UNIVERSIDADE ESTADUAL DO SUDOESTE DA BAHIA

Colegiado do curso de Ciência da Computação

LEONARDO RODRIGUES RIBEIRO

UM PROCESSO DE DESENVOLVIMENTO DE SOFTWARE: ESTUDO DE CASO EMPRESA CREATIVER

Vitória da Conquista – BA
Fevereiro 2007

LEONARDO RODRIGUES RIBEIRO

UM PROCESSO DE DESENVOLVIMENTO DE SOFTWARE: ESTUDO DE CASO EMPRESA CREATIVER

Monografia de conclusão de curso apresentada a Universidade Estadual do Sudoeste da Bahia – UESB - para obtenção do título de Bacharel em Ciência da Computação.

Área de Concentração: Engenharia de Software
Orientadora: Maísa Soares dos Santos Lopez

Vitória da Conquista – BA
Fevereiro 2007

Dedico este trabalho a minha família que sempre me apoiou e acreditou na minha capacidade de um dia vencer. Meu muito obrigado a todos vocês por ter proporcionado esta oportunidade.

AGRADECIMENTOS

Primeiramente a Deus por ter me proporcionado as oportunidades que sempre pedi.

A minha orientadora Maísa Soares por sempre indicar os caminhos corretos e me incentivar para a finalização deste trabalho.

Aos meus colegas de curso e grandes amigos: Eduardo Rodrigues (Duda), Cláudio Rodolfo de Souza (Craudim), Dario Portugal, Edmilson Santos (PM) e Marcos Pereira (Mestre) que incentivaram para vencer mais essa etapa da minha vida.

Aos meus professores e mestres que contribuíram para a minha formação.

“Tente, e não diga que a vitória esta perdida”.

(Raul Seixas)

RESUMO

Mesmo com a evolução dos computadores, das técnicas e ferramentas nos últimos anos, a produção de software confiável, correto e entregue dentro dos prazos e custos estipulados ainda é muito difícil. Para tentar sanar essas dificuldades, técnicas e métodos para especificar, gerenciar e desenvolver sistemas foram propostos ao longo do tempo. Algumas se concentram no formalismo e planejamento enquanto outras buscam formas mais ágeis de implementação. Com o auxílio desses métodos as empresas passaram a desenvolver sistemas de forma a garantir a qualidade e o baixo custo de seus produtos.

Este trabalho visa à adaptação e adequação de um processo de desenvolvimento de software para a empresa Creativer – Serviços e Saúde. Empresa esta que trabalha com consultoria na área de saúde. Este processo se concentra nas principais praticas das metodologias RUP e XP, e procura garantir uma sistematização das atividades de desenvolvimento aperfeiçoando assim a definição de requisitos, a distribuição das tarefas e maximizando a qualidade do produto final. O trabalho exibe ainda, uma análise de como é o desenvolvimento dos projetos da empresa e sugere um novo processo de desenvolvimento para substituir o processo utilizado atualmente.

Palavras-chave:

Processo
Metodologias tradicionais
Metodologias Ágeis
Modelos
RUP
XP

ABSTRACT

Even with the evolution of the computers, of the techniques and tools in the last years, the production of software reliable, correct and give inside of the periods and costs stipulated is still very difficult. To try to cure those difficulties, techniques and methods to specify, to manage and to develop systems was proposed along the time. Some concentrate on the formalism and planning while another look for forms more agile of implementation. With I aid him of those methods the companies passed to develop form systems to guarantee the quality and the low cost of their products.

This work aims at to the adaptation and adequacy of a process of development of software for the Creativer company - Services and Health. Company this that works with consultoria in the health area. This process if concentrates in the main ones practises of methodologies RUP and XP, and looks for to guarantee a systematization of the activities of development thus perfecting the definition of requirements, the distribution of the tasks and maximizing the final product quality. The work still shows, an analysis of as it is the development of the projects of the company and suggests a new process of development to substitute the process used currently.

Key word:

Process
Traditional Methodologies
Agile methodologies
Models
RUP
XP

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

1. INTRODUÇÃO

Quando o matemático inglês Babbage terminou a construção de uma máquina capaz de efetuar cálculos, envolvendo diferenças finitas, ele nem imaginava o que estava por vir. Impulsos elétricos passaram a transmitir informações através dos telégrafos. Vieram as válvulas e a primeira geração de computadores. Gigantes como o Eniac faziam cálculos cada vez mais rápido e a simplicidade dos bits (0 e 1) executavam tarefas com resultados cada vez mais surpreendentes para a época.

Na década de 50, surgiram os computadores da segunda geração. Os transistores substituíram as válvulas e os cálculos ficaram cada vez mais rápidos e poderosos. Não demorou muito e os transistores foram miniaturizados em pequenos circuitos integrados, nascia a terceira geração. Os programas foram evoluindo a cada dia e as máquinas tornavam-se cada vez menores. Surgiram os computadores pessoais e juntamente com eles vieram redes de computadores e a necessidade de automatizar processos.

Na década de 80, com o surgimento da programação orientada a objetos e de linguagens visuais, que possibilitam o desenvolvimento de interfaces gráficas (Graphical User Interface - GUI), a programação foi se difundindo pelo mundo e muitas pessoas passaram a construir seus próprios programas e softwares para outras pessoas e empresas. Para isso, bastava conhecer um pouco das idéias básicas de programação (entrada e saída de comandos, repetições e laços, comandos de decisão).

Segundo Pressman (2006), o software se torna a força motriz por traz da revolução do computador pessoal e se encontra embutido em sistemas de toda a espécie: transporte, médico, telecomunicações, militar, industrial, entretenimento, etc. Entretanto, ninguém poderia prever que milhões de programas de computadores tivessem de ser corrigidos, adaptados e aperfeiçoados e que essa atividade de manutenção absorveria mais pessoas e mais recursos que todo o trabalho aplicado na criação de novos softwares.

Dessa forma, a busca pela qualidade no desenvolvimento de sistemas se se tornou uma preocupação mundial. O desenvolvimento de software não era mais uma atividade personalizada, uma arte que dependia exclusivamente do profissional que o concebeu. O programador solitário teve que dar lugar a uma equipe de especialistas em software, cada um se concentrando numa parte da tecnologia necessária para produzir uma aplicação complexa.

As empresas e instituições passaram a se informatizar e aperfeiçoar utilizando técnicas e métodos para especificar, gerenciar e desenvolver sistemas de forma a garantir a adesão entre a especificação de requisitos e o produto acabado, além de medir a qualidade de seus softwares de acordo com os critérios e padrões unificados, diminuindo assim seu custo de desenvolvimento de sistemas e aumentando a qualidade de seus produtos.

Uma das áreas que tem se preocupado com a informatização é a Saúde. De acordo com Sabbatine (1998), o divisor de águas da Informática em Saúde nacional ocorreu em 1986 quando o Ministério da Saúde reconheceu o grau de desenvolvimento nacional na área a partir de um seminário realizado em Informática na Saúde em Brasília. Os pesquisadores presentes resolveram então se organizar e fundaram em novembro de 1986, durante o I Congresso Brasileiro de Informática em Saúde, a Sociedade Brasileira de Informática em Saúde.

De maneira que a informática possa ser aplicada à área médica de diversas formas: desde sistemas para informatizar clínicas e hospitais, até o desenvolvimento de complexos sistemas de tele diagnóstico, estas aplicações variam de acordo com as necessidades de cada profissional. Sabbatine (1998) as define da seguinte maneira:

- Diagnóstico por imagens: Encontramos nesta parte uma série de aplicações destinadas ao manejo de imagens radiológicas, ecografias, anatomia patológica e outras de relevância clínica e sua integração com outras aplicações médicas, desde que se faça o intercâmbio das aplicações através de uma rede de computadores.

- Exames de Laboratórios: Neste campo se desenvolvem sistemas que permitem a integração de equipes na informação de resultados de exames, automação dos processos, controle de qualidade, etc., bem como sua comunicação com outros sistemas de informação em saúde.
- Monitorização de Pacientes: Voltado para o registro de informações do paciente referente à evolução diária de seus sinais vitais, registros na enfermaria e monitorização eletrônica (Pressão Arterial, Eletrocardiograma, Ventilação mecânica, oximetria, infusões, etc.).
- Administração Hospitalar: Os modelos atuais de Administração Hospitalar no Brasil são destinados exclusivamente para a área de Faturamento. Entretanto, o que se pretende é dispor de um sistema de dados a serem preenchidos ao lado do leito do paciente, tornando-se efetivamente um prontuário eletrônico, facilitando o acompanhamento da evolução do mesmo desde sua entrada no serviço de emergência até a alta hospitalar.

Porém, muitas outras aplicações, mais complexas, envolvendo a utilização de redes de computadores e principalmente a Internet, vêm sendo desenvolvidas e utilizadas por profissionais e por grandes centros de saúde em todo o mundo.

A Creativer - Serviços de Saúde é uma empresa que trabalha com consultoria na área de saúde e que busca a cada dia soluções que auxiliem e dêem suporte aos profissionais que trabalham nessa área. Vários programas foram desenvolvidos para auxiliar na gestão do fluxo das atividades do SUS (Sistema Único de Saúde). São programas que controlam desde a parte de administração hospitalar até a marcação e regulação de consultas, passando pelas diversas áreas da saúde pública.

Atualmente, a empresa desenvolve seus programas sem seguir nenhum processo de desenvolvimento. Os requisitos são identificados e logo implementados pelos desenvolvedores, quase nenhuma documentação é gerada.

1.1 Objetivos

1.1.1 Objetivo Geral

O objetivo dessa monografia é definir um processo de desenvolvimento de sistemas para sistematizar e organizar a produção de software da Creativer.

Pretende-se com este trabalho reunir um conjunto de especificações que irá auxiliar no desenvolvimento, evitando problemas como:

- Requisitos mal definidos e que mudam constantemente;
- Dificuldade de comunicação entre as pessoas envolvidas no processo;
- Falta de métrica no desenvolvimento;
- Falta de documentação e
- Alto custo com manutenção.

1.1.2 Objetivos Específicos

1. Trabalhar em cima de quatro valores básicos: Custo, Tempo, Qualidade e Escopo.
2. Definir o ciclo de vida do processo;
3. Definir os personagens envolvidos no processo (Gerente, analista, programador, testador e usuário);
4. Definir a documentação necessária gerada no decorrer do desenvolvimento do produto.

Com isso pretende-se reduzir o tempo e o custo do projeto, aumentar a qualidade do produto final e a satisfação do cliente, e mostrar às pequenas empresas os benefícios que a engenharia de software pode proporcionar ao crescimento das mesmas.

1.2 Justificativa

A cada dia a sociedade depende cada vez mais de softwares que lhe proporcione comodidade e facilidade. Aliado a essa dependência, ela exige uma

maior qualidade desse produto e desenvolvimento, e uma distribuição em um curto espaço de tempo.

Desenvolver sistemas funcionais com qualidade e que cumpram com os requisitos, prazos e custos definidos no planejamento, é possível a partir de processos com métodos e técnicas bem definidas (PETERS, 2001). Alguns desses processos apresentam métodos mais formais de desenvolvimento, enquanto outros se destacam pela forma ágil como gerenciam o projeto.

Segundo Astels (2002), projetos de desenvolvimento de software executados sem um processo estabelecido correm o risco de não serem bem sucedidos. Tentando evitar esses riscos, um processo de desenvolvimento de software será proposto para a empresa Creativer, haja vista que nenhum processo de desenvolvimento é utilizado e varias das etapas da Engenharia de Software são ignoradas, acarretando numa má qualidade de desenvolvimento do produto final.

Através deste processo será possível:

- Sistematizar as atividades do desenvolvimento;
- Aumentar a qualidade dos produtos;
- Evitar deslizes no cronograma;
- Reduzir custos com manutenção.

Diante destas possibilidades citadas acima e da necessidade de colocar em prática o conhecimento adquirido nestes longos anos de aprendizado no curso de Bacharelado em Ciência da Computação, oferecido pela Universidade Estadual do Sudoeste da Bahia – UESB, este estudo torna-se justificável.

1.3 Organização da Monografia

Esta dissertação possui a seguinte organização:

- O capítulo 2 descreve em linhas gerais as Metodologias Tradicionais de desenvolvimento, especialmente a metodologia RUP.

- O capítulo 3 por sua vez aborda as Metodologias Ágeis de desenvolvimento, focando na XP. Relata ainda trabalhos relacionados ao tema proposto.
- O capítulo 4 traz um diagnóstico da empresa apresentando a forma como são desenvolvidos os seus projetos, descrevendo o processo CREATIVO, suas práticas e a forma como o mesmo será adotado.
- A conclusão e trabalhos futuros são apresentados no capítulo 5.

2. METODOLOGIAS TRADICIONAIS

A crise do software culminou em 1969, quando os métodos de desenvolvimento de software existentes se tornaram obsoletos para o desenvolvimento de software de grande porte. Os programas não funcionavam adequadamente, havia atrasos nos projetos e o custo dos mesmos era sempre maior do que o que se havia previsto.

Em conseqüência disso surgiu a necessidade de se ver o processo de desenvolvimento de software como uma engenharia, e uma nova terminologia foi logo criada, o termo “Engenharia de Software”, para identificar “o estabelecimento e uso de sólidos princípios de engenharia a fim de obter economicamente software confiável e que trabalhe de forma eficiente em máquinas reais” (PRESSMAN, 2006).

De lá para cá, os sistemas foram ficando cada vez mais complexos e precisando de menos tempo para estarem prontos, sem perder qualidade. Novas técnicas e modelos precisaram ser criados para gerenciar, requisitos cada vez mais dinâmicos, falta de documentação, incompatibilidade de módulos, baixa qualidade de software, alto custo com manutenção, etc.

Para acompanhar esse ritmo de mudanças, vários modelos de processo de software, cada um deles com suas características e formas de gerenciamento, foram criados.

> “No desenvolvimento de software, o engenheiro se envolve em uma seqüência de atividades que produzem uma variedade de documentos, culminando em um programa satisfatório e executável. Essa atividades de engenharia englobam aquilo que conhecemos por processo de software (uma seqüência de etapas com feedback que resultam na produção e na evolução do software)” – (PETERS, 2001, p.21)

Essa seqüência de etapas é obtida de forma implícita com esquemas e atividades integradas consideradas vitais para o desenvolvimento bem sucedido dos sistemas de software. Seguem abaixo alguns desses modelos.

2.1 Modelos de Desenvolvimento de Software

Todo o software tem um ciclo de vida, ou seja, um período de tempo que começa com a seleção de um modelo de ciclo de vida como por exemplo: Cascata, Incremental, espiral, prototipação, evolucionário, orientado a reuso e termina quando o software deixa de estar disponível para a utilização.

Pelo padrão IEEE 1074-1995, há três processos principais identificados durante a fase de desenvolvimento de software:

- Requisitos – Decidir o que o sistema deve fazer, suas atividades, riscos e planos de teste.
- Projeto – Determinar como um sistema efetua cálculos, sua estrutura e suas funções específicas.
- Implementação – Produzir código-fonte, documentação e testes; validar e verificar.

“Com a experiência adquirida pelos projetistas após uma variedade de projetos esses três requisitos básicos foram virando modelo” (PETERS, 2001).

2.1.1 Modelo Cascata

Proposto por Royce em 1970, foi o primeiro modelo a ser criado. Ele descreve uma seqüência de atividades que começa com a exploração de conceitos para a resolução de um problema e conclui com a manutenção e uma inevitável substituição (PETERS, 2001). Passando por fases de análise de requisitos, projeto, implementação, teste e integração, como mostra a Figura 1.

Cada fase apresenta um feedback com a fase anterior e a mesma inicia-se somente quando a anterior termina. Segundo Peters (2001), a melhor forma de garantir a evolução do software é o feedback, a sua presença nas diversas fases do processo garante a melhoria contínua do produto final e o desenvolvimento de novas versões.

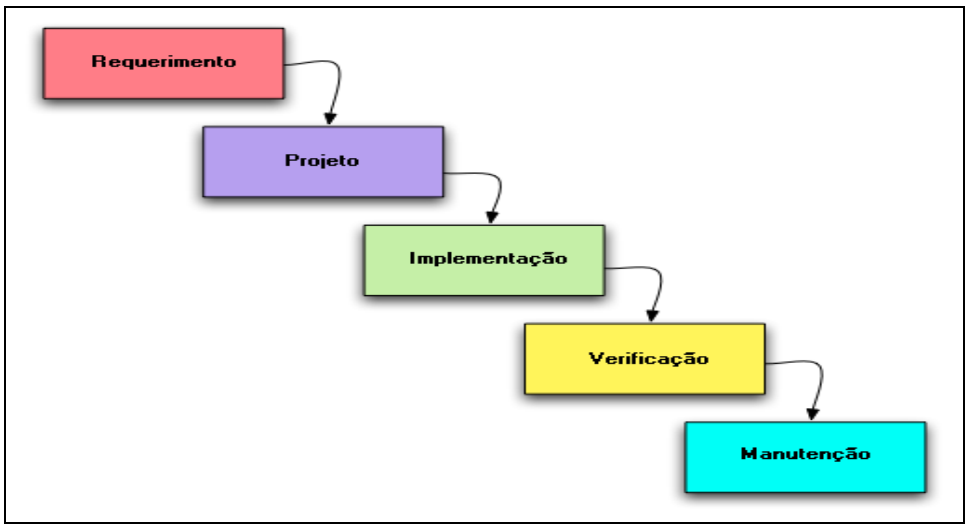

Figura 1 *– Modelo Cascata (Wikipédia, 2007).*

De acordo com Peters (2001), esse feedback estimula a melhoria e a evolução do software. Nota-se, portanto, no modelo cascata um desenvolvimento progressivo do produto de software, que tem como vantagem a produção de documentação fixa a cada fase do ciclo de vida. Porém, suas desvantagens são: o cliente ter que esperar até a fase de instalação para ver o funcionamento do sistema, e a falta de noções de prototipação rápida e de desenvolvimento incremental.

2.1.2 Modelo Incremental

Definido em 1991, pela European Space Agency, o modelo incremental veio em seguida. Aparentemente ele é bem semelhante ao modelo em cascata mas apresenta a flexibilidade de que os requisitos podem mudar (PETERS, 2001). Por isso, o software é desenvolvido em pequenas partes (versões) que vão se agrupando no decorrer do processo e formam o produto final, Figura 2.

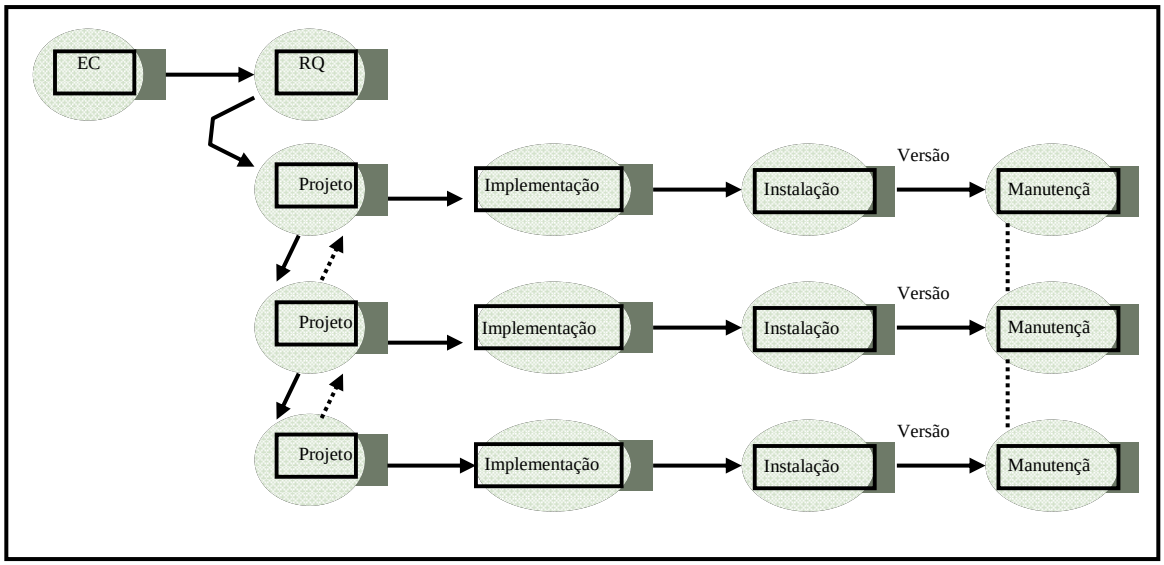

***Figura** 2 – Modelo Incremental (Adaptado de PETERS, 2001).*

De acordo com Peters (2001), as vantagens do modelo incremental são: produção de um produto operacional a cada incremento; avaliação do cliente a cada versão; existência de um risco menor de fracasso completo do sistema.

Porém, suas desvantagens são segundo Peters (2001): pode ser difícil mapear os requisitos dos clientes dentro de incrementos de tamanho correto; pode ser difícil estabelecer prazos e custos; e quando parar, já que sempre se pode ter uma nova versão.

2.1.3 Modelo Espiral

Foi apresentado por Boehm em 1986 e combina as características do modelo cascata e incremental. Ele parte do princípio de que a forma de desenvolvimento de software não pode ser completamente determinada de antemão (PETERS, 2001), Figura 3.

Acrescenta ainda aspectos gerenciais ao processo de desenvolvimento de software, como análise de riscos em intervalos regulares do processo de desenvolvimento de software, planejamento, controle e tomada de decisão.

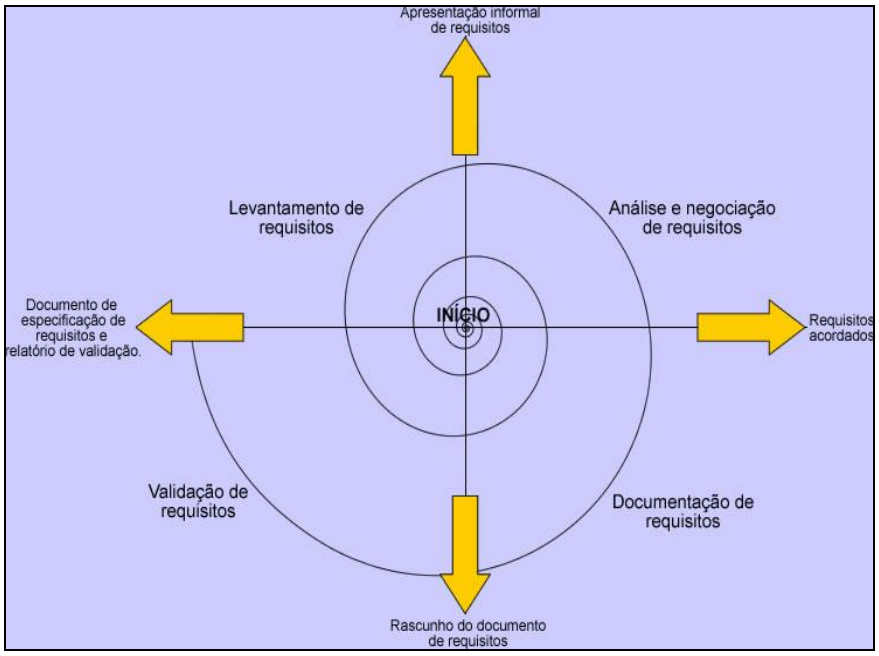

Figura 3 *– Modelo Espiral (Wikipédia, 2007).*

Suas quatro fases prevêem:

- Planejamento: determinar objetivos, alternativas, e restrições.
- Análise de Riscos: análise das alternativas e identificação e resolução dos riscos.
- Projeto ou Engenharia: desenvolvimento e implementação da próxima versão.
- Avaliação dos resultados pelo cliente.

Estas quatro fases são repetidas várias vezes até que o produto final esteja de acordo com o que foi previsto pelo usuário.

2.1.4 Modelo de Prototipação

Tem como foco principal o rápido desenvolvimento de produtos de software que permitam ao usuário experimentar partes do sistema que está por vir, figura 4. É importante ressaltar que esses protótipos têm funcionalidades e desempenho limitados, e geralmente se concentram nos requisitos de mais alto risco (PETERS, 2001).

A construção de protótipos com os quais os usuários possam brincar é uma idéia bastante atrativa para sistemas grandes e complicados, e quando não existir um sistema anterior ou um sistema manual que o ajude a especificar os requerimentos.

Figura 4 *– Modelo de Prototipação (Adaptado de Sommerville, 2003).*

As vantagens da prototipação são: melhoria na qualidade da especificação de requisitos dos futuros programas, o que leva à diminuição dos gastos com manutenção; em alguns casos, o treinamento dos usuários pode até ser feito antes do produto ficar pronto; e algumas partes do protótipo podem vir a ser usadas no desenvolvimento do sistema final.

As desvantagens são: atraso no início da implementação do sistema final devido a construção do protótipo; a exigência do cliente de que pequenos acertos sejam feitos para que o protótipo se transforme no sistema final; utilização de linguagens, ferramentas e algoritmos inadequados e/ou ineficientes usados na implementação rápida do protótipo que acabam fazendo parte do sistema final.

2.1.5 Modelo Evolucionário

Impõe uma sobreposição contínua de atividades de desenvolvimento, o que produz uma sucessão de versões (evolução) do software.

De acordo com Peters (2001), a idéia base é desenvolver uma implementação inicial, expor o resultado para o usuário e fazer seu aprimoramento por meio de versões até que um sistema adequado tenha sido desenvolvido, Figura 5.

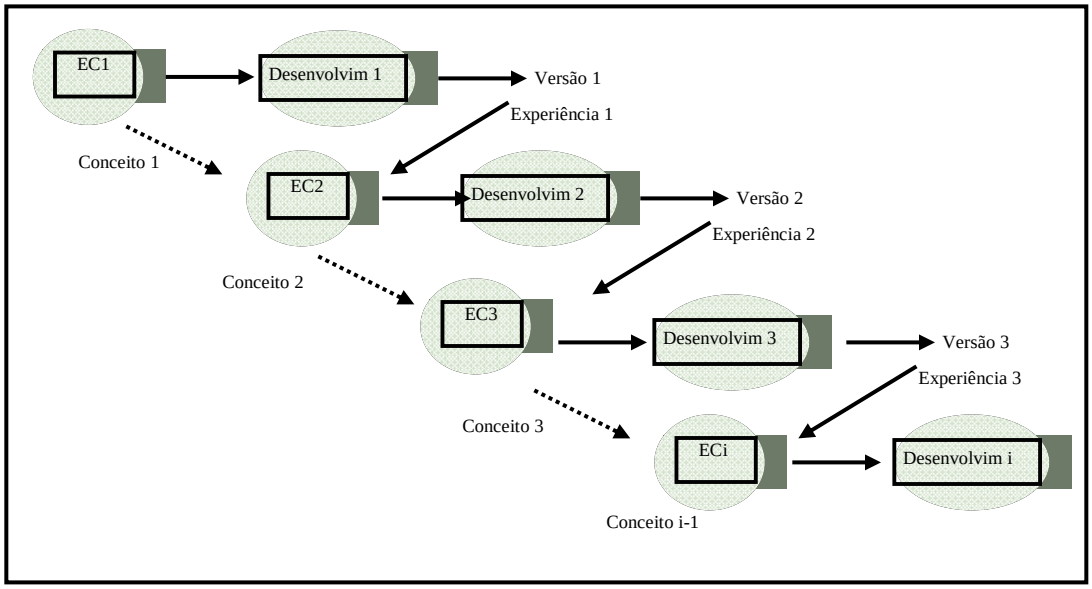

Figura 5 *– Modelo Evolucionário (Adaptado de Sommerville, 2003).*

Muitas vezes, esta abordagem é mais eficiente do que a abordagem em cascata, pois produz um sistema que atende às necessidades imediatas dos clientes. Entretanto, pode se tornar dispendiosa em decorrência da criação de várias versões.

2.1.6 Modelo Orientado a Reuso

Surgiu a partir da década de 80 e vem sendo amplamente utilizado nos últimos anos. Ele possibilita a utilização de partes de software já consolidados, em novos desenvolvimentos através do reuso de funções, componentes e até sistemas já prontos especialmente os chamados produtos de prateleira COTS - *Commerce on the shelf* (PETERS, 2001), Figura 6.

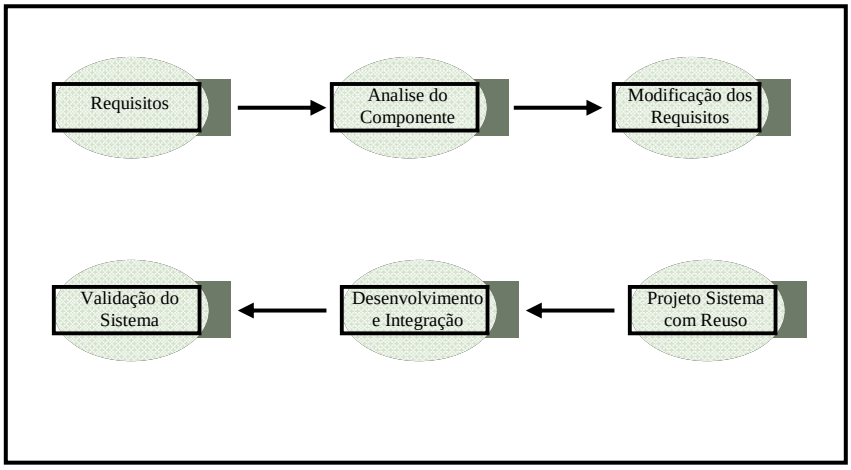

Figura 6 *– Modelo Orientado a Reuso (Adaptado de Sommerville, 2003)*.

Dentro das vantagens da orientação a reuso temos: redução de custos e aumento da produtividade, já que muita coisa pode ser reaproveitada; maior confiabilidade e redução de riscos, pois as partes reutilizáveis já foram testadas; conformidade de padrões e desenvolvimento acelerado.

Entre as desvantagens temos: aumento nos custos de manutenção visto que o código-fonte do componente reutilizado pode não estar disponível; pode ser muito difícil e dispendioso manter uma biblioteca com componentes reutilizáveis; muitos desenvolvedores preferem reescrever os componentes e funções; e a necessidade de fazer adaptações em funções e componentes.

2.2 RUP

Rational Unified Process (RUP) foi um processo desenvolvido em 1998, pela Rational em conjunto com a IBM, que procura estabelecer de forma disciplinada, tarefas e responsabilidades em uma empresa de desenvolvimento de software, visando a construção de sistemas de alta qualidade (MARTINS, 2006).

Ele originou-se de métodos anteriores propostos por Booch, Jacobson e Rumbaugh. Segundo esses autores o RUP é uma metodologia completa,

particularmente aplicável a grandes equipes de desenvolvimento de software trabalhando em projetos maiores.

Visando atacar problemas de tempo, mudanças de requisitos e mudanças de tecnologias, o RUP adota as seguintes premissas básicas:

- Uso de iterações para evitar o impacto de mudanças no projeto,
- Gerenciamento de mudanças,
- Abordagens dos pontos de maior risco o mais cedo possível.

Como o processo é centrado na arquitetura, pensa-se primeiro nos elementos de desempenho, escalabilidade, reutilização, restrições econômicas e tecnológicas, começando pela visão do software. Essa visão vai determinar o futuro do projeto e caso ele seja viável, será implementado.

Os casos de uso são utilizados como o principal artefato para o estabelecimento do comportamento desejado do sistema. “A partir deles são definidos os elementos da arquitetura, as rotinas de teste e a estratégia de implantação do sistema” (MARTINS, 2006). Com eles torna-se possível ainda a padronização na documentação dos requisitos do sistema.

Sua característica incremental envolve a integração contínua da arquitetura do sistema para a produção das versões, de maneira que cada versão incorpore os aprimoramentos incrementais em relação às demais.

Já as características iterativas, envolvem o gerenciamento de seqüências de versões executáveis. O ciclo completo de desenvolvimento resulta em uma versão de um produto executável que constitui um subconjunto do produto final. À medida que vai se juntando o resultado de cada iteração para outra, o sistema final vai tomando corpo e volume.

O RUP possui duas dimensões, conforme mostra a Figura 7. De acordo com Kruchten (2003), a primeira refere-se ao eixo horizontal, que representa o tempo e exibe os aspectos dinâmicos do ciclo de vida do processo caracterizados por ciclos, fases, iterações e pontos de controle. A segunda compreende o eixo vertical e representa o fluxo do processo, os aspectos estáticos do processo, e é descrito através de atividades, disciplinas, artefatos e regras.

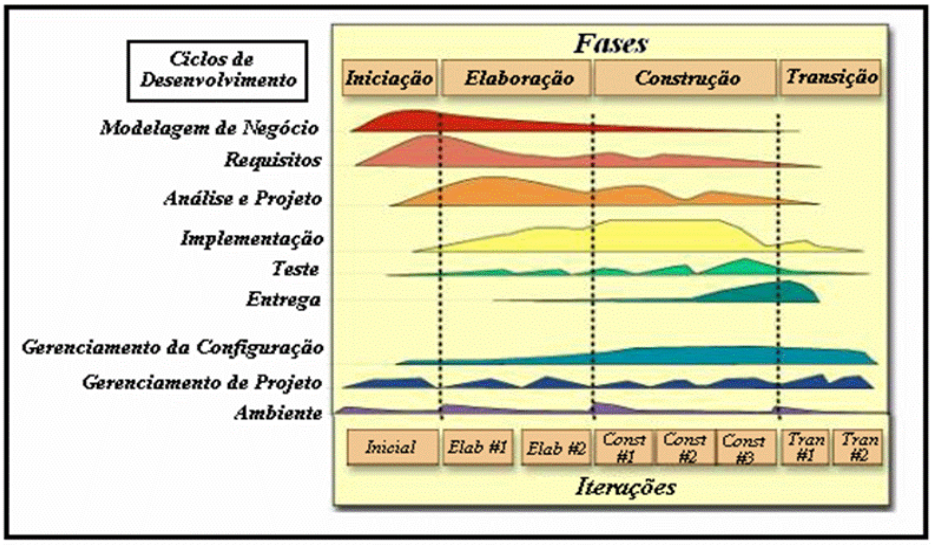

Figura 7 *– Modelo RUP (Kruchten, 2003).*

Ele possui um conjunto de quatro fases, conforme mostra o eixo horizontal. Estas fases existem em todos os projetos e cada uma termina com um marco relevante para o projeto (KRUCHTEN, 2003):

1. Iniciação – Define o escopo do projeto, o que é incluído e o que não é incluído. Nesse momento são identificadas todas a entidades externas (atores) com as quais o sistema irá interagir. Isto envolve a identificação de todos os casos de uso e uma breve descrição dos mais significativos. É definido um plano de negócios, que inclui critérios de sucesso, avaliação de risco, estimativa de recursos necessários ao desenvolvimento do projeto e um plano de fases indicando as datas dos principais pontos de controle.

2. Elaboração – A elaboração é focada na compreensão dos requisitos do sistema e na definição de uma arquitetura base. Nesse momento é possível , segundo Kruchten (2003), chegar a 80% dos casos de uso detalhados. Nesta fase, um protótipo de arquitetura executável é construído em uma ou mais iterações, dependendo do escopo, tamanho e risco do projeto. Demais resultados desta fase: Descrição da arquitetura de software, Lista de Riscos e Plano de Negócios revisados.

3. Construção – Durante esta fase, um produto completo é desenvolvido de forma iterativa e incremental, até gerar uma versão beta. Isto implica em descrever os requisitos remanescentes e os critérios de aceitação, completando a implementação e o teste de software. Demais resultados desta fase: Produto de Software Integrado às plataformas adequadas; Manuais de Usuário; Descrição da Versão Corrente.

4. Implantação – Implantação do sistema para o usuário final focado em treinamento, instalação e suporte. Uma vez que o produto tenha sido entregue ao usuário final, freqüentemente surgem questões que requerem o desenvolvimento de novas versões, correção de alguns problemas ou a finalização de características que foram postergadas.

O RUP possui também seis processos de engenharia e três processos de suporte, o eixo vertical da Figura 7.

Os processos de engenharia são:

- Modelagem de Negócio: documenta os conceitos pertencentes ao negócio. Esse documento mostra quem são os participantes do negócio, os *workflows*, as pessoas e suas funções, e que sistemas essas pessoas utilizam junto ao processo que elas seguem (MARTINS, 2006). As pessoas que trabalham neste processo são: o Analista de processo de negócio e o projetista de negócio. Os artefatos produzidos nessa etapa são: documento de visão, modelo de caso se uso, modelo de objetos do negócio, regras do negócio e o glossário do negócio.
- Requisitos: representa as características funcionais e não-funcionais do sistema. O objetivo deste processo é, segundo Martins (2006), definir as características do sistema conforme as observações do cliente. O resultado dessas observações será o documento de visão do sistema. As pessoas envolvidas neste processo serão: o Analista de Sistemas , o Especificador de Casos de Uso e o Projetista de Interface de Usuário. E os principais artefatos produzidos são:

documento de visão, modelo de caso de uso e protótipo da interface com o usuário (KRUCHTEN, 2003).

- Análise e Projeto: o objetivo deste processo é traduzir os requisitos numa especificação que descreva como implementar o sistema. “A especificação deve ser tão detalhada quanto for necessário (...). Em alguns casos ela é tão detalhada que o código fonte pode ser gerado automaticamente a partir da especificação” (MARTINS, 2006). Os profissionais envolvidos nesta etapa são: Arquiteto, Projetista, Projetista de Banco de Dados, Revisor da Arquitetura e do Design. E os artefatos envolvidos são: modelo de análise, modelo de design, interface, descrição da arquitetura do software e modelo lógico e físico.
- Implementação: é a fase de programação das classes, objetos e componentes. “Cada versão é construída através da integração de novos componentes e subsistemas” (MARTINS, 2006). Os profissionais alocados são: Implementador, Integrador de Sistema, Arquiteto e Revisor de Código. E os artefatos criados são: subsistema implementado, componente e plano de integração das versões.
- Testes: o processo de teste verifica e avalia a qualidade do produto. Vários tipos de teste são descritos para RUP, de acordo com Kruchten (2003) testes de unidade, integração e aceitação são os que mais se destacam. Os profissionais envolvidos neste processo são: Projetista dos testes e o Testador. E os artefatos gerados são: plano de teste e modelo de teste.
- Entrega: disponibiliza o sistema para os usuários. Este processo cuida das atividade de implantação e treinamento.

Os processos de suporte são:

- Gerenciamento de Configuração: o objetivo deste processo é manter atualizado tudo que for produzido pelo processo através dos controles de versões. Desenvolvedores, Integradores e Gerente de Controle de Mudanças são os responsáveis por este gerenciamento.

- Gerencia de Projeto: define o controle geral do processo e das atividade realizadas em cada fase. O Engenheiro do projeto é o principal trabalhador do processo. (MARTINS, 2006).
- Ambiente: este processo provê o suporte para o processo de desenvolvimento, ou seja, cuida da infra-estrutura do mesmo. O principal trabalhador no processo é o Engenheiro de Processo e o principal artefato é o caso de desenvolvimento (MARTINS, 2006).

A vantagem do RUP é ser bem completo e detalhado, porém, por ser muito complexo e conter uma série de atividades, papéis e artefatos, o RUP costuma ser visto como um processo pesado e burocrático, e difícil de identificar que elementos devem ser usados em cada fase do projeto.

3. DESENVOLVIMENTO ÁGIL

O desenvolvimento ágil surgiu do que os desenvolvedores diziam ser as fraquezas da engenharia de software convencional. Ela passou a valorizar, segundo AM (2007):

- Indivíduos e interações, em vez de processos e ferramentas.
- Software funcionando, em vez de documentação abrangente.
- Colaboração de cliente, em vez de negociação de contratos.
- Resposta a modificações, em vez de seguir um plano.

Isso trouxe vários benefícios, porém nem todos os projetos têm uma aplicação válida da modelagem ágil. “Em muitas situações, não podemos mais definir completamente os requisitos antes do início do projeto” (Pressman, 2006), sendo necessário adaptar a engenharia para atingir as necessidades dessa “agilidade”.

Na visão de JACOBSON, o acolhimento de modificações (a nível de equipe, software, tecnologias,etc.) é o principal guia para a agilidade.

Dentre os benefícios temos:

- Fácil comunicação entre os membros da equipe,
- Rápida entrega de software operacional,
- Adota o cliente como parte da equipe,
- Um plano de projeto flexível.

De acordo com a aliança ágil, que foi a união de dezessete especialistas em processos de desenvolvimento de software, representando os métodos Extreme Programming (XP), Scrum, DSDM, Crystal e outros (AM,2007), existem 12 princípios básicos para quem busca a agilidade.

1. Que a maior prioridade seja satisfazer o cliente desde o início por meio da entrega contínua de software valioso.
2. Modificações de requisito são bem vindas, mesmo que tardias no desenvolvimento. Os processos ágeis aproveitam as modificações como vantagem para a competitividade do cliente.

3. Entrega de softwares funcionando frequentemente, a cada duas semanas até dois meses, de preferência no maior espaço de tempo.
4. O pessoal de negócio e os desenvolvedores devem trabalhar juntos diariamente durante todo projeto.
5. Construção de projetos em torno de indivíduos motivados, fornecendo-lhes o ambiente e apoio que precisam e confiando que eles farão o trabalho.
6. O método mais eficiente e efetivo de levar a informação para dentro de uma equipe de desenvolvimento é a conversa face a face.
7. Software funcionando é a principal medida de progresso.
8. Progressos ágeis promovem desenvolvimento sustentável. Os patrocinadores, desenvolvedores e usuários devem ser capazes de manter um ritmo constante, indefinidamente.
9. Atenção contínua à excelência técnica e ao bom projeto facilitam a agilidade.
10. Simplicidade - a arte de maximizar a quantidade de trabalho não efetuado é essencial.
11. As melhores arquiteturas, requisitos e projetos surgem de equipes auto-organizadas.
12. Em intervalos regulares, a equipe reflete sobre como se tornar mais efetiva, então sintoniza e ajusta adequadamente seu comportamento.

Resumindo, um processo ágil deve gerenciar a imprevisibilidade. Para isso ele deve ser adaptável a mudanças e para essa adaptação, faz-se necessário um suporte incremental. Isso por sua vez necessita de um feedback (resposta do cliente) e o cliente quer por sua vez ver o processo andando, assim partes do sistema (protótipos) precisam ser implementados. Esse feedback do cliente é quem vai influenciar as adaptações do processo.

“O desenvolvimento ágil enfoca os talentos e agilidades dos indivíduos moldando o processo a pessoas e equipes específicas.” (PRESMAN, 2006), uma equipe bem organizada deve ser capaz de tomar suas decisões e seguir

com seus compromissos. Ela deve ser competente e estar sempre focada em uma meta como cumprir um cronograma ou entregar um incremento de software.

A colaboração entre os membros da equipe deve ser constantes e todos devem estar a par da atual situação do projeto.A confiança que deve haver entre eles fortalecerá ainda mais a estrutura da equipe.

3.1 Métodos Ágeis

3.1.1 XP – Extremme Programming

XP é uma metodologia leve, usada em pequenas e médias equipes que desenvolvem softwares com requisitos que se modificam rapidamente. Seu processo de software é moldado em cima do paradigma da orientação a objeto, veja o fluxo na Figura 8 . “A programação extrema (XP) define a codificação como a principal atividade de um projeto de software.” (BECK,2004). Essa codificação aparece na comunicação entre os membros da equipe, na definição dos testes e na refatoração.

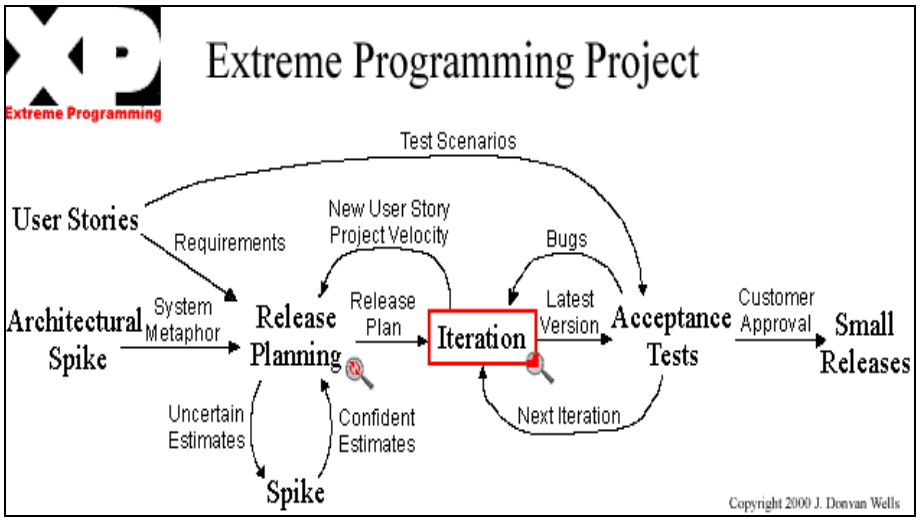

Figura 8 *– Modelo XP (XP, 2007).*

Através de suas iterações curtas o desenvolvimento torna-se rápido e os riscos do projeto são atacados a todo o momento. O código é revisado o tempo todo através da prática de programação aos pares. Essa técnica também

permite fazer verificações de qualidade em tempo real. Os testes são feitos antes, durante e depois da produção do software, e o cliente, o mais interessado, é colocado como membro da equipe para que possa com sua experiência ajudar no desenvolvimento do projeto especialmente na criação da *User Stories* (estórias que descrevem de uma forma sucinta as funcionalidades previstas no programa).

Durante o desenvolvimento de uma funcionalidade os programadores trabalham juntos. Os testes são projetados antes da codificação e até que todos sejam verificados, a funcionalidade não é incrementada ao projeto. Ao final da integração é feito o chamado teste de integração.

Segundo BECK (2004), a XP é a maneira mais leve, eficiente, de baixo risco, flexível, previsível, científica e divertida de desenvolver software. Ela permite ao programador trabalhar no que realmente lhe interessa , o código. E aos clientes e gerentes, perceber o progresso do projeto de uma forma concreta.

A XP se distingue de outras metodologias por:

- Ter seu feedback antecipado, concreto e contínuo derivado de ciclos curtos;
- Sua abordagem incremental de planejamento, que gera rapidamente um plano geral que vai evoluir com o decorrer do projeto;
- Sua habilidade de agendar de forma flexível a implementação das funcionalidades, respondendo as mutáveis necessidades de negócio;
- Sua confiança nos testes automatizados escrito por programadores e clientes para monitorar o progresso do desenvolvimento, para permitir que o sistema evolua e para detectar cedo os erros;
- Sua confiança na comunicação oral, teste e código fonte para comunicar a estrutura e o objetivo do sistema;
- Sua confiança em um processo de projeto evolutivo que dura tanto quanto o sistema em si;
- Sua confiança na intensa colaboração de programadores com habilidades comuns;

- Sua confiança nas práticas que combinam tanto os instintos de curto prazo dos programadores, quanto com os interesses de longo prazo do projeto.

Para maximizar o valor econômico do projeto, a XP prega que o ideal é que a equipe gaste menos investindo menos no início do projeto e ganhe mais através do marketing no produto. Aumentar também a probabilidade do projeto se manter vivo, garante um bom retorno financeiro ao projeto.

Para a XP a mudança é a única constante. Logo, é preciso estar atento para decidir o que fazer e com menos custo. “Precisamos controlar o desenvolvimento de software fazendo muitos pequenos ajustes e não uns poucos grandes ajustes” (BECK, 2004).

A XP trabalha com quatro valores que ajudam no gerenciamento do desenvolvimento de software.

- Comunicação: a comunicação é fundamental para o desenvolvimento das atividades XP. Através de práticas como testar programas em pares e estimar, a equipe vai se comunicando e estabelecendo vínculo entre os membros.
- Simplicidade: fazer funcionar da forma mais simples possível é o que a XP propõe. Caso seja necessário alterar ou fazer alguma mudança, faça depois. Unindo simplicidade e comunicação é possível precisar o que é necessário ser feito e o que não é.
- Feedback: “o feedback funciona em diferentes escalas de tempo” (BECK,2004). O programador tem feedback do estado do sistema, o cliente tem feedback do programador quando ele descreve as funcionalidades em estórias. Quanto mais feedback mais comunicação e consequentemente, simplicidade. A idéia é obter o feedback, interpretar e aplicar soluções o mais rápido possível.
- Coragem: coragem para mudar no meio do caminho não deve faltar.

Os Princípios básicos e fundamentais da XP são:

- Feedback rápido – um dos princípios básicos da XP é obter o feedback, interpretá-lo e aplicar o que é aprendido no sistema o mais rápido possível.
- Simplicidade presumida – ser simples e não complexo, fazer o trabalho (teste, refatoração e comunicação) diário e confiar na sua capacidade de acrescentar complexidade depois, onde for necessário.
- Mudanças incrementais – resolva os problemas com uma série de pequenas mudanças. “Grandes modificações de uma vez só, simplesmente não funcionam” (BECK,2004).
- Aceitação das mudanças – procure ter várias opções para resolver problemas mais urgentes, se necessário faça mudanças.
- Alta qualidade – buscar sempre o melhor para a equipe, caso contrário, o projeto poderá não ser concretizado.
- Investimento inicial pequeno – trabalhar com poucos recursos. Segundo BECK, recursos demasiados muito cedo em um projeto é uma receita para o desastre do projeto.
- Experimentação concreta – quanto mais se decide mais riscos se acumula. Por isso, todas as decisões devem ser testadas para garantir o sucesso do projeto.
- Aceitação de responsabilidades – as responsabilidades devem ser aceitas e não impostas.
- Métricas genuínas – a adoção de medidas ajuda a controlar o desenvolvimento do projeto.

As quatro atividades básicas do desenvolvimento segundo BECK, são: codificar, testar, escutar e projetar. Estruturando essas quatro atividades e compensando pontos fortes e fracos de cada prática veremos que a XP realmente funciona. Segue abaixo essas práticas:

- *O jogo do planejamento* – “o desenvolvimento de software é sempre um diálogo evolutivo entre o possível e o desejado” (BECK,2004). Ele começa com o cliente descrevendo estórias e funcionalidades requeridas para o software. O planejamento é feito pelas pessoas da

área de negócio que decidem sobre o escopo do projeto, a prioridade dos requisitos e as datas de entrega; e por pessoas da área técnica que decidem sobre estimativas de prazos, conseqüências das decisões de negócio, o processo e o cronograma detalhado do projeto. As reuniões são geralmente curtas e feita com os membro de pé. Novas *user stories* podem ser escritas a qualquer momento, entretanto, cliente e equipe devem trabalhar juntos para decidir a prioridade e a ordem de desenvolvimento das estórias. Como exemplo temos:

- Todas as estórias serão implementadas imediatamente.
- As estórias com valor mais alto serão antecipadas no cronograma e implementadas primeiro.
- As estórias de maior risco serão antecipadas no cronograma e implementadas primeiro.

- *Entregas Freqüentes* – entregue sempre pequenas versões que contemplem os requisitos de maior valor de negócio. Planeje um ou dois meses, em vez de seis meses ou um ano. Logo depois que a primeira versão for entregue (o primeiro incremento de software), a equipe calcula a velocidade do projeto, que é a quantidade de histórias do cliente que foram implementadas no primeiro incremento. Essa variável ajuda a estimar datas de entrega e o cronograma dos próximos incrementos. Caso ocorra algum atraso no projeto, as datas de entrega final são alteradas.
- *Metáfora* – cada projeto XP deve ser guiado por uma metáfora. Ela é uma forma fácil de conversar com o cliente e um meio simples de deixar todos os membros da equipe cientes do que será feito. ”A metáfora na XP substitui muito daquilo que outras pessoas chamam de arquitetura” (BECK, 2004).
- *Projeto Simples* – projete da maneira mais simples possível. É o chamado princípio KIS (Keep it Simple – Mantenha a simplicidade). Execute os testes, retire as lógicas duplicadas e faça isso com o

menor número de métodos e classes possíveis. Acrescente o que precisa quando você precisar. A implementação se baseia no modelo como a estória foi escrita. O uso de funcionalidade extra é desencorajado. Segundo Pressman (2006), Cartões CRC (Class Responsability Colaboration) é o único produto de trabalho da fase de projeto. Esse cartão é dividido em três seções. No alto do cartão o nome da classe. No corpo são listadas as responsabilidades da classe à esquerda e os colaboradores à direita.

- *Testes* – “Qualquer aspecto/função do programa que não tenha um teste automatizado simplesmente não existe” (BECK,2004). Os programadores devem escrever teste de unidade e os clientes testes de funcionalidades, o resultado disso é um programa mais confiável e uma redução nos riscos do projeto. Se um problema (uma situação de risco) é encontrado na parte de projeto, o XP recomenda que um protótipo (solução de ponta) para a resolução desse problema seja logo criado para ser avaliada. “A intenção é diminuir o risco quando a implementação começa a validar as estimativas originais correspondentes à estória que contém o problema do projeto” (Pressman,2006).
- *Refatoração* – é o processo de modificar um sistema de software de tal modo que ele não altere o comportamento externo do código, mas aperfeiçoe a estrutura interna, ou seja, é a capacidade de simplificar o programa, mantendo todos os testes em execução. Isso sempre deve ser feito quando se encontrar código duplicado ou o programador achar que é necessário. “Deve-se notar, no entanto, que o esforço necessário à refabricação pode crescer sensivelmente à medida que o tamanho de uma aplicação cresce” (Pressman, 2006).
- *Programação em Pares* – todo código de produção é escrito por dois programadores em uma máquina. Duas pessoas trabalhando juntas no computador, na mesma estória. Isso garante uma melhor qualidade do produto final, além de manter o foco de duas cabeças pensantes no

mesmo problema. Enquanto um codifica, o outro auxilia verificando os padrões no código, dando dicas, propondo soluções melhores, etc. Quando necessário a dupla inverte os papéis.

- *Propriedade coletiva* – o código está aberto pra todos os membros da equipe e pode ser modificado a qualquer momento.
- *Integração Contínua* – integração e atualização devem ser feitas cada vez que uma tarefa é terminada. À medida que os pares de programador vão completando suas estórias elas vão sendo integradas umas as outras por uma equipe de integração ou pelos próprios programadores. Quando um par de programadores integra e algum dos testes falha, fica fácil saber quem deverá fazer as correções. Essa estratégia de integração contínua evita problemas de compatibilidade e interface, e ajuda a descobrir rapidamente os erros.
- *Semana de 40 horas* – A equipe deve trabalhar apenas 40 horas por semana, mais do que isto se torna desgastante. Evitar fazer hora extra por duas semanas seguidas também é recomendável.
- *Cliente Presente* – o cliente deve fazer parte do time e estar disponível o tempo todo para ajudar no processo.
- *Padrão de Codificação* – a padronização ajuda para um melhor entendimento do projeto e uma melhor comunicação dos programadores e deve ser adotada voluntariamente por todo o time.

Seguindo essa prática, a XP certamente irá ajudá-lo no desenvolvimento do seu projeto de software. Ela pode não funcionar em situações como:

- Em equipes grandes e espalhadas geograficamente;
- Situações onde não se tem controle sobre o código (código legado que não pode ser modificado);
- Situações onde o feedback é demorado;

3.1.2 Scrum

É um modelo ágil que surgiu no início da década de 90 e foi desenvolvida por Jeff Suterland e sua equipe. O método ágil Scrum tem como objetivo, segundo Zanatta (2004), definir um processo para projeto e desenvolvimento de software orientado a objeto, que seja focado nas pessoas e que seja indicado para ambientes em que os requisitos surgem e mudam rapidamente. O Scrum também é considerado um método específico para o gerenciamento do processo de desenvolvimento de software.

Os princípios Scrum (Pressman, 2006) são consistentes com o manifesto ágil:

- pequenas equipes de trabalho;

- o processo precisa ser adaptável tanto a modificações técnicas quanto de negócios;

- o processo produz freqüentes incrementos de software;

- o trabalho de desenvolvimento e o pessoal que o realiza é dividido em partições claras, de baixo acoplamento, ou em pacotes.

- testes e documentação constantes são realizados ao passo que o produto é construído.

- o processo Scrum fornece a habilidade de declarar o produto pronto sempre que necessário.

Faz parte do método Scrum as seguintes atividades: requisito, análise, projeto, evolução e entrega. Ele incorpora um conjunto de padrões de processo que enfatiza prioridades de projeto, unidade de trabalho compartimentalizadas, comunicação e feedback freqüente do retorno.

O fluxo do Scrum acontece da seguinte maneira:

Pendências – cria-se uma lista priorizada de requisitos ou características importantes do projeto. Novos itens podem ser adicionados a pendência, ficando a cargo de o gerente atualizar a nova lista.

Sprints – é um intervalo de tempo, normalmente 30 dias, onde os itens em pendências são trabalhados pela equipe. Nesse momento não pode haver

modificações nos itens de pendência, o que torna o ambiente de trabalho mais estável.

Reuniões Scrum – são reuniões curtas feitas diariamente pela equipe. Ela é liderada pelo Scrum Master e tem por objetivo “socializar o conhecimento” (Pressman,2006). Ela deve responder às seguintes questões:

- O que você fez desde a última reunião da equipe?
- que obstáculos você está encontrando?
- O que você planeja realizar até a próxima reunião da equipe?

Demos – é a entrega de incrementos de software ao cliente para que possa ser avaliado pelo mesmo. Não significa que as demos tenham todas as funcionalidades implementadas.

3.1.3 DAS – Desenvolvimento Adaptativo de Software

Foi um modelo proposto por Jim Highsmith como uma técnica para construção de sistemas e software completo e ressalta a colaboração humana e a auto organização da equipe. O DAS é organizado em três atividades:

- Especulação: o projeto é iniciado com a idéia geral do cliente, os requisitos básicos e as restrições do projeto. É o chamado ciclo adaptativo.

- Colaboração: a equipe trabalha em conjunto buscando o máximo de cada um, fazendo com que colaborem na busca por melhores resultados. Segundo Pressman (2006), a abordagem colaborativa é um tema recorrente em todos os métodos ágeis.

- Aprendizado: à medida que os membros da equipe vão desenvolvendo as atividades do ciclo adaptativo, o aprendizado vai melhorando seu nível de compreensão do projeto e consequentemente vai aperfeiçoando a qualidade da equipe.

O DAS usa um processo interativo que incorpora o ciclo adaptativo, um rigoroso levantamento de requisitos e um ciclo de desenvolvimento focado no cliente e revisões técnicas formais em tempo real, baseadas no feedback (PRESSMAN, 2006).

3.1.4 DSDM – Método de Desenvolvimento Dinâmico de Sistemas

É uma abordagem ágil que “fornece um arcabouço para construir e manter sistemas que satisfazem às restrições de prazo apertadas por meio de uso de prototipagem incremental em um ambiente controlado de projeto” (PRESSMAN,2006). É uma versão modificada do princípios de Pareto, onde 80% do software pode ser entregue em 20% do tempo que levaria pra ser completamente acabado.

O DSDM é um processo interativo onde a cada incremento de software é necessário apenas o trabalho suficiente a fim de facilitar o avanço para o incremento seguinte. Os detalhes restantes são completados depois quando mais requisitos de negócio forem conhecidos ou alguma modificação for solicitada.

O ciclo de vida do DSDM se resume a três ciclos iterativos diferentes, precedidos por duas atividades adicionais de ciclo de vida:

- Estudo de viabilidade: estabelece os requisitos básicos e as restrições de negócio, avaliando se o projeto é viável ao processo DSDM.

- Estudo de negócio: estabelece os requisitos funcionais e de informação, define a arquitetura básica da aplicação e identifica os requisitos de manutenabilidade.

- Iteração do modelo funcional: os protótipos produzidos são mostrados ao cliente. “O intuito durante esse ciclo interativo é obter requisitos adicionais pela provocação de feedback pelos usuários à medida que eles exercitam o protótipo” (PRESSMAN,2006).

- Iteração de projeto e construção: analisa os protótipos da fase de iteração funcional, para garantir que os mesmo forneçam valor de negócio operacional para o usuário final. “Em alguns casos, a iteração do modelo funcional e a iteração de projeto e construção ocorrem simultaneamente” (PRESSMAN,2006).

- Implementação: coloca o último incremento de software no ambiente operacional.

3.1.5 Crystal

É um conjunto de processos ágeis, definidas por Alistair CockBurn e Jim Highsmith, cada qual com seus elementos centrais buscando entregar softwares úteis funcionando. Esse conjunto, chamado família Crystal, inclui grande número de diferentes métodos que são selecionados de acordo com as características do projeto a ser desenvolvido.

Os membros da família Crystal são identificados por cores, que indicam a intensidade do método. Ou seja, quanto mais escura a cor, maior é a complexidade do projeto, figura 9.

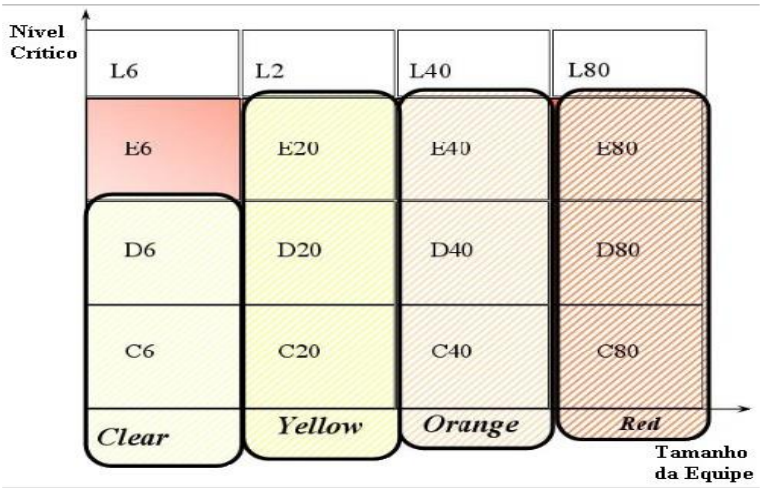

Figura 9 *– Família Crystal, (Silva et al, 2006)*

Suas características principais são:

- desenvolvimento incremental com ciclos de no máximo quatro meses.
- grande ênfase na comunicação e cooperação das pessoas.
- não limitação de quaisquer práticas de desenvolvimento, ferramentas ou produtos de trabalho.
- incorporação de objetivos para reduzir produtos de trabalho intermediários e desenvolvê-los como projetos evoluídos.

Existem três métodos principais desenvolvidos:

- Crystal Clear, usado em pequenos projetos, com até 6 desenvolvedores;
- Crystal Orange , desenhado para projetos de tamanho médio, com um total de 10 a 40 desenvolvedores;
- Crystal Orange Web, que é o Crystal Orange acrescido de práticas específicas para web.

3.2. Trabalhos Relacionados

3.2.1 Easy Process

É uma metodologia de desenvolvimento mais voltada para a área acadêmica. Ela foi criada pelo grupo de pesquisa e desenvolvimento em engenharia de software do Departamento de Sistemas e Computação da Universidade Federal de Campinas e se apresenta como um processo simplificado, apoiado nas práticas XP, RUP e Modelagem Ágil.

Seu fluxo de trabalho, conforme Figura 10, começa com a identificação do escopo do problema. Em seguida, vem a definição dos papéis, onde são recomendados pelo menos cinco papéis: o cliente, o usuário, o gerente, o desenvolvedor e o testador. Logo depois, é feita uma conversa com o cliente para saber as suas necessidades e funcionalidades do projeto. Com essas funcionalidades em mãos, é gerado o documento de visão contendo as idéias gerais sobre o que o sistema se propõe a fazer, baseado nos processos de negócio do cliente.

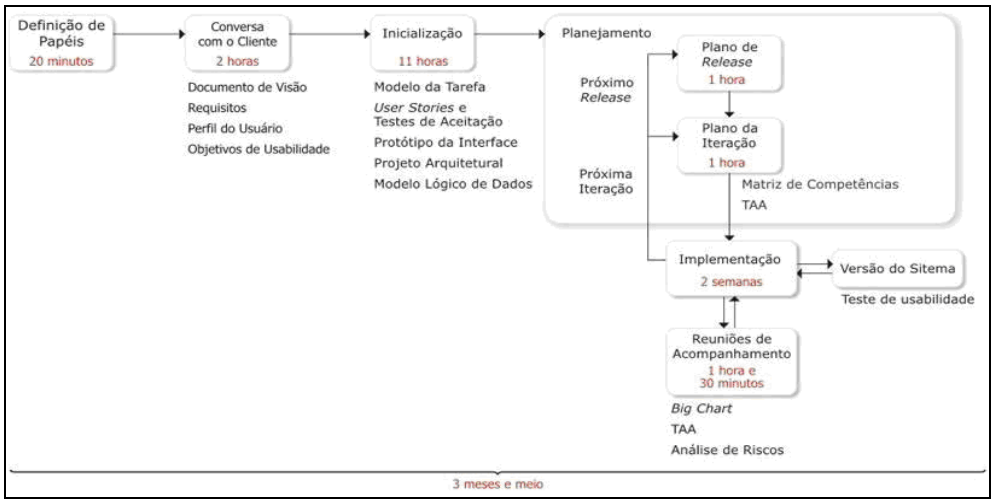

Figura 10 – *Síntese do YP (YP, 2007)*

Com o documento de visão aprovado, começa a fase de inicialização do YP. Nela são definidas as *User Stories*, é elaborado o projeto arquitetural e é construído o modelo lógico de dados. As *User Stories* são alocadas em *Releases* (o YP recomenda três *Releases* com duas iterações de duas semanas cada uma) e priorizadas por ordem de importância.

Durante a fase de implementação são feitos testes contíguos, revisão de código e refatoramento, terminada a iteração, uma nova é recomeçada, até que todas as *releases* sejam implementadas.

3.2.2 XP1

É um processo de desenvolvimento de software criado para atender as necessidades dos alunos de Projeto em Computação I, da Universidade. Segundo Sauvé (2007), este processo é particularmente indicado quando os requisitos funcionais do software a ser desenvolvido não estiverem muito bem definidos, pois permite fazer mudanças a um custo menor.

Ele se baseia nas práticas de Extreme Programming (XP), mas com algumas mudanças e simplificações. Possui papéis definidos para cliente, desenvolvedor e gerente de projeto.

O cliente descreve as funcionalidades desejadas, define os requisitos não-funcionais, elabora o plano de release e desenvolve os testes de aceitação.

O desenvolvedor ajuda a levantar as *user stories* e os requisitos funcionais junto ao cliente, elabora o projeto arquitetural, o plano de release e o plano de iterações, escreve os testes de unidade e de aceitação, e é responsável pelo refatoramento.

O gerente de projeto é responsável por conduzir as atividades de planejamento, avaliar os riscos e manter o progresso do projeto.

Seu fluxo se baseia nas seguintes atividades:

- Atividade de planejamento: escrita das *user stories*, levantamento dos requisitos não-funcionais, planejamento de release e iteração.
- Atividade de projeto: definição do projeto arquitetural, do projeto lógico dos dados e refatoramento do código;
- Atividade de testes: elaboração dos teste de aceitação, teste de unidade e o *test review*.
- Atividade de integração: inicia o controle de versões, realiza check-in e check-out das partes integradas.
- Atividade de Gerência: controlar a gerência dos riscos e publicar o andamento do processo.

4. PROCESSO CREATIVO

Neste capítulo é proposto um processo de desenvolvimento de software para a empresa Creativer, com regras e práticas que auxiliarão no desenvolvimento de sistemas de software, levando em consideração os princípios da Engenharia de Software.

A adoção a processos híbridos, que mesclam técnicas de dois ou mais processos, tem se tornado comum nos tempos atuais. E o aproveitamento dos pontos fortes de cada processo tem ajudado a conduzir a construção de produtos de boa qualidade (PETERS, 2001).

4.1 Empresa Creativer – Soluções em Saúde

A Creativer – Soluções em Saúde é uma empresa localizada em Vitória da Conquista – BA, que está no mercado há quatro anos e presta serviços de consultoria na área da saúde desenvolvendo sistemas que dão suporte a essa consultoria. São sistemas de pequeno e médio porte que implementam soluções para serviços da própria empresa e soluções externas baseadas no fluxo do SUS – Sistema Único de Saúde.

O Departamento de Desenvolvimento de Sistemas (DDS) é o setor responsável pela área de TI da empresa. Esse setor tem apresentado vários problemas decorrentes da falta de um método padrão para desenvolvimento de seus projetos, ou seja, o desenvolvimento de sistemas na empresa ainda acontece de uma forma desestruturada. O tempo é muito levado em consideração, ou seja, quanto mais rápido ficar pronto melhor. Assim, algumas etapas da engenharia de software acabam sendo atropeladas. Os requisitos não são documentados e apenas o modelo lógico dos dados e o código-fonte são gerados como documentação. Apenas um padrão de codificação e o reuso de funções são utilizados.

4.1.1 Descrição da Equipe de Desenvolvimento de Sistemas

Analista de Sistemas: ele é responsável pela análise dos requisitos e por ajudar a dar manutenção no código fonte. Acumula ainda as atividades de consultor, baseada em seu conhecimento adquirido na empresa. Também ajuda na instalação e manutenção dos programas.

Programador: é o responsável por codificar os programas; tem liberdade para alterar requisitos mediante aprovação do analista; dá manutenção no código-fonte; faz o planejamento da interface gráfica dos sistemas; instala, oferece suporte e treina os usuários dos sistemas.

Suporte técnico: é o responsável por dar manutenção em hardware e software; fazer a instalação dos sistemas e dar treinamento aos usuários.

Cliente/Usuários: são responsáveis por definir as funcionalidades do sistema. São as pessoas que usufruem dos produtos e consultoria viabilizados pela empresa. São eles: prefeituras de pequenos municípios da região sul e sudoeste da Bahia (na pessoa de funcionários que atuam na área da saúde do município) .

4.1.2 Análise das atividades desempenhadas pela equipe

Após observações diretas foi possível descrever o fluxo das atividades de desenvolvimento de software da equipe da Creativer. A Figura 11 representa em alto nível de abstração o desenvolvimento de sistemas na empresa.

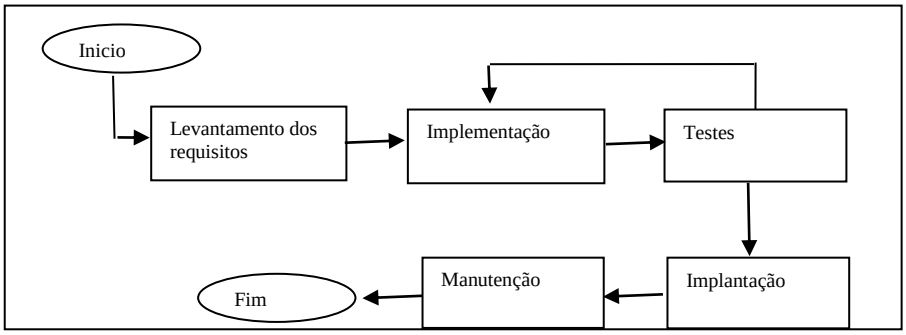

Figura 11 – *Fluxograma do processo atual de desenvolvimento da Creativer - Resumo.*

Pode-se observar que o software é construído com pouquíssima documentação e sem nenhum tipo de padronização das tarefas e atividades realizadas. Os requisitos são identificados, porém, não são documentados. Apenas uma modelagem do banco de dados é criada. Esses requisitos em sua maioria são identificados pelo analista e são levantados junto ao próprio fluxo do SUS, em conjunto com programas disponibilizados pelo mesmo, e junto ao cliente. Nenhum planejamento é feito para definir a ordem de implementação dos requisitos, o próprio programador é quem se encarrega de decidir quais funcionalidades implementar.

Durante a implementação, o reuso de funções e módulos é muito utilizado, e um padrão de codificação é seguido na hora do desenvolvimento do código fonte. Os testes de cada módulo são feitos conforme a conclusão do mesmo e não é usado nenhum plano de teste. Os testes são geralmente feitos pelo próprio programador do módulo, usando os recursos do compilador.

Segundo os programadores, os testes quando realizados pelos mesmos acabam não tendo um bom aproveitamento, pois o programador acaba sendo condicionado nas funcionalidades previstas no teste. Consequentemente, em alguns momentos do teste, poderão ocorrer erros imperceptíveis que indiretamente irão parar nas mãos do usuário final. Isso implicará no aumento de custos financeiros e de tempo com manutenções.

O uso de comentários é feito no próprio código fonte e a XP prega que código muito comentado acaba se tornando um código sujo e de difícil entendimento.

O uso de versões é sempre utilizado, contudo, muitas acabam não sendo devidamente documentadas, o que dificulta o entendimento das alterações e atualizações que foram feitas.

As atividades de transição são feitas pelo suporte técnico. Normalmente é oferecido um treinamento ao cliente e um acompanhamento com o mesmo, haja vista, que nenhum manual de utilização do programa é gerado. O suporte geralmente é feito on-line, por telefone ou troca de mensagens, e quando se faz necessário, o suporte é enviado até o município para sanar o problema.

Por conseguinte, ficou evidente que a CREATIVER por não utilizar nenhum processo para a produção dos seus sistemas sofre com a falta de documentação e organização de desenvolvimento, e com o alto custo na manutenção de seus programas.

4.2 Definição do Processo CREATIVO

Esta sessão descreve um processo de desenvolvimento de software para a empresa Creativer – Soluções em Saúde. Ele terá como finalidade criar um ambiente de desenvolvimento que proporcione qualidade, baixo custo e agilidade no desenvolvimento dos produtos da empresa. O processo descrito apóia-se nos seguintes princípios:

1. É baseado nas práticas de Extreme Programming (XP) e do RUP, mas com algumas mudanças e simplificações para que se adaptem facilmente às necessidades da Creativer.
2. Pretende abraçar as tarefas imprescindíveis: aquelas que precisam ser feitas em qualquer processo de desenvolvimento de software, por mais simples que sejam.
3. Visa produzir uma documentação mínima, que promova o entendimento do projeto e proporcione velocidade de planejamento.
4. Pretende ser tão simples que as chances de sucesso em aplicá-lo na empresa são muito altas.
5. Foi especialmente elaborado para levar em consideração a características especiais da realidade da empresa observando o número de funcionários e as tecnologias utilizadas na mesma.

Para se definir um processo, quatro variáveis devem ser analisadas: custo, tempo, qualidade e escopo. O escopo é a variável mais importante a ser estudada. “Se você controlar ativamente o escopo, poderá fornecer aos gerentes e aos clientes o controle sobre custo, qualidade e tempo.” (BECK,2004). Para controlar esse escopo serão definidas as seguintes características:

- Processo iterativo e incremental;

- Definição de papéis funcionais: Cliente/Usuário, Testador, Programador, Analista de Sistema e Gerente de Projeto.
- Ciclo de vida baseado nas quatro fases do RUP: Concepção, Elaboração, Construção e Transição;

A abordagem iterativa foi concebida para ser dinâmica e adaptativa, podendo acomodar mudanças de objetivos e estratégias. (MARTINS, 2006)

Como práticas teremos:

- Jogo do planejamento;
- Entregas freqüentes;
- Uso de metáfora;
- Testes;
- Refatoração;
- Propriedade coletiva;
- Integração contínua;
- Semana de 40 horas;
- Cliente presente;
- Padrões de codificação.

4.2.1 Definição da Equipe

Uma mesma pessoa poderá desempenhar mais de uma função dentro do processo, haja vista que a equipe da Creativer é restrita. A seguir serão apresentados os papéis funcionais sugeridos à equipe, ampliando as funções já apresentadas anteriormente, passando de três para cinco perfis.

A definição da equipe será disposta pelos seguintes papéis funcionais:

- Cliente/Usuário – ele é uma engrenagem fundamental do processo. O cliente define o escopo, a prioridade das funcionalidades, o conteúdo das *releases*, faz os testes de aceitação, define o modelo das interfaces e valida a versão ao final de cada *release*. Esse cliente

pode ser uma ou mais pessoas. Ele deve ter domínio do conteúdo de que se quer projetar.

São os clientes quem descrevem as User Stories que são passadas para a equipe de desenvolvimento. São a eles que o desenvolvedor recorre quando não entende a definição de algum requisito do sistema e são eles os responsáveis por aceitar ou não as funcionalidades implementadas em cada release, o chamado teste de aceitação.

Quando houver casos em que o cliente não existe, ou pelo menos ele não seja determinado, um dos consultores da empresa, que tenha visão do projeto a ser implementado, fornecerá os requisitos necessários para a construção do software.

- Testador: é a pessoa responsável pelos testes. Grande parte desses testes serão feitos pelo próprio programador durante o desenvolvimento. A pessoa irá trabalhar junto com o cliente definindo os testes funcionais e testando, a todo o momento na empresa, as funcionalidades e versões disponibilizadas pelos programadores. Os resultados de seus testes serão disponibilizados nos documentos de plano de teste e seus resultados deverão ser repassados aos programadores o mais rápido possível, independente dele ter sido validado ou não.
- Programador: o programador será a peça fundamental (juntamente com o cliente) do processo. Além de ser a pessoa responsável pela implementação do código, o programador deve ter uma boa comunicação com os membros da equipe. Ele tem que a todo o momento, informar a equipe sobre o andamento do projeto.

  Ele é o responsável também por participar do levantamento das *User Stories* e por estimar os prazos de entrega das mesmas. Terá participação em conjunto com o analista, para o planejamento das *releases* e a elaboração do modelo arquitetural e lógico, para que o mesmo fique a par do que será codificado. Deverá sempre buscar a

simplicidade no projeto, programar o que realmente é necessário para que funcione. Tem a obrigação de testar as funcionalidade implementadas, refatorar o código sempre que encontrar alguma duplicidade ou quando achar que é necessário.

Enfim, o Programador deve desenvolver seu trabalho buscando o melhor para o projeto e para a equipe. Deve ser cobrado mas nunca intimidado. Assim ele poderá trabalhar mais relaxado sempre focado nos seus objetivos e prazos.

- Analista de Sistema: pessoa responsável pela análise dos requisitos, definição da arquitetura e do modelo lógico. Participará ativamente no planejamento das releases e iterações. Ele deve ser uma pessoa que tenha conhecimento técnico aprofundado e que esteja atualizado com as novas tendências e tecnologias. Deverá também buscar o conhecimento sobre o domínio do problema. Ele estará o tempo todo em contato com o cliente ajudando a definir o escopo do projeto, e em contato com os programadores e testadores, ajudando a definir as estimativas e analisando os planos de teste gerados.
- Gerente do projeto: pessoa responsável por administrar todo o desenvolvimento e a equipe. Ele deve coordenar as reuniões e manter a equipe em equilíbrio. Deve apoiar e incentivar o tempo todo, intervindo quando realmente for necessário. Ele será responsável juntamente com o analista por analisar os riscos do projeto. Por fim, deve garantir os recursos necessários ao desenvolvimento e andamento do processo.

4.2.2 As fases do processo

As fases do processo serão definidas partindo do princípio das quatro fases do ciclo de vida do RUP aliadas as facilidades e práticas da modelagem Ágil. Elas serão úteis, pois definirão pontos chaves relevantes para o projeto.

A primeira fase será a fase de concepção. Nela teremos uma visão geral do projeto, ou seja, a definição de seu escopo. A segunda fase será a fase de elaboração. Ao final dessa fase teremos definido todo o planejamento para o andamento do projeto. A terceira fase é a fase de construção do produto, onde os programadores irão implementar as funcionalidades definidas nas fases anteriores. E por fim, a fase de transição, onde o resultado ou produto final será entregue em definitivo ao cliente.

4.2.2.1 Concepção:

O objetivo dessa fase é definir de uma forma geral o escopo do projeto. A única parte mais difícil da criação de um sistema de software é resolver com precisão o que deve ser criado (Astels et al Apud Brooks, 2002). A Tabela 1 resume as atividades realizadas durante essa fase.

Atividade	Responsável	Documentos
Criar uma visão geral do projeto	Cliente	- Documento de visão
Definição das User Stories	Analista de Sistema/Cliente	- Cartão de User Story
Estimativas do Projeto	Analista de Sistema/Programador	- Cartão de User Story
Análise de riscos do Projeto	Gerente do projeto	- Plano de Riscos

Tabela 1- *Fase de Concepção (Adaptado de Rocha, 2007).*

Após uma primeira conversa informal entre o cliente e o gerente, entramos na fase de concepção do CREATIVO. O cliente é o responsável por expor as funcionalidades do projeto e gerar o documento de visão, conforme Anexo I. Essa visão geral do projeto deve ser conseguida no menor espaço de tempo possível, e a descrição deve ser feita de uma forma sucinta.

A partir daí, entra em cena o analista do sistema. Ele é o responsável por documentar as funcionalidades do cliente nos cartões de *User Story*, conforme anexo II. Esses cartões devem conter um título com o nome da funcionalidade e seu corpo conterá uma descrição detalhada da funcionalidade. Durante a definição das *stories* é importante que o programador também esteja presente para que ele tenha conhecimento do que será desenvolvido.

Ao final da definição das *User Stories*, uma pilha de cartões deve ser gerada. Esses cartões são passados para os programadores, que irão analisar cada caso e estimar o tempo necessário para a implementação de cada *Story*. Essas estimativas não precisam ser perfeitas, apenas suficientemente boas (ASTELS et all,2004), até mesmo pela experiência dos programadores. Com o tempo e experiências de outros projetos essas estimativas acabam ficando mais reais. O ideal para programadores com pouca experiência é que eles superestimem esse tempo acrescentando 20% de tempo extra para atividades normais, e 50% ou mais, para atividades com conteúdo tecnológico novo e aparentemente complexo.

Essa estimativa de tempo fica especificada no cartão de *user story*. Caso não seja possível estimar o tempo de uma *user story* o programador deve informar ao analista a necessidade de dividir a *use storie* em partes menores, e documentá-la em novos cartões. O analista poderá consultar o cliente sobre essa necessidade, caso seja necessário.

Ao final de cada reunião o gerente do projeto fica responsável por analisar os ricos do projeto e documentá-los no plano de riscos, anexo V.

O fechamento dessa fase ocorre imediatamente após a definição do escopo e dos documentos especificados. Uma ou mais reuniões poderão ser feitas nesta fase até que esses objetivos sejam alcançados. Pode ser que durante o andamento do projeto seja necessário retornar a essa fase para modificar algum requisito, acrescentar uma outra funcionalidade ou refazer uma estimativa e até mesmo reformular o documento de visão. A equipe terá essa flexibilidade.

4.2.2.2 Elaboração:

O objetivo dessa fase é definir e projetar todo o planejamento para o andamento do projeto. “Projetar é criar uma estrutura que organiza a lógica do sistema” (BECK, 2004). Para que isso aconteça é necessário que as atividades

da fase de concepção tenham sido efetivadas. A Tabela 2 resume as atividades realizadas durante essa fase.

Atividade	Responsável	Documentos
Elaborar os planos de controle do projeto	Analista de Sistemas	- Plano de Release - Plano de Iteração - Plano de Tarefas
Definição da arquitetura	Analista de Sistema	- Modelo arquitetural
Definição do modelo de dados	Analista de Sistema/Programador	- Modelo de dados
Elaboração dos casos de teste	Testador	- Caso de teste

Tabela 2 *- Fase de Elaboração (Adaptado de Rocha, 2007).*

O Analista de Sistemas é o responsável por definir as *releases* e as iterações do processo. As *releases* devem descrever de forma clara e objetiva o planejamento das atividades que serão desenvolvidas na etapa de construção. Seriam como um cronograma do projeto, com cada release contendo duas iterações. A iteração é uma quantidade de tempo fixa que poderá variar de uma a três semanas de duração. Cada iteração termina com a liberação de um produto executável validado pelo programador e pelo testador.

Além de alocar as *user stories* nas *releases*, o analista deve também alocar um tempo para a definição do modelo arquitetural e do modelo de dados, caso exista algum banco de dados envolvido no projeto. Um plano de *release*, anexo III, será gerado para ilustrar a distribuição das *User Stories* que serão implementadas. Um plano de iterações, anexo IV, detalhará o tempo e a prioridade das *User Stories*. E um plano de tarefas, anexo VII, será criado para exibir os detalhes de cada *Story*, detalhes como quem esta implementando, o seu status e sua precedência. A ordem das *User Stories* poderá ser definida pelo cliente, porém o analista e o programador deverão deixá-lo ciente da implementação de uma *User story* que tecnicamente deva ser implementada primeiro.

Ao final de cada *release*, uma versão do software deve ser apresentada ao cliente para ser validada ou não. Caso não seja validada alguma funcionalidade, elas deverão ser revistas e um novo release é planejado pelo Analista.

O somatório das estimativas das *User Stories* será considerado o tempo total de duração do projeto. O número de iterações e release portanto, será definido de acordo com o tempo total estimado do projeto. Caso alguma coisa dê errado, novas *releases* e iterações poderão ser adicionadas para suprir tal dificuldade.

O testador tem a responsabilidade de definir os casos de teste para cada *User Story* e o gerenciador deve seguir atualizando sua tabela de riscos.

Essa fase é uma fase curta, e busca traçar de imediato uma solução para a fase de construção. As reuniões diárias, no ambiente de trabalho da empresa, poderão ser feitas durante esse período para que toda equipe tenha conhecimento do andamento do projeto e possa atualizar os artefatos do processo. Quanto à participação do cliente, apesar de ele poder ser de município distante ele deve estar presente nas reuniões que definem a prioridade da implementação das *User Stories*. Caso contrário, o analista se encarregará de definir essas prioridades.

Ao final desta fase os seguintes artefatos devem ser gerados :

- O Modelo Arquitetural.
- O Modelo de Dados.
- O Plano de *releases*, plano de iterações e o plano de tarefas.

4.2.2.3 Construção:

É a fase de codificação propriamente dita, onde o produto será construído e só deve ser iniciada após o término da fase de elaboração. Ela será trabalhada pelo programador e pelo testador, Tabela 3.

Atividade	Responsável	Documentos
Codificação das funcionalidades e refatoramento	Programador	- Código-fonte
Teste das User Stoties	Testador	- Caso de teste

Tabela 3 – *Fase de Construção (Adaptado de Rocha, 2007).*

O programador será o profissional responsável pela implementação do sistema. Ele deve seguir um padrão de codificação. O padrão de codificação é importante, pois permite um melhor entendimento do código fonte, permitindo uma melhor comunicação entre os programadores. As funções implementadas no código fonte devem ser semanticamente parecidas com a definição da *User Story*. “Usando os termos da metáfora, é possível manter a nomeação consistente e aumentar a facilidade de compreensão” (BECK,2004).

Cabe ao programador implementar as *User Stories* procurando a forma mais simples de resolvê-la. Futuramente se for necessário aumentar a complexidade, o programador deve fazer as mudanças devidas. Isso evitará perda de tempo na implementação de funcionalidades que não terão uma grande utilidade.

A refatoração pode e deverá ser usada cada vez que o programador achar necessária. Segundo Astels (2002), refatorar:

- Melhora o projeto do software evitando a decadência.
- Torna o código mais fácil de entender, mesmo quando você revisita o seu código.
- Ajuda você a encontrar os problemas e fornece uma perspectiva melhor do seu código.

Como as linguagem de programação oferecem depuração em tempo de desenvolvimento as funcionalidades serão implementadas pelo programador e testadas em tempo de produção. São os chamados testes de unidade e os mesmos não precisarão ser documentados, visto que o programador deve se concentrar na codificação e fazê-lo parar para documentar esses testes, aumentaria o tempo de desenvolvimento do projeto. Ao final do desenvolvimento da funcionalidade, a mesma será passada para o testador para que sejam feitos testes mais elaborados e documentados, anexo VI.

Os testes indicam se o programador terminou ou não de implementar a funcionalidade, ou seja, se todos os testes foram executados com sucesso e nenhum falhou, significa que o código pode ser produzir com a confiança de que não terá problemas no futuro. Sendo assim, um plano de teste deve ser

executado para cada *User Story*, é função do testador criá-los e executá-los, e dar um feedback ao programador independente do plano de teste ter sido validado ou não. “Programar e testar ao mesmo tempo é mais rápido e produtivo do que só programar” (BECK,2004).

Ao final da implementação de uma funcionalidade, quando a mesma tiver sido testada e validada pelo programador e pelo testador, ela deverá ser integrada ao projeto e novamente testada para evitar eventuais problemas de integração.

É importante observar que apenas uma integração poderá estar ocorrendo de cada vez. Durante a integração, é possível que casos de testes que funcionavam anteriormente passem a não funcionar. Como a iteração é de tempo fixo, e como a integração é seqüencial, você não pode demorar muito na integração e se as coisas complicarem, libere a vez de integrar para outro, e faça a integração "por fora", reiniciando todo o processo quando os testes estiverem passando. Essa demora pode indicar que você precise integrar mais freqüentemente (uma vez por dia, ou mais). Para minimizar problemas de integração, integre com freqüência.

Portanto, as atividades dessa fase serão:

- Tomar decisões de projeto
- Implementar o código (teste e produção)
- Revisar o código (teste e produção)
- Fazer refatoração do código (teste e produção)
- Informar a equipe sobre o andamento do projeto

4.2.2.4 Transição:

Esta fase ocorre ao final de uma release e seu objetivo é validar uma versão do software. Esse objetivo é representado na Tabela 4.

Atividade	Responsável	Documentos
Teste de Validação da versão	Cliente	- Atualização no cartão de *User Story*

Tabela 4 – *Fase de Transição (Adaptado de Rocha, 2007).*

Durante a fase de transição a versão gerada ao final de cada *release* será colocada no cliente. O sistema é entregue em incrementos ao longo do tempo. Cada incremento deve ter sido aprovado em seus testes para evitar que erros acabem parando nas mãos do cliente. Caso a versão entregue não seja aprovada pelo cliente, o mesmo terá o direito de pedir alterações, mas deve-se levar em conta que novas *user stories* poderão ser adicionadas ou removidas e que novas estimativas deverão ser feitas, consequentemente, uma atualização no planejamento das *releases*. O feedback nesta etapa é extremamente importante, comemorar caso obtenha sucesso é importante.

Para um maior controle do código fonte é indicado que os programadores utilizem alguma ferramenta de controle de versão. A ferramenta *subversion* [1] é indicada neste caso, já que a mesma é bem simples e fácil de usar.

Deixe que o cliente usufrua das funcionalidades previstas na *release*. Fechando assim o ciclo do processo que se reinicia com o desenvolvimento das próximas *releases*. Ao final das *releases* o cliente terá em mãos o produto pronto para ser usado.

4.3 Artefatos do processo:

Neste processo, "artefatos" representam informação que deve ser guardada sob o projeto de forma permanente e sempre atualizada. O CREATIVO é pouco formal em termos de artefato, numa tentativa de minimizar a formalidade e promover a agilidade. Os artefatos do CREATIVO são:

- Modelo Arquitetural do projeto – de responsabilidade do analista, visa representar a estrutura do sistema;
- Modelagem de dados – de responsabilidade do analista, visa representar estrutura dos dados armazenados e a sua manipulação;

[1] É um controle de versão free/open-source que permite armazenar documentos de qualquer natureza efetuando controle de acesso aos repositórios e mantendo as alterações através de um controle de revisões. Disponível em: < http://subversion.tigris.org/>

- Documento de Visão – de responsabilidade do cliente, visa descrever de uma forma geral o projeto. Anexo I;
- Cartão de *User story* – de responsabilidade do cliente, visa documentar cada funcionalidade especificada pelo mesmo. Anexo II;
- Plano de *Releases* – de responsabilidade do analista e terá a influência do cliente para definir as prioridades, visa formular o planejamento do projeto. Anexo III;
- Plano de Iteração – de responsabilidade do analista, visa definir um cronograma das tarefas. Anexo IV;
- Caso de Teste – de responsabilidade do testador, visa documentar os testes e garantir a qualidade do projeto. Anexo V;
- Plano de Risco – de responsabilidade do gerente, visa auxiliar as decisões buscando logo as suas resoluções. Anexo VI;
- Plano de Tarefas – de responsabilidade do Analista, visa documentar as atividades de cada programador. Anexo VII;
- Código – de responsabilidade do programador.

4.4 Adotando o processo

"Os fatores organizacionais determinam o sucesso ou o fracasso de um processo de desenvolvimento de software" (ASTELS, 2004). Todo o processo tem suas vantagens e desvantagens e o que pode ser um benefício para uma organização pode ser um problema para outra. Adaptar um processo, seja ele qual for, para uma equipe com um projeto já em andamento é mais complexo do que adotá-lo para uma equipe nova, pois os membros já estão condicionados a outras práticas e todavia as novas práticas sempre acabam sofrendo certa resistência.

Como as práticas do CREATIVO são simples, elas poderão ser adotadas com simples reuniões e exposição deste documento aos funcionários da empresa e um treinamento individual pode ser concedido para cada membro envolvido no processo.

Para começar, serão definidos os papéis, as pessoas que estarão diretamente envolvidas no processo. Estas pessoas devem estar dispostas a abraçar a idéia para que se obtenha um bom resultado.

Dificuldades em trazer o cliente para dentro da equipe, já que a maioria é de municípios vizinhos, poderão ser encontradas. Por isso, é ideal que ele entenda a sua importância e perceba que sua ajuda estará dando retorno num futuro próximo. Sem o feedback do cliente, a capacidade de entregar um sistema funcional fica mais distante. Ele será instruído a utilizar as metáforas para definir as funcionalidades previstas pelo mesmo e aprenderá a manusear o cartão de *user story*. Vale lembrar que esse cartão deve ser preenchido de forma sucinta e objetiva.

Ao gerente será confiada a direção geral do projeto. Ele deve estar ciente de suas obrigações descritas anteriormente e ter um perfil de liderança para sempre direcionar a equipe a um objetivo. A ele será ensinado como fazer o plano de risco e como utilizá-lo, para evitar que o projeto não dê certo.

Ao analista será entregue a parte mais formal do processo, a parte de planejamento. Será-lhe explicado como preencher os planos de release e de iteração. Ele deve estar ciente de que estes deverão estar sempre atualizados. O modelo lógico, que já era feito pelos programadores agora terá o analista como seu criador, além de criar também o modelo arquitetural do projeto. Pelo fato do analista ser um conhecedor da área de computação julgamos que ele não encontrará dificuldades para assumir esta responsabilidade. Ferramentas como o JUDE[2] e o ERWIN[3] , que já são utilizadas na empresa poderão dar apoio as processo CREATIVO na concepção do modelo arquitetural e do modelo lógico.

Os programadores talvez sejam as pessoas mais difíceis de adaptarem às novas práticas do CREATIVO. Porém, essa dificuldade tem boa chance de ser contornada devido à formação acadêmica na qual os dois programadores da

---

[2] JUDE ou Java and UML Developer Enviroment é uma ferramenta *freeware* que permite a criação de vários diagramas da UML. Disponível em :<https://jude.change-vision.com/jude-web/index.html>

[3] Erwin é uma ferramenta *shareware* para definição de modelo de dados. Disponível em: <http://www.e-central.com.br/ca/erwin>

empresa estão engajados. Práticas como código simples e padronizado e a refatoração devem ser frisadas.

Ao testador fica a responsabilidade de garantir a qualidade do produto. A forma como preencher o plano de testes e a importância de repassá-los aos outros integrantes será pregado como sua atividade principal.

Enfim, com um pouco de trabalho, persistência e acompanhamento espera-se que a adoção do processo por parte da empresa, aconteça de forma pacífica e que o mesmo seja sucesso. A empresa pode começar adotando as práticas de planejamento para que possa se estruturar melhor e em seguida adotar as demais práticas do processo.

5. CONCLUSÃO E TRABALHOS FUTUROS

A definição de um processo para guiar o desenvolvimento é fundamental para se conseguir qualidade e produtividade. A Engenharia de Software vem proporcionando isso desde o seu surgimento.

Nesta monografia, foi apresentado um processo para o desenvolvimento de software para ser adotado pela empresa Creativer – Soluções em Saúde, tendo por base o RUP e a XP, com o objetivo de aumentar a qualidade, reduzir custo e sistematizar o desenvolvimento de sistemas na empresa. A escolha desses dois processos se deu pela certeza de que ambos representam excelentes processos e garantem a qualidade do produto final.

Com a utilização do processo CREATIVO, espera-se que a analise dos requisitos seja melhor documentada, as atividades de cada profissional sejam melhores definidas, que as estimativas tornem-se mais precisas, que os produtos gerados apresentem maior qualidade e que a produtividade aumente em decorrência da introdução de uma abordagem sistemática de desenvolvimento de software.

Espera-se que algumas dificuldades sejam encontradas durante a fase de implantação devido a fatores como: mudança de paradigma de desenvolvimento de sistemas; ocorrência de treinamentos simultâneos à continuidade dos trabalhos; presença constante do cliente, tendo em vista que a maioria são de municípios vizinhos; mudança na cultura de alguns profissionais.

Como soluções futuras, recomenda-se a utilização do processo CREATIVO no desenvolvimento de um projeto piloto, avaliando ao final do projeto se houve alguma melhoria em termos de qualidade no produto final; o uso de diagramas UML para serem anexados as User Stories e permitir um melhor entendimento da mesma; e a utilização das técnicas de programação em pares, já que segundo Beck (2004), a programação em pares é mais produtiva, pois melhora a qualidade do design, código e testes, e proporciona uma revisão constante do código.

Referências

AM. *Agile Modeling*. Disponível em: <http://www.agilemodeling.com>. Acessado em: 7 jan. 2007.

ASTELS, David; MILLER, Granville; NOVAK, Miroslav. *Extreme Programming guia pratico*. Rio de Janeiro: Editora Campus, 2002.

FÁBRICA DE SOFTWARE III. *Metodologia de desenvolvimento de software*. Recife: UFPE, 2003. Disponível em: <http://www.cin.ufpe.br/~fabrica3/homePage processo.htm> Acesso em: 7 jan. 2007.

FALBO, Ricardo de Almeida. *A Experiência na Definição de Um Processo Padrão Baseado no Processo Unificado*. Artigo (Graduação em Ciência da Computação) – UFES, Departamento de Informática, Vitória, ES, 2000. Disponível em: < www.inf.ufes.br/~falbo/download/pub/Simpros2000.pdf>. Acessado em: 20 dez. 2006.

FILHO, Wilson de Pádua P. *Engenharia de Software-Fundamentos, métodos e padrões*. ed 2. Rio de Janeiro: LTC, 2001.

FOWLER, Scott. *UML Essencial Um breve guia para a linguagem-padrão de modelagem de objetos*. ed 2. Porto Alegre: Bookman, 2000.

IEEE. *IEEE standard for developing software life cycle processes*. Disponivel em: < http://ieeexplore.ieee.org/xpl/tocresult.jsp?isNumber=10452>. Acessado em: 02 jan 2007.

JACOBSON, Ivar; BOOCH, Grady; RUMBAUGH, James. *UML – Guia do Usuário.* Rio de Janeiro: Ed. Campus, 1999.

KENT, Beck. *Programação Extrema (XP) explicada acolha as mudanças*. Porto Alegre: Bookman, 2004.

KRUCHTEN, Philippe. *Introdução ao RUP Rational Unified Process*. Rio de Janeiro: Editora Ciência Moderna, 2003.

MARTINS, José Carlos C. *Gerenciando projetos de desenvolvimento de software com PMI, RUP e UML*.3 Ed. Rio de Janeiro: Brasport. 2006.

PETERS, James F; PEDRYCZ Witold. *Engenharia de software teoria e pratica*. Rio de janeiro: Editora Campus, 2001.

PRESSMAN, R. S. *Engenharia de Software*. 6 Ed. São Paulo: Editora McGraw-Hill, 2006.

ROCHA, Roberto do Santos. *Modelagem do Processo de Desenvolvimento e Manutenção de Software para a UINFOR/UESB.* Artigo – UESB Universidade Estadual do Sudoeste da Bahia, Vitória da Conquista.

RUP. *Rational Unified Process – Origem*. Disponível em: <http://pt.wikipedia.org/wiki/Rational_Unified_Process>. Acessado em: 07 jan. 2007.

SABBATINI, R.M.E. - *Internet e Medicina. Os Recursos*. Revista de Informática Médica: v.1,n.1,p. 5-9,Jan/Fev. 1998.

SAUVÉ, Jacques P.; SILVA, Airon F. da; NETO, Francisco R.; CABRAL, Francisco W. L.. *XP1-Um processo de desenvolvimento*. DSC, disponivel em <http://www.dsc.ufcg.edu.br/~jacques/projetos/common/xp1/xp1.html>. Acessado em: 05 jan. 2007.

SILVA, Ana E. Pedroso; UESONO, Gabriel; PEGHINI, Lívia. *Agili Métodos Ágeis*. Artigo (Graduação em Ciência da Computação) – Universidade Federal de São Carlos, Centro de Ciências Exatas e Tecnologia, São Carlo, 2006. Disponível em: < http://www.dc.ufscar.br/~rosangel/mds/Seminarios/MetodosAgeis.pdf >. Acessado em: 20 jan. 2007.

SOMMERVILLE, Ian. *Engenharia de Software*. 6. ed. São Paulo: Addison Wesley, 2003.

VAROTO, Cristina Ane. *Visões em Arquitetura de Software*. Dissertação (Mestrado em Ciência da Computação) – USP Universidade de São Paulo, São Paulo,2002.

VIEIRA, Leocília Aparecida. *Projeto de pesquisa e monografia – Normas da ABNT* . ed 3. Curitiba: Ed do Autor, 2004.

XP. *Extreme Programming: A gentle introduction*. Pagina oficial. Disponível em: <http://www.extremeprogramming.org/ >. Acessado em: 07 jan. 2007.

WIKIPEDIA.*Enciclopédia Livre*. GNU Free. Disponível em<http://pt.wikipedia.org.>. Acessado em: 07 jan. 2007.

YP. Easy Process. Pagina oficial. Disponível em: <http://www.dsc.ufcg.edu.br/~yp>. Acessado em: 27 jan. 2007.

ZANATTA Lazaretti, Alexandre. *xScrum: uma proposta de extensão de um Método Ágil para Gerência e Desenvolvimento de Requisitos visando adequação ao CMMI.* Florianópolis, 2004. Dissertação (Mestrado em Ciência da Computação) - Curso de Pós-Graduação em Ciência da Computação, Universidade Federal de Santa Catarina, Florianópolis, SC, 2004. Disponível em: < http://wer.inf.puc-rio.br/WERpapers/artigos/artigos_WER05/alexandre_zanatta.pdf >. Acessado em: 15 dez. 2006.

ANEXOS