela qualidade do processo de desenvolvimento e de manutenção usado para construí-lo.

Basicamente, o CMM consiste de cinco níveis que orientam uma organização em como amadurecer seus processos de software. Os cinco níveis de maturidade descrevem fundamentos sucessivos para melhoria contínua do processo e definem uma escala ordinal para medir a maturidade de um processo de uma organização.

O CMM é bastante abstrato, pois não determina como o processo de software é implementado pela organização, descrevendo o que normalmente se espera em um processo de software e não como o processo é implementado.

O CMM pode ser usado com diferentes objetivos:

- Identificar pontos fracos e fortes na organização;
- Identificar riscos ao selecionar subcontratado;
- Entender as atividades necessárias para iniciar um programa de melhoria de processo de software.

Como dito anteriormente o CMM está estrutura em 5 níveis de maturidade que consistem de 18 KPA's que são divididas em 52 metas e estes últimos são divididos em 316 práticas chave.

Key Process Areas (KPAs) são áreas chaves do processo que identificam um grupo de atividades relacionadas que, quando realizadas coletivamente, atingem um conjunto de objetivos considerados importantes para o aumento da capacitação do processo.

As metas (goals) de cada KPA resumem as práticas-chave das KPAs de processo e podem ser usadas para determinar se uma organização ou projeto implementou efetivamente a área chave de processo.

No nosso caso, estudaremos somente como partir do nível 1 (inicial) para o nível 2 (repetível) .

No nível um (inicial) é o caso onde poucos processos são definidos e o sucesso depende de esforços individuais e heróicos. O gerenciamento de software é uma caixa preta, os requisitos fluem para dentro, o produto de software é normalmente produzido através de algum processo disforme e o produto flui para fora com algumas funcionalidades.

No CMM nível um a organização não provê um ambiente estável para o desenvolvimento e manutenção de software. Cronogramas e orçamentos são freqüentemente abandonados por não serem baseados em estimativas realísticas. Em uma crise para cumprir cronograma, etapas planejadas do ciclo de vida não são realizadas prejudicando a qualidade do software. O desempenho é basicamente em função da competência e heroísmo das pessoas que fazer o trabalho. O processo de software é imprevisível, já que é constantemente alterado no decorrer do projeto.

Os maiores problemas com os quais se defrontam as organizações de software são gerenciais e não técnicos.

No CMM nível dois (repetível), os processos administrativos básicos são estabelecidos para acompanhar o custo, cronograma e funcionalidade. A disciplina de processo está em repetir sucessos anteriores em projetos com aplicações similares.

Está em vigor um sistema de gerenciamento de projeto. O processo de construção de software é uma série de caixas pretas com pontos de verificação definidos.

O nível repetível é caracterizado pela existência de um processo efetivo de planejamento e gerenciamento do projeto de software onde os controles sobre os procedimentos, compromissos e atividades são bem fundamentados. Os processos de planejamento e gerenciamento do projeto de software devem ser praticados na organização, documentados, treinados e controlados. Neste nível ainda não há preocupação com o processo de engenharia de software.

O planejamento e gerenciamento de novos projetos são baseados na experiência obtida com projetos similares, que tenham obtido sucesso no passado. Um fator relevante para a organização nesse nível é a dependência das experiências anteriores. O desenvolvimento de novos tipos de produtos pode causar um desequilíbrio no projeto, nas estimativas de custos e nos cronogramas.

2.2.1 Requirements Management (RM)

Esta kpa envolve o estabelecimento e manutenção de um acordo com o usuário ou cliente no que se refere aos requisitos para o projeto de software. As metas são:

- Os requisitos de sistemas alocados ao software são controlados para estabelecer uma baselina para uso da engenharia de software e gerência;
- Planos, produtos e atividades de software serão mantidos consistentes com os requisitos de sistemas alocados ao software;

2.2.2 Software Project Planning (SPP)

O propósito é estabelecer planos para a engenharia de software e para o gerenciamento de software. As metas são:

- As estimativas de software são documentadas e usadas para planejamento e acompanhamento do projeto de software;
- Atividades e compromissos do projeto de software são planejadas e documentadas;
- Grupos e indivíduos afetados concordam com os seus compromissos relacionados ao projeto de software.

2.2.3 Software Project Tracking and Oversight (PTO)

O propósito é de prover visibilidade adequada sobre o progresso atual, para que a gerência possa tomar ações efetivas quando a performance do projeto tiver um desvio sobre o plano do projeto. As metas são:

- Resultados e performances atuais são checadas contra os planos de software;
- Ações corretivas são tomadas quando resultados e performances atuais tem um desvio significativo sobre os planos de software;
- Mudanças no software são concordadas pelos grupos e indivíduos afetados;

2.2.4 Software Quality Assurance (SQA)

O propósito é prover visibilidade para a gerência sobre o processo que está sendo usado pelo projeto que está sendo construído. As metas são:

- As atividades de SQA deve ser planejadas;
- Aderência dos produtos de software e atividades, padrões, procedimentos e requisitos são verificados objetivamente;
- Grupos e indivíduos afetados são informados dos resultados e atividades do SQA;
- Tarefas que não podem ser resolvidas são endereçadas ao gerente sênior;

2.2.5 Software Configuration Management (SCM)

O propósito é estabelecer e manter a integridade dos produtos durante o ciclo de vida do projeto. As metas são:

- Atividades de gerenciamento de configuração do software são planejadas;
- Produtos de software são identificados, controlados e disponibilizados;
- Mudanças nos produtos de software são controlados;

- Grupos e indivíduos afetados são informados sobre o status e conteúdo do baseline do software;

2.2.6 Software Subcontract Management (SSM)

O propósito é de selecionar subcontractors qualificados e gerenciá-los efetivamente. As metas são:

- O contratante principal seleciona subcontractors qualificados;
- O contratante principal e o subcontractor concordam com os seus compromissos;
- O contratante principal e o subcontractor mantêm comunicação constante;

2.3 - Scrum

De acordo com Ken Schwaber[4] SCRUM é uma coleção de práticas, baseadas em vários valores, gerando princípios para a engenharia de software. SCRUM é uma forma ágil de gerenciar os esforços de projetos de software e tem notícia que está sendo usado desde 1990 . O XP teve a sua prática do Planning Game criada a partir do SCRUM.

O SCRUM poderá ser usado juntamente com um processo de desenvolvimento ágil, ou quando você está trabalhando com processos iterativos incrementais . Na maior parte dos casos, os requisitos de software serão voláteis ou incertos, e a meta principal é a de se desenvolver software. É necessária a participação ativa dos stakeholders para que o produto inicial seja produzido com a qualidade adequada.

2.3.1 – Práticas

O termo Scrum surgiu após uma analogia com o Rugby / Futebol Americano, Scrum é aquela parte de equipe responsável por pegar a bola e levá-la para frente. O Scrum tem poucas formalidades, com foco em promover a comunicação, formação de equipe e

controle de resultados, eliminando práticas de gestão de projeto desnecessárias, inadequadas e burocráticas, concentrando-se na essência do trabalho, possuindo todos os controles de gerenciamento necessários para que os desenvolvedores liberem produtos de qualidade, rapidamente.

De acordo com a teoria do Scrum, softwares são Produtos, que irão evoluir em versões ao longo de vários Projetos, e para isso criar o conceito de Produto é fundamental.

O *Product Backlog* (Requerimentos do Produto) é uma lista textual de pendências de toda natureza, advinda de diversas fontes e aprovada pelo Dono do Produto. Os requerimentos são aprovados pelo Dono do Produto, em caráter pró-ativo, que também estabelece Prioridade e HH Estimado, para cada requerimento, sendo que os Requerimentos para Produto devem ser disponibilizados para leitura fácil por todos os envolvidos.

Dá-se o nome de Sprint Backlog a lista de funcinalidades (features) designadas para aquele Sprint.

Projetos (ou *Product Releases*) são montados através de uma seleção de Requerimentos do Produto para serem atendidos em um período de tempo, a seleção é realizada por clientes, analistas e interessados de acordo com prioridade e em função do orçamento, na primeira reunião do Projeto (Sprint Meeting inaugural). Nesta reunião, deve-se também apresentar o Scrum a todos.

Aos Requerimentos selecionados para um Projeto, damos o nome de *Project Backlog* (Requerimentos do Projeto). Podem-se elaborar diagramas baseados em Pert ou de Gantt, mas estes diagramas não são usados no Scrum. Um planejamento pretendido de ciclos pode ser utilizado em substituição.

Scrum Meeting são reuniões diárias de 15 minutos que devem ser realizadas pela equipe e pelo Coordenador (podendo incluir um Scrum Master, nos primeiros projetos), para responder a três, e somente três, perguntas:

- O que cada membro da equipe fez no dia?
- Que impedimentos surgiram?
- Que cada membro da equipe pretende fazer no próximo dia?

Ao final de um Sprint, um produto final deve ser liberado, ou seja, software funcionando. Para tal, etapas clássicas de análise, projeto, desenvolvimento, testes e documentação.

Conclusão: O que podemos concluir deste capítulo é que o Capability Maturiy Model diz o que deve ser feito para que se possa atingir a maturidade no processo de desenvolvimento de software. Enquanto que o XP diz como tudo deve ser feito. Há sinergia entre os dois processos e o SCRUM, e é isto que será explorado nos capítulos seguintes.

3 – APLICANDO AS PRATICAS DO XP E SCRUM NO CMM NÍVEL 2

Objetivo: fazer uma abordagem da forma com que as kpa's do CMM podem se relacionar com as práticas do XP e SCRUM[14]. Todas as kpa's serão examidadas uma a uma e um mapeamento entre as práticas do XP e SCRUM e as kpa's do CMM será feito.

Uma das maiores limitações do XP, visto de uma maneiro organizacional, é que foi desenhada para pequenas equipes de desenvolvimento, e não tem ativo nenhum endereçamento para as responsabilidades gerenciais. É geralmente dito que o XP foi feito por programadores para programadores.

Pesquisas recentes procuraram consertar alguns desses problemas, como, por exemplo, o surgimento do Scrum[8], que é provavelmente o modelo mais similar aos valores e metas do XP . Enquanto o Scrum está recebendo bastante atenção por parte dos grupos de XP ao redor do mundo, ele ainda é pouco conhecido no universo de desenvolvedores. O mesmo, por exemplo, não pode ser dito sobre o CMM, que está sendo usado em milhares de empresas ao redor do mundo. A origem do CMM tem uma base muito com muito mais credibilidade que o scrum, considerando o tempo investido por diversas pessoas para o desenvolvimento dos respectivos modelos.

3.1 – O problema

Este tópico falará um pouco sobre o que vem acontecendo com o mercado atual de desenvolvimento de software, os problemas que costumam acontecer em um projeto de desenvolvimento de software.

O maior problema no desenvolvimento de software é o risco. Alguns exemplos de risco são[9]:

- Problemas com tempo – Quando a data final do projeto chega, você é obrigado a dizer para o seu cliente que a solução ainda demorará alguns meses para ficar completa;
- Projeto cancelado – Após inúmeros imprevistos/problemas o projeto é cancelado antes mesmo de ser implantado em produção;
- Custo alto de mudança – Após o software ser instalado em produção, o custo da mudança de uma funcionalidade fica tão alto, que o sistema precisa ser trocado;
- Alta quantidade de defeitos – Assim que o software é colocado em produção, a quantidade de defeitos é tão grande que o software não é usado;
- Problema conceitual – Após o software ter sido colocado em produção, o mesmo não é capaz de resolver os problemas de negócio que ele se propôs a resolver;
- Mudanças no negócio – Após o software ter sido colocado em produção, o negócio envolvido tomou outro rumo, e o que foi feito já não interessa mais ao cliente;
- Falsa riqueza de funcionalidades – O software mostra uma grande potencialidade de funcionalidades, mas nenhuma delas faz o cliente feliz ou eficaz no seu negócio;
- Problemas de pessoas – Após algum tempo de projeto, os bons programadores começam a não gostar mais do que fazem e mudam de projeto;

Nós próximos tópicos, as kpa's serão analisadas e o seu relacionamento com as práticas ágeis, feito.

É importante ressaltar que o CMM certifica a organização, não o processo, ou seja, poderiam ser levantadas questões sobre como seriam medidas habilidades e atividades neste caso.

Foi considerado para esta monografia que os membros da equipe seguiriam todas as indicações de livros sobre XP e Scrum, da mesma forma em que os estudos feitos aqui

seriam sobre o caso ideal, ou seja, a organização teria que seguir fielmente todas as práticas/valores/variáveis/princípios das metodologias ágeis.

Esta avaliação não tem a intenção de fazer uma análise profunda do CMM. Para uma avaliação mais completa seria necessário um maior conhecimento sobre o assunto envolvido e ter o skill necessário para avaliar um processo de acordo com o CMM, portanto esta monografia não reflete uma avaliação SEI SW-CMM e servirá apenas para fins educacionais.

3.2 – Requirements Management (RM)

Esta kpa é inicialmente endereçada nas reuniões de planejamento de iterações, onde o cliente e o equipe XP discutem os requerimentos de software. Os principais passos nesta reunião são[5]:

- Apresentação das users stories por parte do cliente.
- Estimativas das user stories e tasks por parte da equipe XP.
- Concordância entre ambas às partes.

As práticas do XP que estão ligadas com o assunto desta kpa são as user stories, on-site customer e integração contínua[10]. É interessante notar que no XP o senso de requerimentos vai bem além do que o CMM pede, ou seja, o mais importante é entregar ao cliente o que ele deseja, não somente os seus requerimentos. O que o XP quer dizer é que nem sempre os clientes são "felizes" ao colocar no papel todas as suas necessidades em um documento formal de requerimentos de software. O on-site customer facilita o diálogo entre o cliente e a equipe XP no que diz respeito a detalhes específicos que podem vir à tona com a implementação do projeto.

Em relação ao Scrum, o Scrum Master trabalha com os mebros do Scrum team para alocar e decidir quem irão fazer as tarefas para aquele sprint. É um processo colaborativo. Enquanto a equipe estiver motivada e completando as tarefas, o que

significa que os membros estão cientes de suas habilidades, e interessados no sucesso daquele sprint, as pessoas gostariam de fazer aquilo que elas acham que se dariam melhor, ou seja, não seria interessante que uma pessoa delegasse as tarefas a serem feitas já que assim haveria maiores chances da tarefa ter sido delegada para a pessoa errada.

Durante o sprint planning, os requisitos no product backlog são revistos com o time, incluindo o Scrum Master e o Product Owner, para determinar quais tarefas serão alocadas. Os requisitos serão usados para fazer os produtos de trabalho e guiar as atividades do Sprint team porque eles são a fonte principal dos requisitos. Após o Sprint começar, não haverá o acréscimo de nenhum requisito, assim como no XP. Atividades para gerenciamento dos requisitos alocados são revistos em períodos regulares com a gerência durante o sprint review. O scrum é bastante interessado em controlar o desenvolvimento e muitas das atividades são periódicas ao invés de baseadas em eventos.

No final da iteração, ou durante a reunião de planejamento de iteração, a equipe mostra ao cliente o estado atual do projeto, ajudando o cliente a saber o que está sendo feito.

3.3 – Software Project Planning (SPP)

Incialmente, esta kpa também é endereçada durante a reunião de planejamento de iteração, onde as estimativas das tasks e user stories são escritas no cartões de índices. Durante a reunião, as atividades e compromissos são bem definidos pela equipe XP, na forma de user stories e tasks.

O planejamento geral do projeto é feito através da prática XP de small releases em combinação com um plano de construção de release baseado nas user stories junto ao cliente. A estratégia de planejamento do XP (curtas iterações) é perto do provérbio de Watts Humphrey's "If you can't plan well , plan often"[3]. O último goal desta KPA, o

acordo do que foi prometido, são evidentemente executados através dos próprios programadores que pegam as tasks a serem implementadas durante a próxima iteração.

Em relação ao Scrum, o Scrum Master é o líder do time e responsável pelos work products feitos durante o período. Embora o Scrum Master não seja uma pessoa com o único intuito de delegar tarefas aos membros da equipe, ele(a) seria responsável por assegurar que um planejamento seja desenvolvido. O Scrum[15] especifica que um plano seja usado para os finais de releases e sprint , mas o plano para o desenvolvimento no sprint é deixado para o time fazer. O Scrum Master negocia com o cliente e muda os comprometimentos para que o grupo de engenharia de software participe da proposta de projeto da equipe.

O product e sprint backlogs são usados para indicar o que precisa ser feito. O gráfico do backlog é um método de controlar o desenvolvimento do projeto inteiro porque mostra a quantidade de esforço que ainda será gasto no projeto. O sprint backlog é também usado para controlar o progresso do sprint, e de ter uma maneira de controlar o progresso.

3.4 – Software Project Tracking and Oversight (PTO)

Esta kpa é parcialmente atendida na prática de planning game. O XP não é baseado em nenhuma documentação formal, mas isso não quer dizer que não possa haver documentação dentro do XP[6].

No final de cada iteração, (ou mais caso necessário) o responsável pelas métricas faz todo o trabalho de performance da iteração. É bastante comum em projetos XP que poucas/simples ferramentas sejam usadas para fazer este acompanhamento. Desta forma, o project-plan é correlacionado com o que foi projetado. Este hábito cumpre satisfatoriamente com o primeiro goal desta kpa.

É a tarefa do coach de assegurar que os programadores estão trabalhando eficientemente e efetivamente, nos casos de um programador não estar alcançando as suas estimativas, é importante para o coach encontrar uma solução para este problema o mais rápido possível, embora não exista nenhuma regra formal de como isto pode ser alcançado com o XP .

Mudanças aos compromissos no desenvolvimento de software[11], ou user stories, durante uma iteração são manuseadas através de comunicação entre desenvolvedores e o cliente, com a assistência do coach. Se por exemplo um desenvolvedor percebe que uma estimativa está errada, é possível quebrar a user story em duas ou colocar esta para ser implementada na próxima iteração.

No Scrum, o Scrum Master é a figura principal responsável por controlar o progresso do projeto. Em termos de incrementos, o Scrum provê um plano de desenvolvimento de software de alto-nível e, portanto não é possível que esta kpa tire nota 10, por exemplo. O Scrum Master é o responsável pelas tarefas de sprint que serão assignadas aos membros da equipe para cada sprint, que resultará na possibilidade para o gerente de projeto de software assignar explicitamente responsabilidades para o work product de software[15]. O Scrum Master é também a pessoa que iria se relacionar com outros grupos e, portanto qualquer mudança em potencial deveria ser feito com o Scrum Master e mais tarde, no Sprint review os stakeholders sentariam juntos para rever o que aconteceu durante o Sprint[12].

O Scrum tem um método controlado de comunicação na forma das Scrum meetings e Sprint reviews e a habilidade de checar as atividades, software e os status de comunicação.

Um schedule do projeto é checado com o backlog chart, que representa a quantidade de esforço requirido para finalizar um projeto. Este gráfico é atualizado e re-estimações são feitas durante o projeto quando nova informação é obtida.

3.5 – Software Quality Assurance (SQA)

Apesar de não haver uma equipe de SQA independente no XP, esta kpa é satisfeita através das práticas do pair-programming, toda a parte de testing, Coding standards e on-site customer.

A prática de criação dos testes antes da implementação ajuda muito no que diz respeito à qualidade de software[12], principalmente porque os testes funcionais serão escritos pelos clientes, ou seja, o desenvolvedor saberá exatamente qual a finalidade do seu software. Além da prática da criação dos testes antes da implementação, a prática de Coding standards ajuda muito a todos os desenvolvedores, e simplifica o seu trabalho quando é necessário que ele altere um código escrito por outro desenvolvedor.

Uma das formas de apresentar os resultados para quality assurance são feitos através de um gráfico que mostraria a porcentagem de falhas nos testes para cada versão. Este número geralmente ficaria próximo do zero.

Deve-se lembrar que a comunicação faz parte constante do XP, o que faria com que o terceiro goal fosse cumprido. E o quarto goal é facilmente cumprido já que não existe nada no XP que obstrua o que está sendo pedido.

O Scrum enfatiza o SQA de duas formas. O cliente (product owner) é a entidade que irá decidir a qualidade do produto. Entretanto o cliente pode não entender a verdadeira qualidade do produto até que o produto esteja sendo usado. E para estes casos, o Sprint review é uma forma de checar de que o produto esteja sendo construído corretamente[15].

Outra forma de assegurar a qualidade é a de seguir que os processos especificados pela organização estejam sendo usados corretamente, e neste caso é papel do Scrum Master

certificar-se que tudo funciona como previsto. Ele(a) deve ser o responsável para que todos sigam os procedimentos acordados.

3.6 – Software Configuration Management (SCM)

Esta kpa pode ser alcançada através das práticas de continuous integration, collective ownership, refactoring e small releases.

O CMM sugere que um grupo seja criado para gerenciar o baseline do software. Isto não será necessário em um ambiente XP, pois a responsabilidade do código é distribuída através de todos os desenvolvedores. As outras metas, são implicitamente suportados pelo XP, e para grandes equipes, talvez somente o collective ownership não funcione, já que um grande canal de comunicação deveria ser criado.

No Scrum, é mencionado que o product backlog tem que ser mantido e atualizado, assim como o backlog chart. Entretanto, não há detalhes de como manter estes ítens. No Scrum, não há um grupo específico para gerenciar a configuração, e também não tem detalhes de definição dos status dos work products , deixando sem informações qualquer tipo de auditoria.

3.7 – Software Subcontract Management (SSM)

Esta é a única kpa que é relevante somente para organizações que tem o seu desenvolvimento terceirizado, e é a única kpa que não precisa ser atingida para atingir este nível no CMM. Não há nada no XP que mencione software subcontracting, e nada em particular que previna isso.

Tabela criada por Mark C. Paulk[2] para o relacionamento entre CMM e XP
Inclusão dos dados do Scrum

Requirements management	*Software project planning*	*SW proj tracking and oversight*	*SQA*	*SW config management*
on-site customer	Planning game	Small releases	Pair programming	Collective ownership
continuous integration	Small releases	Scrum Master	Testing	Small releases
use of stories	Scrum Master	Scrum Meetings	Coding standards	continuous integration
Scrum Master	Product Backolog	Sprint reviews	On-site customer	Product Backlog
Sprint Planning	Sprint Backlog		Sprint review	

Legenda : *CMM* , XP , Scrum

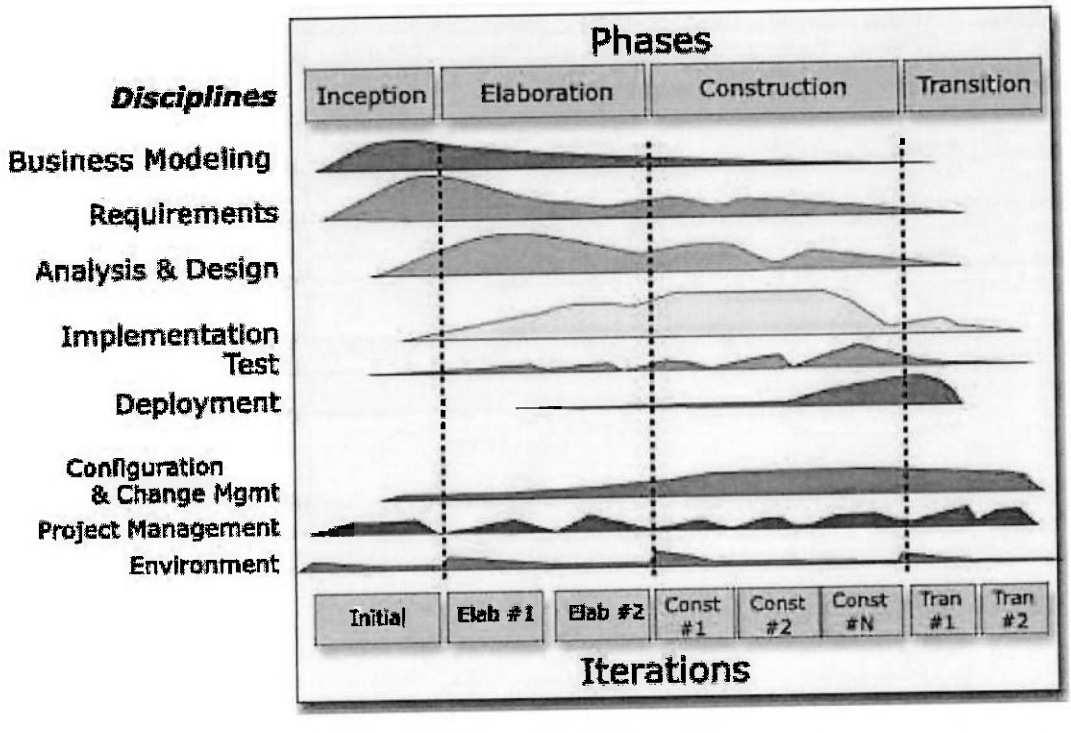
Conclusão: O que podemos concluir é que é factível o relacionamento do CMM com o XP e Scrum. Enquanto o XP é mais forte em atividades de desenvolvimento, o Scrum tem o seu grande potencial no que diz respeito às tarefas de gerenciamento. Fazendo uma mistura de ambos e extraindo o melhor é possível que um processo ágil se torne CMM nível 2.

4 – ESTUDO DE CASO USANDO XP

Objetivo: Neste capítulo estudaremos o caso de uma empresa que usa como processo de desenvolvimento de software uma instância do RUP e que tem como objetivo principal tornar o seu desenvolvimento de software ágil e cumprir com todas as metas de todas as kpa's do nível 2 do CMM. Tudo o que foi discutido no capítulo anterior será usado neste capítulo nas diferentes fases do projeto. O RUP foi usado como estrutura básica do processo derivado, e os fundamentos do XP e Scrum, foram usados para definir quais as atividades deveriam fazer parte deste processo de forma a garantir que o mesmo estivesse compatível com os idéias ágeis[10], já que esta abordagem é o tema principal desta monografia. A escolha do RUP como base é entendida aqui como principal veículo de expressão de desenvolvimento em termos atuais da indústria de software. Além disso, o RUP consolida muitas práticas de sucesso e é largamente utilizado pelo mercado de desenvolvimento.

4.1 – O processo em uso

Este tópico servirá para descrever o processo de desenvolvimento de software em uso (no caso será uma instância do RUP) descrevendo basicamente as suas fases e iterações.



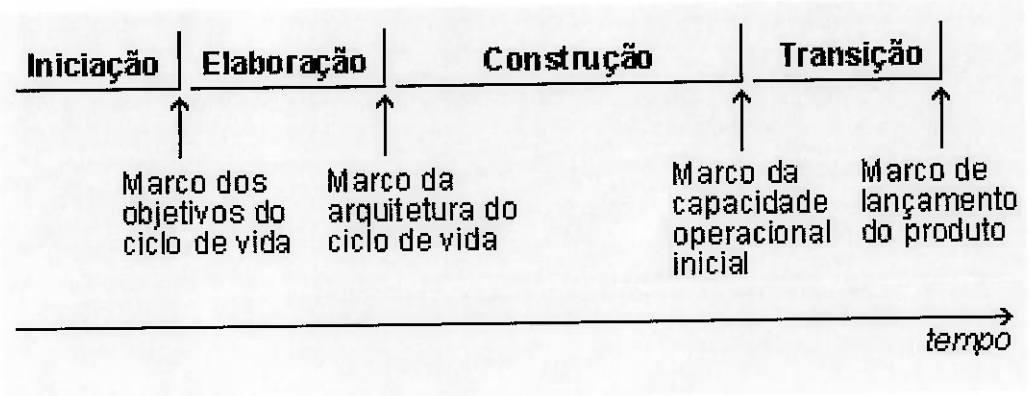
De acordo com a figura acima, percebe-se que o RUP tem 2 dimensões:

- Uma dimensão horizontal que representa tempos, e mostra os aspectos do ciclo de vida do processo;
- Uma dimensão vertical que representa as disciplinas, que são agrupadas logicamente pela sua natureza;

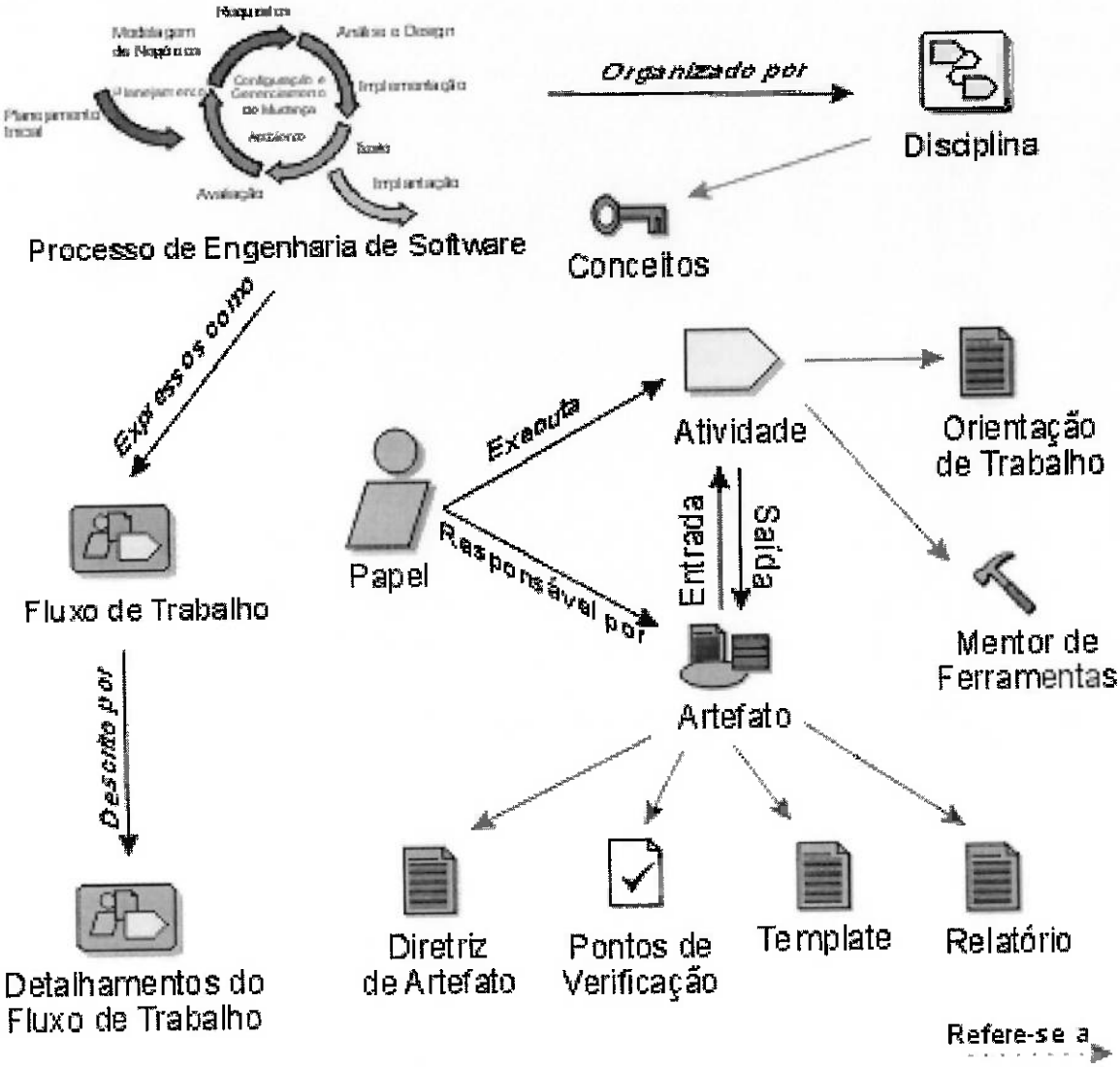
O RUP (Rational Unified Process) é um processo de Eng. de Software que oferece uma abordagem em disciplina para atribuir tarefas e responsabilidades dentro de uma organização de desenvolvimento. Ele é baseado nas 6 melhores práticas do desenvolvimento de software:

- Desenvolvimento Iterativo;
- Gerencia de Requisitos;
- Arquitetura de Componentes;
- Modelagem Visual;
- Verificação contínua da Qualidade;
- Gerencia de Mudanças;

O RUP tem os seguintes milestones:



Os principais conceitos do RUP são:



Mais do que um simples processo, o RUP é um framework e definir a instância deste framework adequada para a realidade particular de cada equipe não é uma tarefa fácil. Desta forma, XP e Scrum funcionam como balizadores[13], ajudando a definir uma instância suficiente do RUP para projetos com características ágeis.

Uma vez apresentadas às motivações e justificativas para se definir um processo com as características indicadas acima, a seqüência deste trabalho se ocupa em descrever passo a passo cada uma das partes componentes de tal abordagem proposta.

4.2 – Práticas e ciclo de vida

A partir da teoria apresentada no capítulo 2, o próximo passo é analisar quais as práticas de desenvolvimento serão seguidas e como elas serão dispostas pelo ciclo de vida desta nova abordagem.

A base para a definição do ciclo de vida foi extraída do RUP, o que estrutura a abordagem como um conjunto de **fases**, onde as atividades são realizadas inerentes a **disciplinas** comuns ao processo. Cada fase enfatiza, mais ou menos, uma determinada disciplina, embora a maioria das disciplinas se repita por quase todas as fases.

A grande questão surge exatamente aqui. O que de cada disciplina deve ser desenvolvido em cada fase do desenvolvimento. Embora cada projeto tenha suas necessidades específicas, o processo define um roteiro básico, que vale para a maioria dos projetos a serem desenvolvidos.

Os pontos de controle do projeto estabelecidos ao final de cada fase são assumidos aqui como momento para avaliar a fase terminada e verificar a possibilidade de se iniciar a nova fase. Conforme definido no RUP usamos os seguintes **pontos de controle maiores** para o processo:

- LCO (Life Cicle Objective) – no final da Concepção quando o escopo é conhecido e bem fundamentado;
- LCA (Life Cicle Architecture) – no final da Elaboração quando a arquitetura está completa e o conjunto de requisitos foi definido;
- IOC (Initial Operational Capability) – no final da Contrução e marca a primeira versão beta;

- PR (Product Release) – no final da Transição e do ciclo de desenvolvimento;

Uma análise de cada fase é apresentada nesta parte do trabalho, apresentando quais as atividades dentro de cada disciplina podem devem ser utilizadas segundo o objetivo de manter o processo resultante compatível com os ideais ágeis e com as regras ditas pelo SW-CMM.

É importante notar que serão apresentadas algumas opções de diagramas e abordagens na elaboração dos mesmos. Neste caso, será usado Agile Modeling, pois toda atividade de modelagem deve seguir as práticas propostas por esta abordagem. Os detalhamentos de quais diagramas devem ser desenvolvidos a cada momento orienta a equipe na *aplicação dos artefatos corretos*. Em toda proposta de modelagem desta abordagem vale propostas como *modelar com os outros*, *criar conteúdo simples*, *modelar para comunicar*, *modelar para entender*, *aplicar padrões de modelagem*, entre outros.

CONCEPÇÃO

O foco principal da Concepção está na aquisição de conhecimento sobre o problema do cliente, de maneira a se delinear a fronteira do sistema a ser desenvolvido. O objetivo ao final da fase é possuir informações suficientes sobre o negócio de forma a permitir verificar a viabilidade de se instaurar um projeto de software. É importante obter uma Visão inicial do sistema atual do cliente, seja este um conjunto de procedimentos manuais ou um sistema legado qualquer. Uma boa definição dos Riscos técnicos e não técnicos para se prosseguir o projeto também deve ser realizada, para permitir realizar um bom estudo de viabilidade. Tal definição de riscos também será usada nas próximas fases para se definir atividades que permitam atacar tais riscos e impedir que eles atrapalhem o andamento do projeto.

Durante a concepção, como as forças estão concentradas na análise do ambiente atual do usuário, as disciplinas mais utilizadas são as de Modelagem de Negócio e a de

Requisitos. A Modelagem de Negócio nos permite explorar, em detalhes, o funcionamento dos processos do cliente. A disciplina de Requisitos explora as necessidades dos usuários e visa estabelecer uma idéia inicial do que o sistema deve fazer.

Deve-se realizar uma seqüência de entrevistas com os usuários envolvidos nos processos de negócio que se deseja explorar, observações do dia de trabalho, levantamento e avaliação dos documentos trocados e análise dos sistemas automatizados existentes. Este levantamento pode ser registrado em papel manualmente e depois passado para arquivo. O importante é registrar o máximo de informação possível e depois documentar, na forma de tópicos os pontos identificados, tentando organizar alguma seqüência de como as coisas acontecem para facilitar o entendimento depois.

Também vale a pena explorar um pouco a tecnologia que será empregada, definir uma idéia inicial de estilo de arquitetura a se adaptar melhor para o projeto diante dos requisitos identificados. Para verificar se será possível esta ou aquela solução é aconselhável à utilização de spikes, como propõe o XP, que são soluções temporárias empregadas apenas para verificar melhor um dado problema. Também pode ser útil a definição de uma metáfora do sistema para facilitar comunicar as soluções imaginadas para o usuário e receber seu feedback de tais soluções mais facilmente.

A boa definição da Visão do sistema atual e a lista de possíveis Riscos do projeto ajudarão na elaboração de um Plano de Projeto que visa orientar como se darão os próximos passos do desenvolvimento. É importante que o Scrum Master seja o líder desta etapa e que as atividades desta fase tenham revisões periódicas pelo gerente sênior.

O diagrama de atividades da UML pode ser muito útil nesta fase para representar o mapeamento dos processos de negócios do usuário, mostrando como os diversos departamentos ou profissionais trocam informações entre si, aumentando a capacidade de entendimento do problema a ser resolvido. É também um ótimo veículo para validar

se o que foi entendido sobre o funcionamento atual do negócio corresponde efetivamente com a realidade, já que o próprio usuário pode avaliar o diagrama e ajudar a refiná-lo.

O ideal é poder realizar a Concepção dia após dia, sem intervalos, para que ela demore o menor tempo possível e também para que se evite perdas de informação. É importante tem reuniões freqüentes com os clientes para que não haja perda de informações, discussões antigas recomeçarem e assim por diante.

Dependendo do problema a ser entendido pode ser necessária uma Concepção maior (o RUP permite que esta fase leve meses mas este não é o objetivo desta abordagem), normalmente pequenos e médios projetos de software não precisam de uma fase de Concepção maior do que uma ou duas semanas. Lembre-se que a idéia é definir se é possível ou não seguir o projeto e obter uma idéia macro do escopo a ser tratado. Análise e Projeto têm mais força na etapa seguinte.

ELABORAÇÃO

Aqui começam as semelhanças fortes com um projeto XP e Scrum. A Elaboração tem início com as primeiras Sprint meetings, misturando um pouco com o Jogo do Planejamento de XP. A idéia é começar com a escrita das estórias que o usuário deseja contar sobre o sistema que será construído.

Da mesma forma que as estórias serão usadas como base para a escrita dos testes de aceitação do sistema, assim será feito com os casos de uso. Neste ponto do processo queremos apenas definir uma breve descrição do caso de uso que nos permita identificar um escopo mais detalhado do sistema e mensurar o tempo que será necessário para implementá-lo, exatamente como funcionam as estórias. Casos de uso com este nível de detalhes são chamados casos de uso não expandidos ou de alto-nível, isto é, sem os detalhamentos dos cenários que podem aparecer em especificações mais detalhadas.

Existe muita discussão sobre casos de uso, sobre o que deve conter suas especificações, qual o nível de detalhe, etc. Depende qual o "tamanho" cada um entende como ideal para um caso de uso. Aqui, será usado o mesmo princípio de estórias para definir casos de uso: um caso de uso representa uma necessidade ou objetivo do cliente, algo que ele deseja que o sistema faça para ele. Se um caso de uso for estimado maior do que o tamanho de uma iteração, pelo menos cada um de seus cenários deve ser menor, para que possam caber na mesma. Cenários maiores devem ser quebrados, o que pode implicar na reestruturação do próprio caso de uso como um todo.

O registro de casos de uso e suas descrições podem ser feitos em arquivo de computador, mas AM orienta para o uso das ferramentas mais simples possíveis. Neste caso, mais uma vez estórias e casos de uso se assemelham: cartões são a maneira mais simples de representá-los. Pode ser válido gastar tempo documentando os casos de uso em arquivos de computador, se isto implicar em alguma vantagem para o projeto. Por exemplo, a publicação destas descrições em uma página do projeto, mantida pela equipe. De qualquer maneira, ter em mente a regra proposta por AM, ajuda a manter o foco na essência da modelagem e não cometer o erro de desprender muito tempo com a aparência de documentos, formatação do conteúdo, etc[16].

Na maior parte do tempo, fazer com que o cliente deve fazer parte da equipe do projeto (*On-Site Customer*). Tal participação efetiva do cliente aumenta as possibilidades de sucesso já que garante um feedback rápido e melhora a comunicação, dois valores básicos deste processo.

Em XP o próprio cliente escreve as estórias. Se o tipo de usuário do projeto permitir esta abordagem então ela deve ser aplicada. Na verdade o processo deve criar estruturas para que os usuários possam ser treinados nesta tarefa e, com o tempo, possam desempenhá-la. Problemas culturais ou políticos costumam impedir a aplicação desta abordagem, portanto, nestes casos, a própria equipe deve escrever os casos de uso a partir das

informações extraídas do cliente. Mas deve-se garantir que os usuários concordem com estas definições e as reconheça como se eles mesmos as tivessem escrito.

Enquanto o foco da Concepção estava em tentar conhecer mais sobre o problema a ser tratado para verificar sua viabilidade, na Elaboração é desejado que os objetivos do novo sistema sejam, então, definidos. Entendam-se estes objetivos como as funções que o sistema deve executar, seu escopo real, seu conjunto de funcionalidades que agregam valor ao negócio do cliente. Estes objetivos são representados neste processo pelos casos de uso.

XP e os demais processos ágeis são contra a abordagem tradicional de longas sessões de análise e projeto antes do início da codificação (comumente chamado *Big Design Up Front*). Este processo está de acordo com tal abordagem, mas propõe uma solução um pouco diferente da proposta do XP. XP apregoa que o próximo passo logo após a definição das estórias é a construção. Sessões de projeto são feitos já em tempo de implementação para modelar as soluções que serão desenvolvidas em código. A proposta aqui é usar um pouco de Análise e Projeto para ajudar a definir um bom conjunto de casos de uso e evitar muita modificação do escopo no início da Construção. Deseja-se evitar o Big Design, característica principal dos processos monumentais. Mas seria útil definir uma série de modelagem suficiente antes de partir para o código.

A idéia central é definir uma primeira visão do modelo de classes conceitual, o que pode ser útil para um refinamento inicial da lista de casos de uso estabelecidos. Um diagrama de classe e um diagrama de casos de uso são as ferramentas usadas para expressar estas abstrações. Durante a Construção estes modelos são refinados e detalhados incrementando o nível de conhecimento de cada parte do projeto. Tal abordagem está de acordo com a prática de AM de *modelar em pequenos incrementos*.
Desenhar protótipos da interface gráfica e anexá-las aos casos de uso melhora a comunicação, pois permite que o cliente tenha uma visão mais clara de como o sistema deve parecer. Ajuda também a equipe de desenvolvimento a ter uma idéia melhor do

tempo que será necessário para implementar o caso de uso. Caso possa ser obtido algum ganho de tempo na Construção em prototipar diretamente em ferramentas como geradores de HTML ou mesmo nos ambientes de linguagens RAD, esta é uma boa opção. Mas *usar as ferramentas mais simples* deve ser a prioridade do processo conforme o padrão de AM.

A Elaboração termina com a conclusão do sprint e uma reunião no modelo Scrum é feita para os clientes definirem as prioridades das estórias e a equipe de desenvolvimento determinar o tempo gasto para realizá-los de acordo com a proposta de XP de tempo ideal de desenvolvimento. Isto é, define-se o tempo para que um caso de uso seja implementado considerando que o responsável por isto se dedicará única e exclusivamente a esta tarefa, sem interrupções durante o tempo de trabalho. A Elaboração termina com o planejamento da Construção, definindo qual o tamanho das iterações, se será necessário uma iteração de funcionalidade zero, quantas iterações serão necessárias, quais os casos de uso devem ser terminados para que o primeiro release possa ser entregue ao cliente e etc.

CONSTRUÇÃO

A Construção é o momento correto para aplicação de uma das práticas mais anunciadas como capaz de minimizar os possíveis desvios de cronograma e orçamento tão comum em nossos projetos: o desenvolvimento iterativo e incremental. Brooks já apresentava esta solução como um dos mais importantes ataques às tarefas essenciais do desenvolvimento. O RUP apresenta esta como uma de suas práticas. Processos iterativos são também a base dos processos ágeis de desenvolvimento.

Apesar de toda certeza na melhoria significativa que esta prática pode fornecer ao processo aplicado, a maioria absoluta dos projetos de software, ainda hoje se utilizam o modelo clássico de ciclo de vida, o chamado modelo em cascata. Diversos livros se preocupam em apresentar as vantagens e desvantagens destes modelos. Um bom resumo

pode ser encontrado em Agile Alliance, (http://www.agilealliance.org). Acreditando que o ciclo de vida iterativo e incremental é muito mais eficiente que os diversos modelos existentes, este processo também se baseia nesta técnica.

Embora o RUP permita que iterações sejam utilizadas em outras fases do ciclo de desenvolvimento, o único lugar onde esta estrutura de gerenciamento se torna obrigatório é na Construção. Alguns processos podem utilizar-se de uma fase de Elaboração mais robusta, ou seja, uma Elaboração que não apenas identifique os casos de uso e atribua a estes uma descrição resumida suficiente para uma estimativa inicial (conforme proposto pelo XP), mas onde os detalhes destes casos de uso sejam explorados em maiores detalhes, com especificações de cenário e etc. Mas esta abordagem pode conduzir a um *Big Design*, característica que se deseja evitar. Portanto, como se deseja uma Elaboração *"light"*, a melhor proposta é realmente deixar o desenvolvimento iterativo para a fase de Construção.

A Construção torna-se então, um conjunto de iterações de tempo pré-definido (*timeframe* da iteração), onde um conjunto de casos de uso é escolhido no início da iteração, explorado, implementado e testado ao longo da mesma e entregue como pronto ao seu final. Portanto, cada iteração começa com Planejamento da Iteração, uma continuação do Jogo do Planejamento de XP e do Plannig do Scrum, onde o cliente define quais casos de uso devem ser implementados na iteração. A equipe é dividida, definindo-se quem é responsável por qual caso de uso e a iteração tem então início.

O uso da Programação em Pares, como proposto pelo XP, é extremamente recomendado, contudo nem sempre é possível aplicá-lo. Novamente problemas culturais e políticos podem impedir sua aplicação e, mais uma vez, é aconselhável que algumas estruturas sejam criadas para permitir comprovar a eficiência da técnica, de forma que ela possa ser aplicada oficialmente em futuros projetos. Quando não for possível aplicar a Programação em Pares, recomenda-se o uso de uma bem conhecida prática de Engenharia de Software, a Revisão pelo Companheiro. O objetivo é tentar ganhar pelo

menos algum dos benefícios da programação em pares como conhecimento compartilhado por pelo menos duas pessoas, identificação de problemas que quem desenvolveu normalmente não encontra, entre outros.

De volta a estrutura da iteração é possível perceber facilmente que o grande foco da Construção está em atividades de Implementação e Testes. É claro que as disciplinas de Análise e Projeto também são fortemente empregadas aqui, principalmente por que os detalhes de cada caso de uso foram deixados para serem explorados durante as iterações da Construção. Dentro das iterações os casos de uso serão explorados e os mais complicados devem ser reescritos em sua forma expandida. Para estudar os cenários Diagramas de Seqüência e Colaboração da UML podem e devem ser usados, principalmente quando elaborados de uma maneira mais próxima possível de como a solução será implementada. A aplicação de padrões como os padrões de atribuição de responsabilidades, proposto em e os inúmeros padrões de projeto existentes (principalmente o conjunto conhecido como GoF – *Gang of Four*) [16] orientam as soluções empregadas para reconhecidos modelos de sucesso e garantem uma qualidade maior do código gerado que vai refletir em uma melhor manutenibilidade no futuro. Não se pode esquecer, entretanto, a prática de XP de Projeto Simples. Manter as soluções o mais simples possível é fundamental para o entendimento e para o crescimento do sistema à medida que se aprende mais sobre ele. As idéias de AM de *modelar em grupo*, *criar conteúdo simples*, *representar os modelos de maneira simples* e *usar as ferramentas mais simples* devem ser aplicadas aqui. Com relação as ferramentas usadas para registrar os modelos, na maioria dos casos, modelos em papel, em quadros ou flip-charts são muito mais úteis e eficientes. Caso a equipe possua alguma ferramenta CASE que possa produzir código derivado destes modelos valer a pena perder um tempo documentando as coisas na ferramenta para ganhar na frente com a geração de código. Outros diagramas úteis neste processo de modelagem são os Diagramas de Estado e os Diagramas de Componentes e Implantação da UML[16]. Os primeiros ajudam a representar detalhes dos estados das classes ao longo da execução dos casos de uso e os

últimos formam bons instrumentos para representação da arquitetura a ser empregada na implementação do sistema.

À medida que mais detalhes vãos sendo conhecidos sobre os casos de uso, refinamentos vão acontecendo nos modelos de classes e de casos de uso existentes. Algumas destas modificações atingem diretamente o cronograma estabelecido e implicam em replanejamento. Durante o desenrolar da iteração, deve ser medido o andamento das tarefas e, no final da mesma, esta medição deve conduzir a uma análise honesta e aberta de como o projeto se encontra. Se durante a iteração a velocidade da equipe aumentou, isto é, ela foi capaz de produzir mais casos de uso do que o previsto isto deve refletir na nova iteração que será iniciada. Se a equipe andar mais devagar o escopo deve ser reorganizado, possivelmente as prioridades deverão ser revistas e menos casos de uso (em termos do tamanho dos mesmos) serão escolhidos para a iteração seguinte. Isto tudo significa analisar o plano a todo o momento. Lembre-se que a única coisa certa em projeto de software é que ele vai mudar, portanto é necessário identificar tais mudanças e refleti-las o quanto antes no plano do projeto.

Durante a codificação dos casos de uso as práticas de XP são fortemente recomendadas. O uso de padrões de codificação é uma delas, o que permite uma melhor comunicação por parte da equipe já que fica mais claro compreender os programas que estão sendo escritos. Outra é a aplicação de técnicas de Refatoração para melhorar o código existente e torná-lo mais reutilizável. Uma boa estratégia é quando for preciso modificar um determinado código, aplicar as técnicas de refatoração antes das alterações para tornar o código melhor e, então, proceder às modificações. Isto evita o efeito cascata de se propagar código ruim pelo sistema, muito comum quando é dada manutenção em partes do código de baixa qualidade. O uso de refatoração com constância exige um mecanismo de teste bem eficiente. Os Testes de Unidade e os Testes de Aceitação de XP são as ferramentas ideais aqui, pela sua simplicidade e robustez. Criar testes automatizados baseados no framework xUnit é uma prática que deve ser experimentada e difundida pela equipe, pois os resultados que serão obtidos são realmente fantásticos.

Algumas outras práticas interessantes devem ser salientadas. O desenvolvimento iterativo e incremental conduz o projeto a produzir Pequenas Versões, uma estratégia muito útil, pois permite a introdução gradativa do sistema no ambiente do usuário, além de permitir que a equipe entregue software o mais cedo possível e receba um feedback mais rápido do mesmo. Este modelo de desenvolvimento ajuda a evitar situações comuns nos projetos de software que são as excessivas horas extras. Semanas de 40 horas, mais uma prática de XP, tem o objetivo de manter a equipe estimulada, dando às pessoas tempo para sua própria vida, já que elas são o grande fator de sucesso dos nossos projetos. Os desvios do cronograma serão conhecidos e necessidades extremas de horas extras podem ser minimizadas.

TRANSIÇÃO

O foco principal da Transição é distribuir o software no ambiente do cliente, ou seja, em produção. A disciplina mais valorizada nesta fase é a Implantação e, embora seja possível o aparecimento até mesmo de novos requisitos nesta fase, a aplicação das práticas propostas nas fases anteriores, principalmente o uso de iterações durante a Construção visam diminuir o máximo possível esta possibilidade.

Durante a implantação erros ainda podem ser encontrados, embora minimizados pelos testes feitos na Construção e pelo uso dos casos de uso implementados pelos usuários como forma de validação e difusão de conhecimento sobre o sistema, também proposto para a Construção. Para tratar estes erros, é necessária a implementação de correções que talvez impliquem em alguma atividade de modelagem adicional.

Pode ser preciso codificar rotinas de migração de dados dos sistemas antigos para o novo sistema. Talvez seja necessário que alguma interface seja implementada caso não tenha sido definida em fases anteriores.

A transição segue da versão beta do produto até sua implantação definitiva no universo do usuário. Passar para a Transição caracteriza que a ênfase do projeto agora e sua passagem para produção, mesmo que algumas coisas ainda estejam sendo implementadas.

Conclusão: Percebe-se que uma abordagem através do processo discutido (uma instância do RUP) é possível que o XP e Scrum seja aplicado para que o processo além de tornar-se ágil, torne-se maduro, ou seja, capaz de cumprir com todos as kpa's do CMM.

5 – ANÁLISE DOS RESULTADOS

Objetivo: o principal objetivo deste capítulo é fazer uma análise clara do que foi proposto e do que foi obtido por este trabalho, além e indicar a possibilidade de que as kpa's do CMM nível 3 também podem ser alcançadas a partir do mesmo estudo de caso, desde que claro cuidando muito mais do aspecto organizacional da empresa.

Como se pode perceber o XP apesar de ser um framework para o desenvolvimento de software não é capaz o suficiente, quando aplicado isoladamente de alcançar todas as metas impostas pelo CMM nível 2. O mesmo tipo de pensamento seria válido para o Scrum.

O CMM Integrated (CMMI) poderia ser usado no lugar do CMM, já que o CMMi tem uma representação contínua que permitiria uma organização ter acessos a somente algumas áreas ao invés de ter todas as áreas cobertas pela representação

É importante citar também que entre outras coisas as metodologias ágeis ajudam a aumentar a produtividade da equipe de projeto. Neste caso específico, Scrum e XP são metodologias complementares, mas o mais importante é que ambas metodologias têm os seus focos principais na entrega de artefatos que realmente adicionem valor ao negócio do cliente além é claro de ter uma equipe unificada e disciplinada. Apesar de as metodologias ágeis pregarem a não criação de muita documentação, isso não quer dizer que ambas metodologias sejam rígidas e bem controladas.

Conforme dito anteriormente, XP , Scrum e CMM são claramente complementares. Embora este estudo tenha compreendido somente o nível 2 do CMM é interessante notar que várias kpa's dos níves 3, 4, 5 poderiam ser avaliadas usando esta abordagem através de melhoras no nível organizacional da empresa .

É importante lembrar que nem todos as metas do CMM nível 2 são alcançados única e exclusivamente com o XP e Scrum. É preciso lembrar que no caso de uso uma instância do RUP foi usada, e que, portanto metas como um grupo independente de qualidade de software e métricas do mesmo seriam implementadas por este grupo.

Uma outra medida extremamente importante seria a criação de um SEPG (Software Engineering Process Group) que seria importante para a organização no sentido em que eles seriam responsáveis pelo acompanhamento do processo e de seu relacionamento com o CMM.

De acordo com a SEI, o CMM requere um compromisso de longo termo e isso tomaria quase 2 anos ou mais para atingir o segundo nível. De acordo com esta monografia, o trabalho já foi feito usando o que foi dito anteriormente . Seria isto um paradoxo?

Um próximo passo ou a continuação deste estudo seria a criação de uma nova abordagem, que traria o melhor das metodologias ágeis (XP, Scrum, AM, DSDM, Crystal, etc) com o intuito alcançar os 5 níveis do CMM. Neste caso, seria extremamente interessante e desafiador não usar como base nenhum processo de software já existente (neste caso foi usado o RUP) para que todo o modelo/processo seja de origem ágil e que tenha como meta entregar software de qualidade e de acordo com as regras exigidas pelo SW-CMM.

6 - REFERÊNCIAS BIBLIOGRÁFICAS

[1] Kent Beck, *Extreme Programming Explained* , ISBN 0-201-61641-6, Addison-Wesley, 2000.

[2] Mark C. Paulk, *Extreme Programming from a CMM Perspective*, http://www.sei.cmu.edu/cmm/papers/xp-cmm-paper.pdf, 2001.

[3] Kenneth M. Dymond , *A Guide to the CMM* , ISBN 0-9646008-0-3 , Process Transition , 1995.

[4] Schwaber e Beedle, *Agile Software Development with Scrum*, ISBN: 0-13-067634-9, Prentice Hall, 2001.

[5] Yahoo! Grupo de discussão para XP, http://groups.yahoo.com/group/extremeprogramming/

[6] Ron Jeffries, *Extreme Programming and the Capability Maturity Model*, http://www.xprogramming.com/xpmag/xp_and_cmm.htm, 2000.

[7] Kent Beck, Martin Fowler, *Planning Extreme Programming*, ISBN: 0201710919, Addison-Wesley Pub Co, 2000.

[8] Scrum , www.controlchaos.com

[9] Richard HighTower e Nicholas Lesieck , *Java Tools for eXtreme Programming* , ISBN: 047120708-X , John Wiley & Sons, 2002.

[10] Agile Alliance, http://www.agilealliance.org

[11] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, *Design Patterns*, ISBN: 0-20-163361-2, Addison-Wesley Pub Co, 1995.

[12] Martin Fowler, *Refactoring*, ISBN: 0-20-148567-2, Addison-Wesley Pub Co, 1999.

[13] Ron Jeffries, Ann Anderson, Chet Hendrickson, Kent Beck, *Extreme Programming Installed*, ISBN: 0201708426, Addison-Wesley Pub Co, 2000.

[14] Kane Mar , *Scrum with XP.*

[15] Seth Bowen , *An Assessment of Scrum with the SW-CMM , Department of Computer Science University of Calgary.*

[16] Jefferson de Barros Santos , *Extraindo o Melhor de XP, Agile Modeling e RUP para melhor produzir software* , PUC-Rio.