TRABALHO DE CONCLUSÃO DE CURSO

O IMPACTO DA ADOÇÃO DE METODOLOGIAS DE DESENVOLVIMENTO DE SISTEMA EM EMPRESAS DE PEQUENO PORTE.

PROFESSORA ORIENTADORA: SANDRA

ÁREA: PROCESSAMENTO DE DADOS

SÃO PAULO – SP

2011

DENIS LUNA BORGES DA SILVA

O IMPACTO DA ADOÇÃO DE METODOLOGIAS DE DESENVOLVIMENTO DE SISTEMA EM EMPRESAS DE PEQUENO PORTE.

TRABALHO DE CONCLUSÃO DE CURSO APRESENTADO À FACULDADE FATEC COMO REQUISITO PARA OBTENÇÃO DO TÍTULO DE TECNÓLOGO EM PROCESSAMENTO DE DADOS

ORIENTADORA: PROFESSORA SANDRA

ÁREA: PROCESSAMENTO DE DADOS

SÃO PAULO – SP

2011

O IMPACTO DA ADOÇÃO DE METODOLOGIAS DE DESENVOLVIMENTO DE SISTEMA EM EMPRESAS DE PEQUENO PORTE.

DENIS LUNA BORGES DA SILVA

Resumo

Este trabalho de conclusão de curso visa estudar e demonstrar as diversas metodologias de desenvolvimento disponíveis na área de Tecnologia de Informação, abordando quais são suas origens, características, benefícios e desvantagens de sua adoção. Além desse estudo, este trabalho buscará estudar principalmente a resistência e o impacto da adoção de uma ou mais dessas metodologias em uma empresa de pequeno porte, que tem pouco tempo de vida e está em fase de crescimento. Porém o ambiente de desenvolvimento dessa empresa utiliza-se de poucas ou nenhuma dessas metodologias, ferramentas ou técnicas que ajudam a ter um desenvolvimento mais elaborado, e tem um tempo curto para finalizar seus projetos, gerando diversos atrasos ou versões instáveis.

Neste caso, será utilizado o ambiente de uma empresa chamada TEx Soluções, situada no bairro de Moema, na zona sul de São Paulo, cujo sistema principal foi iniciado há 3 anos e a empresa demonstra um desejo de se organizar para poder crescer e oferecer novas soluções para seus clientes.

Lista de ilustrações

Lista de Siglas

CC - Cópia de Cortesia

CRC - Classes, Responsabilidades e Colaborações

HTML - Hyper Text Markup Language

IBM - International Business Machines

RUP - Relational Unified Process

SVN - Sub Version

UML - Unified Modeling Language

VCL - Visual Component Library

XP - Programação Extrema (Extreme Programming)

Sumário

1 Introdução

Essa pesquisa pretende demonstrar primeiramente uma abordagem das metodologias de desenvolvimento de sistema, suas características de funcionamento e posteriormente, estudar o impacto que a adoção de uma dessas metodologias pode causar em uma empresa de pequeno porte, cujo ambiente de desenvolvimento não se utiliza de uma metodologia específica.

Embora seja importante o uso de uma ou mais metodologias de desenvolvimento, há um impacto ao adotar uma metodologia em um ambiente onde anteriormente nenhuma era utilizada. Após a introdução e o estudo de algumas das metodologias conhecidas, será analisado o impacto da adoção de uma ou mais dessas metodologias.

O cenário se passa em uma empresa de desenvolvimento de software chamada TEx Soluções em Informática, cujo sistema chamado TELEPORT é o carro chefe da empresa, sendo um sistema de multi cálculo de seguros, utilizado por diversos corretores de seguros.

A empresa de pequeno porte possui um conjunto de sete desenvolvedores e três responsáveis pelo suporte ao sistema. A princípio, a empresa não se utiliza de uma metodologia, mas sim algo que é conhecido como "Codifica e Corrige", ou seja, nada mais sendo o conceito de “apenas faça funcionar”.

Após a proposta, foi adotada a metodologia Programação Extrema (XP), e passados 3 meses os impactos e ganhos gerados por essa mudança serão analisados e expostos nesse estudo.

2 Um estudo sobre o que é metodologia

Já faz alguns anos que o desenvolvimento de software deixou de ser sinônimo apenas de código. Hoje em dia, sabe-se que é necessária a utilização de uma metodologia de trabalho.

Entende-se por metodologia, como a maneira ou forma de se utilizar um conjunto coerente e coordenado de métodos para atingir um objetivo. De modo que se evite, tanto quanto possível, a subjetividade na execução do trabalho, fornecendo um roteiro, um processo dinâmico e interativo para desenvolvimento estruturado de projetos, sistemas ou software, visando à qualidade e produtividade dos projetos.

O dicionário Websters (WEBSTERS[1], 1998), define metodologia como um conjunto de métodos, regras e postulados empregados por uma disciplina: um procedimento particular ou conjuntos de procedimentos.

É objetivo de uma metodologia definir de forma clara “quem” faz “o que”, “quando”, “como”, e até mesmo “onde”, para todos os que estejam envolvidos diretamente ou não com o desenvolvimento de software. Deve-se definir também, qual o papel dos técnicos, dos usuários, e o da administração da empresa no processo de desenvolvimento. Com isso, evita-se a situação na qual o conhecimento sobre o sistema é de poucos, comumente apelidados, de “os donos do sistema”. Além disso, deve-se instruir um conjunto de padrões preestabelecidos, de modo a ser evitar a subjetividade na abordagem, a fim de garantir fácil integração entre os sistemas desenvolvidos. Assim, o uso de uma metodologia possibilita:

- Ao gerente: controlar o projeto de desenvolvimento de software mantendo o rumo do projeto sobre controle para que não haja desvios de planejamentos de custos e prazos, que, se negligenciados ou mal conduzidos, podem por em risco o sucesso do projeto.
- Ao desenvolvedor: obter a base para produzir de maneira eficiente, software de qualidade que satisfaça os requisitos estabelecidos.

Muitas vezes, o uso de uma metodologia é encarado como cerceamento da criatividade dos técnicos, ou como, acréscimo de burocracia, leia-se documentações, por muitos tidos como desnecessário a construção de software. Uma metodologia não deve limitar a criatividade profissional, mas deve ser um instrumento que determine um planejamento sistemático, que harmonize e coordene as áreas envolvidas. O que limita a criatividade não é a metodologia, mas os requisitos de qualidade e produtividade de um projeto.

---

1 FONTE: WEBSTERS Ninth Neiv Collegiate Dictionary. 1998.

Como uma metodologia é um conjunto de métodos, convém definir o que é um método e qual o seu objetivo. Um método é abordagem técnica passo a passo para realizar uma ou mais tarefas indicadas na metodologia. Ou seja, é (são) o(s) procedimento(s) necessário(s) a ser (em) adotado(s) para atingir um objetivo. Já uma técnica, pode ser compreendida como sendo um modo apropriado de se investigar sistematicamente um universo de interesse ou domínio do problema. Para tanto, utiliza-se de uma notação, como exemplo de técnica, que são: análise estruturada, análise essencial, projeto estruturado e análise orientada a objetos.

3 Os tipos de metodologia de desenvolvimento de sistemas

A escolha de uma metodologia a ser utilizada no desenvolvimento, deve ser realizada com base na natureza do projeto e do produto a ser desenvolvido, dos métodos e ferramentas a serem utilizadas e dos controles e produtos intermediários desejados.

O uso de metodologia, mesmo que ainda não fortemente sedimentada, no desenvolvimento de software é de extrema importância, para que o sistema construído atenda as necessidades dos interessados, com um mínimo de qualidade.

Cada metodologia tem as suas características, qualidades e funcionalidades, e aqui será feito um breve estudo sobre as diversas disponíveis, bem como o estudo da adoção de uma metodologia em uma empresa de pequeno porte.

3.1 A Metodologia de Desenvolvimento Codifica e Corrige

Esta abordagem não pode ser chamada de metodologia no sentido real da palavra, mas é interessante mencioná-la, pois muitos desenvolvimentos ainda continuam utilizando esta abordagem. "Codifica e Corrige" nada mais é do que o conceito de “apenas faça funcionar”.

Inicialmente, o cliente pode fornecer uma especificação do que ele precisa, mas isto não será nada substancial. Esta especificação pode ser obtida por meio de algumas anotações, e-mail, ou de qualquer outra fonte não muito consistente. Esta abordagem se apoia nos conhecimentos da equipe para tentar preencher as lacunas.

O desenvolvimento então se inicia com ciclos rápidos de codificação seguidos por correção. De tempos em tempos, o desenvolvedor apresenta uma nova versão (ou release) da aplicação para o cliente para obter *feedback* e então continua o desenvolvimento.

Figura 1: A metodologia “Codifica e Corrige” [2]

A metodologia “Codifica e Corrige” possui diversos efeitos colaterais negativos:

- A qualidade do produto é baixa.
- O sistema frequentemente se transforma em um código “bagunçado”, com falta de adaptabilidade, reuso e interoperabilidade.
- Os sistemas são difíceis de serem mantidos e aprimorados.
- Os sistemas frequentemente tornam-se complicados e com baixa escalabilidade.

2 FONTE: GOMES, Andrey. Metodologias de Desenvolvimento de Software. São Paulo, 2010. Disponível em http://www.andreygomes.com/index.php?option=com_content&view=article&id=1:metodologias-de-desenvolvimento-de-software&catid=1:metodologias&Itemid=2. Acessado em 10 set. 2011.

3.2 A Metodologia de Desenvolvimento Balbúrdia

No início da computação, poucos programadores seguiam algum tipo de metodologia baseando-se, em sua maioria, na própria experiência. Era o que chamamos hoje de “Modelo Balbúrdia”, sistemas desenvolvidos na informalidade sem nenhum tipo de projeto ou documentação.

Neste modelo, o software tende a entrar em um ciclo de somente duas fases: o de implementação e de implantação. E os ajustes ao software para atender aos novos requisitos, sempre são em estados de urgência e de stress, motivados por vários fatores, e principalmente por pressão política. Portanto, havia a necessidade se criar um “Ciclo de Vida” mais inteligente para o desenvolvimento de software. Ou seja, um “Ciclo de Vida” semelhante à própria natureza, com início, meio e fim bem definidos (MCCONNELL[3], 2005).

3 MCCONNELL, S. CODE COMPLETE. *Um guia prático para a construção de software*. 2. ed. São Paulo: Bookman, 2005.

3.3 A Metodologia de Desenvolvimento em Cascata

A metodologia de desenvolvimento em cascata foi desenvolvida pela marinha norte-americana nos anos 60 para permitir o desenvolvimento de softwares militares complexos.

A partir de então, o modelo cascata tornou-se mais conhecido na década de 70 e é referenciado na maioria dos livros de engenharia de software ou manuais de padrões de software. Nele as atividades do processo de desenvolvimento são estruturadas em uma cascata em que a saída de uma é a entrada para a próxima.

Dessa maneira, o projeto segue uma série passos ordenados. Ao final de cada fase, a equipe de projeto finaliza uma revisão. O desenvolvimento não continua até que o cliente esteja satisfeito com os resultados.

As suas principais atividades são:

- Estudo de viabilidade
- Análise e especificação de requisitos
- Design da arquitetura
- Design detalhado
- Codificação e testes de unidades
- Integração e teste do sistema
- Entrega e instalação
- Manutenção

Existem muitas variantes deste modelo propostas por diferentes pesquisadores ou empresas de desenvolvimento e adaptadas a diferentes tipos de software. A característica comum é um fluxo linear e sequencial de atividades semelhantes a descritas anteriormente.

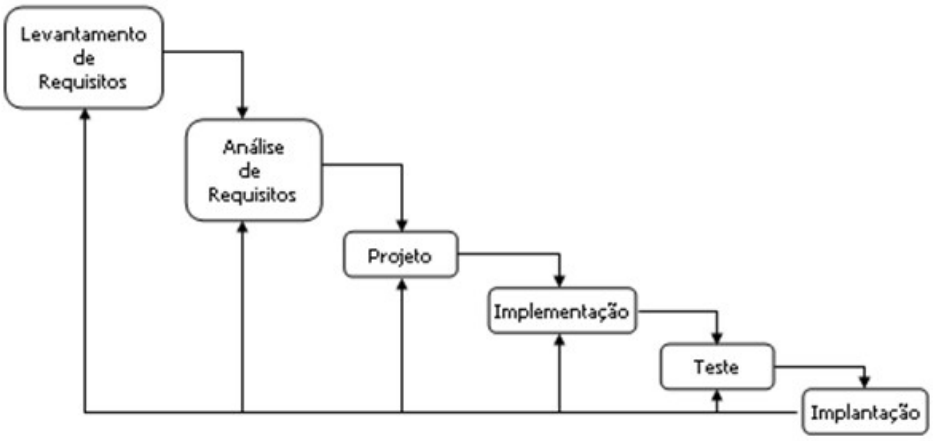

Figura 2: O "modelo cascata" [4]

Este modelo, quando proposto, introduziu importantes qualidades ao desenvolvimento. A primeira chama a atenção de que o processo de desenvolvimento deve ser conduzido de forma disciplinada, com atividades claramente definidas, determinada a partir de um planejamento e sujeitas a gerenciamento durante a realização. Outra qualidade, define de maneira clara, quais são estas atividades e quais os requisitos para desempenhá-las. Por fim, o modelo introduz a separação das atividades da definição e design da atividade de programação que era o centro das atenções no desenvolvimento de software.

O "Modelo Cascata" também é criticado por ser linear, rígido e monolítico. Inspirados em modelos de outras atividades de engenharia, este modelo argumenta que cada atividade apenas deve ser iniciada quando a outra estiver terminada e verificada. Ele é considerado monolítico por não introduzir a participação de clientes e usuário durante as atividades do desenvolvimento, mas apenas o software ter sido implementado e entregue. Não existe como o cliente verificar antecipadamente qual o produto final para detectar eventuais problemas.

Características particulares do software (ser conceitual, por exemplo) e a falta de modelos teóricos, técnicas e ferramentas adequadas mostram que é necessário haver maior flexibilidade neste fluxo sequencial, permitindo voltar atrás para eventuais modificações. Veremos

---

4 FONTE: GOMES, A., *Metodologias de Desenvolvimento de Software.* São Paulo, 2010. Disponível em http://www.andreygomes.com/index.php?option=com_content&view=article&id=1:metodologias-de-desenvolvimento-de-software&catid=1:metodologias&Itemid=2. Acessado em 10 set. 2011.

mais adiante modelos que propõem maior flexibilidade no fluxo de execução.

As métricas utilizadas nas estimativas de prazos e recursos humanos são ainda bastante imprecisas e quase sempre o planejamento de atividades precisa ser revisto. Normalmente, os prazos não são cumpridos, pois o planejamento, neste modelo, é feito unicamente nas etapas iniciais do desenvolvimento. A estrutura sequencial e rígida, também não permite que o planejamento seja refeito para corrigir falhas nas atividades de desenvolvimento.

Se for necessário efetuar alguma modificação, voltar os passos de desenvolvimento do projeto é complicado. A metodologia em cascata é extremamente formal, como seria normal de se esperar de uma metodologia cujas raízes encontram-se em regimes militares. Pode-se afirmar que a metodologia em cascata é baseada em documentos e com certeza possui uma enorme quantidade de "entregáveis" e saídas que nada mais são do que documentos. Outra características deste modelo é o alto valor dado ao planejamento. O forte planejamento inicial reduz a necessidade de planejamento contínuo, conforme o andamento do projeto.

A metodologia em cascata funciona bem quando os requisitos do usuário são rígidos e podem ser conhecidos com antecedência. O sistema de navegação do ônibus espacial, por exemplo, pode ser um bom candidato para esta metodologia. Contudo, o foco em documentos para descrever o que os clientes e usuários necessitam podem levar problemas. Muito frequentemente aparecem problemas de comunicação que resultam em um software de baixa qualidade. Os documentos produzidos pelo processo de desenvolvimento podem ser perfeitos, mas o produto real pode ser defeituoso ou inutilizável.

De uma forma ou de outra, muitas das metodologias de desenvolvimento são variações da metodologia de desenvolvimento em cascata – apenas diferenciando-se uma das outras em relação à velocidade, tipos de entregáveis e flexibilidade.

A metodologia de desenvolvimento em cascata pode funcionar bem em ambientes rígidos e fortemente controlados, como por exemplo, os militares, mas possui sérios inconvenientes no cenário comercial. Existem casos em que o contratante do desenvolvimento de software se beneficia pela auditoria imposta pelos métodos do desenvolvimento em cascata. Estes casos incluem projetos que possuem componentes de alto risco, tais como projetos para a área médica ou de segurança pública (NASA[5], 2005).

5 NASA, The standard waterfall model for systems development NASA webpage, archived on Internet Archive March 10, 2005.

3.4 A Metodologia de Desenvolvimento de Entrega por Estágios

Esta metodologia é outra abordagem modificada da metodologia de Desenvolvimento em Cascata. Existe um fluxo definido do início ao fim e as aceitações ocorrem a cada estágio. A diferença principal é que os requisitos de negócio do cliente são quebrados em grandes componentes, e estes componentes são entregues ao cliente em estágios discretos. A figura abaixo demonstra esta metodologia.

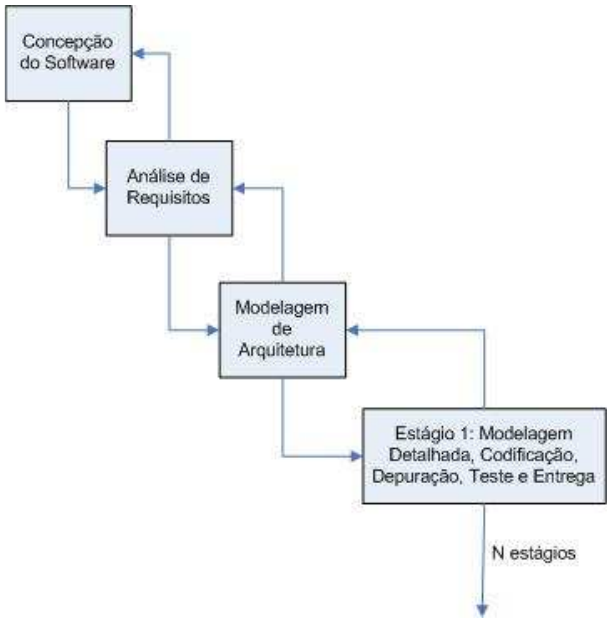

Figura 3: O modelo de entrega por estágios [6]

A utilização da metodologia de Entregas por Estágios, em que a ênfase é dada pelos conjuntos de temas ou funcionalidades, permite que os requisitos mais importantes sejam desenvolvidos primeiro. O cliente obtém um sistema funcional de forma mais rápida. Esta metodologia requer um planejamento cuidadoso e não é muito propício a alterações. Os desenvolvedores precisam entender todas as interdependências entre os componentes e funções que devem ser considerados no planejamento dos estágios. Os riscos são reduzidos para equipes que utilizam entregas por estágios, pois as mesmas são efetuadas em blocos menores entretanto, elas ainda devem ser controladas e monitoradas.

3.5 A Metodologia de Desenvolvimento de Prototipação

O modelo de desenvolvimento baseado na prototipação procura suprir duas grandes limitações do “Modelo Cascata”. A ideia básica deste modelo é que ao invés de manter inalterado os requisitos durante o projeto e codificação, um protótipo é desenvolvido para ajudar no entendimento dos requisitos. Este desenvolvimento passa por um projeto, codificação e teste, sendo que cada uma destas fases não é executada formalmente. Usando assim os protótipos, o cliente pode entender melhor os requisitos do sistema.

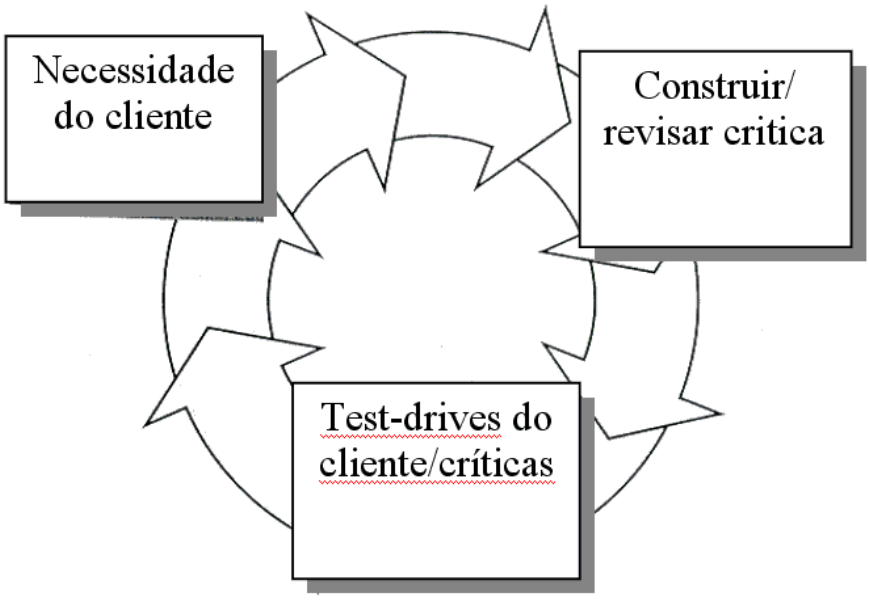

Figura 4: O modelo de prototipação [7]

O protótipo é desenvolvido com uma versão inicial do documento de especificação dos requisitos. Depois do protótipo estar pronto, o cliente o utiliza e com base na sua avaliação do cliente, são fornecidas as impressões do que precisa ser alterado, o que esta faltando e o que não é preciso. O protótipo é então modificado incorporando as sugestões de mudança e o cliente usa o protótipo novamente repetindo o processo até que o mesmo seja válido em termos de custo e tempo. No final os requisitos iniciais são alterados para produzir a especificação final dos requisitos.

Segundo Pressman (PRESSMAN[8], 1997) este modelo pode trazer os seguintes benefícios:

- O modelo é interessante para alguns sistemas de grande porte nos quais representem um

certo grau de dificuldade para exprimir rigorosamente os requisitos

- È possível obter uma versão do que será o sistema com um pequeno investimento inicial
- A experiência de produzir o protótipo pode reduzir o custo das fases posteriores
- A construção do protótipo pode demonstrar a viabilidade do sistema.

Há também, algumas questões a serem consideradas quanto a utilização do modelo:

- A Prototipação deve ser utilizada apenas quando os usuários podem participar ativamente no projeto
- Não descuidar de uma boa análise que deve ser conduzida durante todo o processo de prototipação
- Esclarecer aos usuários que o desempenho apresentado pelo protótipo não necessariamente será o mesmo do sistema final
- Evitar que o sistema final seja um protótipo em que foram implementados todos os requisitos especificados, pois corre-se o risco de ter-se um sistema mal implementado, uma vez que as técnicas utilizadas para desenvolver um protótipo são diferentes daquelas utilizadas na implementação de um sistema (relaxamento de regras de negócio, manipulação de exceções etc)
- Durante a etapa de prototipação, documentar todos os pontos levantados e implementados no protótipo, que não constavam dos requisitos iniciais, para incluí-los na documentação final

3.6 A Metodologia de Prototipação Evolutiva

A metodologia de Prototipagem Evolutiva é uma abordagem que visualiza o desenvolvimento de concepções do sistema conforme o andamento do projeto. Esta metodologia baseia-se na utilização de prototipagem visual ou modelos do sistema final. Estes modelos podem ser simples desenhos, imagens gráficas e até cópias completas em HTML do sistema esperado. Com esta abordagem visual, o cliente tem uma certeza maior do resultado final.

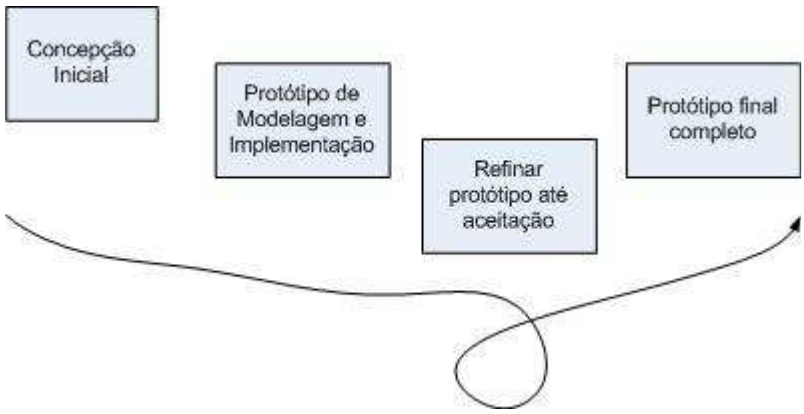

Figura 5: O modelo de prototipação evolutiva [9]

O desenvolvimento real não inicia até que o protótipo final seja aceito. Esta metodologia é até certo ponto bastante flexível a respeito de mudanças de requisitos. Contudo, ela ainda tem necessidade de muito controle. Um ponto negativo desta metodologia é que o planejamento efetivo pode ser difícil e os projetos podem ser reduzidos a um ciclo “Codifica e Corrige” depois que o desenvolvimento é iniciado.

O aspecto de planejamento pode tornar-se um desafio, pois a equipe não tem certeza sobre quanto tempo levará o trabalho. Isto ocorre, em partes, porque o modelo é baseado em interfaces, e não se preocupa tanto com as interdependências mais baixo nível entre os componentes. Os clientes tem uma ideia de como eles querem utilizar o seu novo sistema e utilizam o protótipo como referência. O valor do protótipo, porém, não deve ser superestimado, pois, apesar de tudo, ele não é um software funcional. Existem riscos associados com o planejamento devido a natureza interativa do desenvolvimento, mas eles são atenuados de certa forma pela prototipagem e ciclos de *releases* (versões) menores.

Após o início do desenvolvimento, o gerente de projeto ou líder técnico deve assegurar que os desenvolvedores não utilizem a metodologia “Codifica e Corrige”. Com uma abordagem

fortemente baseada na modelagem de interfaces, a modelagem a nível de componentes pode acabar “caindo pelos lados” quando o cliente dá o “tiro de largada”. A reversão para a metodologia “Codifica e Corrige” é talvez mais um reflexo de mal gerenciamento do que uma fraqueza herdada da prototipagem. Se os desenvolvedores acabarem caindo na metodologia “Codifica e Corrige”, o resultado será uma qualidade difícil de se manter.

3.7 A Metodologia de Desenvolvimento Espiral

O objetivo do modelo espiral é prover um metamodelo que pode acomodar diversos processos específicos. Isto significa que podemos encaixar nele as principais características dos modelos vistos anteriormente, adaptando-os às necessidades específicas de desenvolvedores ou às particularidades do software a ser desenvolvido. Este modelo prevê prototipação, desenvolvimento evolutivo e cíclico, e as principais atividades do "Modelo Cascata".

Sua principal inovação é guiar o processo de desenvolvimento gerado a partir deste metamodelo com base em análise de riscos e planejamento que é realizado durante toda a evolução do desenvolvimento. Riscos são circunstâncias adversas que podem surgir durante o desenvolvimento de software impedindo o processo ou diminuindo a qualidade do produto.

São exemplos de riscos: pessoas que abandonam a equipe de desenvolvimento, ferramentas que não podem ser utilizadas, falhas em equipamentos usados no desenvolvimento ou que serão utilizados no produto final, etc. A identificação e o gerenciamento de riscos é hoje uma atividade importantíssima no desenvolvimento de software devido à imaturidade da área e à falta de conhecimento, técnicas e ferramentas adequadas.

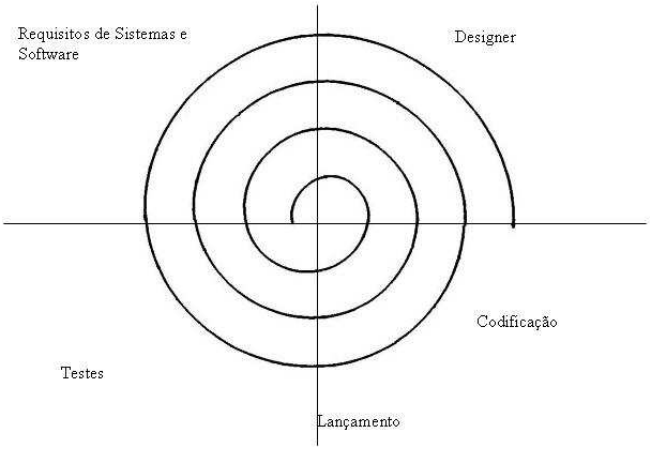

Figura 6: O modelo espiral de desenvolvimento [10]

O modelo espiral descreve um fluxo de atividades cíclico e evolutivo constituído de quatro estágios.

- Estágio 1: devem ser determinados objetivos, soluções alternativas e restrições.
- Estágio 2: devem ser analisados os riscos das decisões do estágio anterior. Durante este estágio podem ser construídos protótipos ou realizar-se simulações do software.

- Estágio 3: consiste nas atividades da fase de desenvolvimento, incluindo design, especificação, codificação e verificação. A principal característica é que a cada especificação que vai surgindo a cada ciclo - especificação de requisitos, do software, da arquitetura, da interface de usuário e dos algoritmos e dados - deve ser feita a verificação apropriadamente.
- Estágio 4: compreende a revisão das etapas anteriores e o planejamento da próxima fase. Neste planejamento, dependendo dos resultados obtidos nos estágios anteriores - decisões, análise de riscos e verificação, pode-se optar por seguir o desenvolvimento em um modelo Cascata (linear), Evolutivo ou Transformação. Por exemplo, se já no primeiro ciclo, os requisitos forem completamente especificados e validados pode-se optar por seguir o modelo Cascata. Caso contrário, pode-se optar pela construção de novos protótipos, incrementando-o, avaliando novos riscos e replanejando o processo (BOEHM[11], 1988).

3.8 A Metodologia de Desenvolvimento Programação Extrema

O método Programação eXtrema (XP, do inglês *eXtreming Programming*) é uma proposta de desenvolvimento ágil e iterativa. O método XP propõe um processo leve, centrado no desenvolvimento iterativo e com a entrega constante de pequenas partes da funcionalidade do software. As partes devem ser incrementadas e requerem a melhoria constante do código (retrabalho).

A possibilidade de entrega rápida do código é um dos fatores de sucesso do XP. Isto no entanto, apenas pode ser feito com o envolvimento constante do cliente que se torna um membro ativo da equipe de desenvolvimento. Esta é uma das características importantes para o método funcionar bem. No entanto, nem sempre o cliente está disponível para a participação ativa.

Uma das características importantes do XP é que não existe um processo de design tradicional com a elaboração de modelos da arquitetura do software. O sistema é concebido a partir de uma metáfora e são descritos em estórias do usuário. Uma metáfora é a transposição de uma conceitualização do mundo real para o sistema a ser desenvolvido. Por exemplo, os programas de correio eletrônico foram construídos utilizando os conceitos de mensagem, caixa de entrada e caixa de saída. Cada mensagem possui remetente, destinatário, assunto e cópias de cortesia (CC).

Este modelo conceitual reflete a forma como correspondências são enviadas nos escritórios e pelo sistema de correio dos Estados Unidos. A metáfora passa a ser fundamental para a elaboração das estórias de usuários.

O uso de cartões CRC (Classes, Responsabilidades e Colaborações) é recomendado de forma a permitir o design em equipe. Cartões CRC permitem a descrição dos conceitos identificados na metáfora na forma de classes. Responsabilidades são identificadas para cada classe. As colaborações determinam as interações entre classes. Os cartões permitem que o todo o time possa colaborar com o design.

3.8.1 Estórias de Usuários

A elaboração de estórias de usuário é uma das atividades fundamentais no XP. As estórias de usuário descrevem cenários com situações de utilização que os envolvidos gostariam que o sistema em desenvolvimento viesse a oferecer. Elas devem ser escritas pelos próprios usuários. As estórias de usuário são semelhantes aos casos de uso da UML, mas não são a mesma coisa. A diferença é que elas não se limitam a descrever o processo de interação do usuário com o sistema.

No XP, as estórias de usuário ocupam o lugar dos longos documentos de requisitos nos

métodos tradicionais. Cada estória deve ser um texto escrito com aproximadamente 3 sentenças.

As estórias de usuários são a base para a criação de estimativas de tempo que serão utilizadas nas reuniões de planejamento de entregas (*releases*). O plano de entregas direciona todo o planejamento do desenvolvimento e do plano de iterações para cada iteração individual. Cada estória deve levar de 2 a 3 semanas para ser implementada em uma iteração individual. Se levar mais do que isso, a estória precisa ser dividida em duas. Um número de 80 estórias aproximadamente é suficiente para definir um plano de entregas.

As estórias de usuário são também a base para a elaboração dos testes de aceitação. Um ou mais testes de aceitação automatizados devem ser criados para verificar se o programa consegue implementar a estória corretamente.

O envolvimento do usuário, portanto, como autor das estórias, é fundamental para a elaboração do plano de entregas e dos testes de aceitação em cada iteração de desenvolvimento.

3.8.2 Práticas de programação

O processo de programação tem como características a programação em pares e a propriedade coletiva do código. Na programação em pares, dois programadores ocupam um único computador.

Embora, a princípio, isto possa ser visto como um desperdício de esforços, várias vantagens podem ser obtidas. Esta estratégia aumenta a qualidade do código e não altera o tempo de entrega, uma vez que haverá um ganho posterior nas etapas de retirada de erros (defeitos).

Os dois programadores atuam de forma colaborativa. Enquanto o que está com o teclado e mouse está pensando diretamente na codificação (sintaxe, variáveis, fluxos de controle, etc.), o outro, pode pensar estrategicamente em como o código irá interagir com outras partes do programa. Além disso, um atua com uma visão mais crítica corrigindo erros dos companheiros e também pode pensar nas futuras mudanças em etapas posteriores. Isto permite que a tradicional prática do “Codifica e Corrige”, criticada nas origens do desenvolvimento de software, possa ser realizada sem perda da qualidade do software.

A propriedade coletiva do código é outra característica fundamental do método XP, com a rotatividade do código entre os programadores e reuniões frequentes para estabilizar o código. A propriedade coletiva encoraja o time todo a contribuir com novas ideias. Qualquer membro do time pode adicionar funcionalidade, corrigir erros ou re-fabricar o código.

Não existe a figura do arquiteto chefe. Todos são responsáveis pelo código inteiro. Não existe alguém para aprovar as mudanças. Reuniões frequentes devem ser definidas como parte do processo iterativo de programação. Estas reuniões devem ser curtas e são realizadas com todos de

pé.

A prática do “Codifica e Corrige” também requer um processo constante de teste de unidade e integração. Para isto funcionar bem, os desenvolvedores devem trabalhar com uma ferramenta de testes automático e um repositório coletivo do código fonte. Os desenvolvedores devem criar testes de unidades e liberar o código apenas quando estiver 100% testado. Uma vez no repositório, qualquer um pode fazer alterações e adicionar novas funcionalidades. Uma ferramenta de controle de versões é fundamental.

O retrabalho ou re-fabricação é uma atividade fundamental e necessária para o XP funcionar. Mesmo que o código esteja 100% testado, para viabilizar reuso, ele precisa ser revisto para remover redundâncias, retirar funcionalidades desnecessárias e modificar arquitetura obsoletas. O retrabalho do código ao longo de todo o ciclo de vida reduz o tempo total de entrega do código e aumenta a qualidade do software produzido.

3.8.3 Regras e Práticas

1. Planejando
    - Estórias do usuário
    - Planejando liberações (*releases*) e pequenas liberações
    - Dividir projetos em iterações (ciclos)
    - Medindo velocidade do projeto
    - Dinâmica de pessoal
    - Reuniões diárias em pé
2. Projetando
    - Simplicidade (não adicione funcionalidades antes do tempo)
    - Metáfora
    - Cartões CRC (Classes, Responsabilidades e Colaboração)
    - Re-fabricar
3. Codificando
    - O cliente deve estar sempre disponível.
    - Programação em pares.
    - Codificar de acordo com padrões acordados.
    - Codificar testes de unidade primeiro.
    - Integrar com frequência.
    - O código é propriedade coletiva.
    - Não atrase.

4. Testando
    - Todo código deve ter os testes de unidade.
    - Cada unidade deve ser testada antes de liberada.
    - Quando um erro é encontrado, testes devem ser realizados.
    - Testes de aceitação devem ser frequentes.

A programação extrema tem sido bem sucedida em projetos pequenos com times pequenos. Relatos mostram que para projetos grandes o método XP não é recomendado (LEITE[12], 2007).

3.9 A Metodologia de Desenvolvimento SCRUM

O SCRUM é um modelo de desenvolvimento ágil de software que fornece métodos para definir o planejamento, os principais papéis de pessoas e a forma de trabalho do time. O nome SCRUM é derivado de uma jogada de rúgbi, em que todo o mesmo time avança como apenas uma unidade, trabalhando com os mesmos jogadores e em conjunto, passando sempre a bola para um e para outro.

A ideia do SCRUM é justamente definir papéis bem específicos para as pessoas envolvidas no projeto e como cada pessoa vai jogar, ou seja, o que cada uma vai ter que fazer para o time seguir em frente no jogo: que no nosso caso é o próprio desenvolvimento do software.

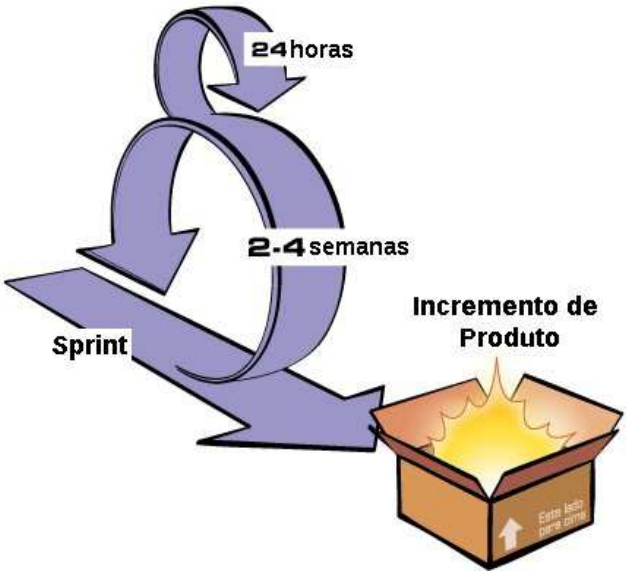

Figura 7: O modelo SCRUM [13]

Em geral e de forma reduzida, esta metodologia funciona da seguinte forma:

1. O produto é definido: quais são os seus requisitos? O que realmente o cliente quer? O responsável por esta tarefa é o que chamamos de Proprietário do Produto (*Product Owner*).
2. O Proprietário do Produto define quais são as funcionalidades do programa que mais importam, criando assim o que chamamos de *Product Backlog*.
3. Com as prioridades definidas, uma pessoa é definida para ser o *ScrumMaster*, uma espécie de coordenador do projeto. O *ScrumMaster*, junto com o Proprietário do Produto e o Time de desenvolvimento definem o que chamamos de *Sprints*.

4. Cada *Sprint* possui uma parte de todo o *Product Backlog*, e devem ser trabalhados de acordo com as prioridades definidas no *Product Backlog*. Os *Sprints* devem ser preparados de uma forma de que durem de 2 a 4 semanas, e que no final de cada período tenham um produto apresentável para o cliente.
5. Os *Sprints* vão sendo feitos até o *Product Backlog* acabar e o Proprietário do Produto definir que o projeto está pronto. Mas isso não quer dizer que novas funcionalidades não podem ser incluídas: basta ir adicionando no *Product Backlog* e realizando outros *Sprints*.

Durante os passos reais de desenvolvimento, os *Sprints*, a principal ferramenta de medição de desempenho é o *Burndown Chart*, que é uma das características mais especiais do SCRUM e que o torna um grande diferencial, no sentido positivo.

Falando mais detalhadamente, o SCRUM tem três partes principais em seu modelo: **Papéis (*Roles*)**, **Cerimônias (*Cerimonies*)** e **Artefatos (*Artifacts*)**. Cada um destes três pilares contém outros sub-itens. Todas estas três partes principais são utilizadas no que chamamos de ciclo de desenvolvimento, ou seja, o *Sprint*. Cada *Sprint* possui suas fases e utiliza destes papéis, cerimônias e artefatos para alcançar seu objetivo final.

3.9.1 Papéis do SCRUM (*Roles*)

Como a metodologia define como o time deve trabalhar, o primeiro passo para o desenvolvimento por SCRUM é definir quem vai fazer o quê. Por isso, chegamos a três entidades principais: o Proprietário do Produto (*Product Owner*), o *ScrumMaster* e o Time.

3.9.1.1 Proprietário do Produto (*Product Owner*)

O Proprietário do Produto representa os interesses do cliente. Ele tem que ser a interface entre o cliente e o time de desenvolvedores, ou seja, estar sempre em contato com o cliente e saber exatamente o que o projeto tem que ter.

Ele tem as seguintes responsabilidades:

- Definir as características e conteúdo do produto;
- Decidir sobre a data de término;
- Ser responsável pela rentabilidade do produto;
- Priorizar as funções de acordo com o valor de mercado e com o cliente;
- Ajustar recursos e priorizar tarefas a cada 30 dias, como necessário;
- Aceitar ou rejeitar o resultado do trabalho.

O trabalho mais árduo do Proprietário do Produto é definir o *Product Backlog*, um dos dois artefatos do SCRUM, que veremos posteriormente. Essa definição se dá durante o

Planejamento, que também veremos adiante.

3.9.1.2 ScrumMaster

O *ScrumMaster* é o líder da equipe de desenvolvimento e durante o trabalho, fica mais em contato com o Proprietário do Produto. Ele gerencia e repassa o trabalho que foi decidido durante o planejamento pelo Proprietário do Produto. Ele deve:

- Assegurar que a equipe de desenvolvimento funcione plenamente e seja produtiva;
- Ajudar na cooperação entre todas as funções e papéis do time;
- Remover barreiras;
- Proteger a equipe de interferências externas;
- Assegurar-se de que a metodologia está sendo seguida, incluindo chamadas para reuniões diárias (*Daily Scrum Meetings*), revisões de atividade (*Sprint Reviews*) e reuniões de planejamento das atividades (*Sprint Planning*).

Devido a todas estas responsabilidades, o *ScrumMaster* é o que podemos chamar de Coordenador do Projeto. Ele facilita a comunicação entre as pessoas e faz o projeto andar de verdade.

Além de comandar as reuniões diárias, o *ScrumMaster* têm três principais responsabilidades:

1. Precisa saber quais atividades foram concluídas, quais foram iniciadas, quaisquer novas tarefas que foram descobertas e qualquer estimativa que possa ter mudado. Isto permite a ele atualizar sempre o *Burndown Chart*, que permite mostrar quanto trabalho falta para um *Sprint* acabar, dia por dia. Ele também tem sempre que olhar cuidadosamente para o número de tarefas em aberto. Estes trabalhos em aberto devem ser minimizados o máximo possível para garantir um trabalho sempre limpo e eficiente.
2. Deve avaliar as dependências superficiais e bloqueios que causam prejuízos ao andamento do Projeto. Estes itens devem receber prioridades e serem acompanhados. Um plano de solução deve ser implementado de acordo com a ordem de prioridade destes impedimentos. Alguns podem ser resolvidos pelo time, outros podem ser resolvidos por meio de vários times, e outros podem precisar de envolvimento da gerência, pois, podem ser problemas relacionados à empresa que bloqueiam qualquer membro do time a fazer qualquer coisa. Como exemplo, deste tipo de impedimento externo, temos as questões judiciais.
3. O *ScrumMaster* deve perceber e resolver problemas pessoais ou conflitos entre os integrantes do time de desenvolvimento SCRUM. Este tipo de problema deve ser clarificado pelo *ScrumMaster* e resolvido por diálogo com o time, ou então, o *ScrumMaster* terá que ter

ajuda da gerência ou do departamento de Recursos Humanos. Alguns estudos apontam que 50% dos problemas de desenvolvimento acontecem por razões pessoais. O *ScrumMaster* precisa estar sempre atento ao time para fazer dele totalmente funcional e produtivo.

3.9.1.3 A Equipe de Desenvolvimento

A equipe de desenvolvimento é quem vai trabalhar no código de programação para que o software comece a ter as funcionalidades implementadas. Pode haver uma ou mais equipes de desenvolvimentos, dependendo da complexidade do software.

Algumas características das equipes de desenvolvimento:

- São pequenas e multidisciplinares, com média de 7 participantes;
- Definem metas de cada *Sprint*, junto ao *ScrumMaster*, e especificam seus resultados de trabalho;
- Têm o direito de fazer tudo dentro dos limites das diretrizes do projeto para atingir a meta de cada *Sprint*;
- Organizam os trabalhos para atingir os objetivos dos *Sprints*;
- Trabalham para atingir todos os resultados definidos pelo Proprietário do Produto.

3.9.2 Cerimônias SCRUM (*Cerimonies*)

As cerimônias SCRUM são eventos que acontecem dentro de um ciclo de desenvolvimento utilizando esta metodologia. Existem três tipos de cerimônias SCRUM: a reunião de Planejamento do *Sprint*, as reuniões diárias SCRUM e a reunião de revisão do *Sprint*. Estes três tipos de evento caracterizam bem o ciclo de vida de cada *Sprint*: início, meio e fim.

3.9.2.1 Reunião de Planejamento do Sprint

Como dito anteriormente em resumo, o primeiro passo de um projeto SCRUM é quando o Proprietário do Produto desenvolve um plano para o projeto de software. Neste plano, o Proprietário do Produto trabalha bem próximo ao cliente para definir todas as funcionalidades que o cliente quer no seu produto. A partir desta lista, ele também tem que definir as prioridades de cada funcionalidade de acordo com várias variáveis: valor de mercado, importância de base, importância para o cliente, entre outras. Esta lista final é o que chamamos de *Product Backlog* e é a base para a reunião de planejamento do *Sprint*.

Nesta reunião, o *ScrumMaster* trabalha junto com o Proprietário do Produto e a Equipe de Desenvolvimento para definir qual a carga de tempo que cada funcionalidade do *Product Backlog* terá. Isto ajuda o Proprietário do Produto a definir prazos reais para o projeto e o habilita a

poder verificar como está o andamento do projeto durante todo o período de desenvolvimento. Esta carga de tempo e esforço é definida de acordo com o tamanho do(s) time(s), horas disponíveis e o nível de produtividade do time.

Quando as prioridades e prazos das funcionalidades do software são definidas por completo, o Proprietário do Produto sai de cena e o *ScrumMaster* começa a trabalhar juntamente com a Equipe de Desenvolvimento para a quebra destas tarefas grandes em pequenas tarefas, divididas por todos os integrantes da equipe de desenvolvimento de acordo com suas especialidades. Esta reunião de planejamento geralmente dura até 4 horas e é ela quem define o *Sprint Backlog*, um dos artefatos SCRUM.

Uma vez definido o *Sprint Backlog*, com a listagem de todas as tarefas, seus responsáveis e seus prazos, o processo de desenvolvimento começa.

3.9.2.2 Reuniões Diárias SCRUM

Uma das principais características deste modelo de desenvolvimento ágil são as reuniões diárias SCRUM, onde o *ScrumMaster* se reúne com a equipe de desenvolvimento para saber como anda o projeto. Esta reunião acontece todo dia durante o ciclo de desenvolvimento (*Sprint*) e tem uma duração curta de aproximadamente 15 minutos.

Durante a reunião, cada um dos membros responde as seguintes três perguntas:

1. O que fiz ontem?
2. O que farei hoje?
3. Quais impedimentos e dificuldades apareceram no caminho?

O *ScrumMaster* tem um papel muito importante nestas reuniões: ele terá que identificar todos os problemas ou novas tarefas que surgirem e planejar como estas aparições serão resolvidas. Além do mais, ele terá assim uma visão completa do projeto e poderá gerenciar todas as sub tarefas de cada *Sprint*, preenchendo assim o *Burndown Chart*.

3.9.2.3 Reunião de Revisão do Sprint

No final do período do *Sprint*, a reunião de revisão do *Sprint* é feita. Nesta reunião, a equipe de desenvolvimento, junto ao *ScrumMaster*, se reúne com o Proprietário do Produto (e convidados, caso ele achar adequado, como por exemplo, o cliente ou acionistas do projeto). Esta reunião dura aproximadamente 4 horas.

Na primeira parte da reunião, o resultado do *Sprint* é apresentado para o Proprietário do Produto, ou seja, tudo que foi desenvolvido durante o ciclo de desenvolvimento. As funcionalidades são revisadas e o valor do produto é definido. Depois de revisar todo este resultado, o Proprietário

do Produto define quais itens do *Product Backlog* foram completados no *Sprint*, discutindo então com os membros da equipe e o cliente quais serão as novas prioridades. Este é o primeiro passo para criar um novo *Sprint*, caso seja necessário, pois dessa primeira parte da reunião, um novo *Product Backlog* é gerado.

Na segunda parte da reunião, o *ScrumMaster* se reúne com a equipe de desenvolvimento e revisa os altos e baixos do ciclo. O time conversa para saber o que foi bom e como se pode melhorar ainda mais o ambiente, as ferramentas e o convívio entre o time e suas funções. Esta parte é basicamente um aprimoramento interno.

Depois que esta reunião é finalizada, um novo ciclo *Sprint* pode começar até que todo o produto seja implementado/finalizado e o *Product Backlog* esteja limpo de pendências.

3.9.3 Artefatos SCRUM (*Artifacts*)

Os artefatos SCRUM são as ferramentas básicas para se trabalhar com este modelo de desenvolvimento. Estes artefatos servem como guias e indicadores durante o processo. São divididos em três: *Product Backlog*, *Sprint* Backlog e Burndown Chart.

3.9.3.1 Product Backlog

Como visto anteriormente, no início do projeto, o Proprietário do Produto prepara uma lista de funcionalidades desenvolvidas em conjunto com o cliente, que é organizada por prioridade de entrega. Essa lista chama-se *Product Backlog*. A equipe SCRUM contribui para o *Product Backlog* estimando o custo do desenvolvimento das funcionalidades.

O *Product Backlog* deve incluir todas as funcionalidades visíveis ao cliente, mas também, os requisitos técnicos para poder desenvolver o produto, e em torno de dez dias de desenvolvimento esses itens deverão estar prontamente definidos para o seu desenvolvimento começar. As tarefas que tem mais prioridade e complexidade são quebradas em sub-tarefas para poderem ser estimadas e testadas.

3.9.3.2 Sprint Backlog

Assim que a equipe de SCRUM escolher e definir a lista de requisitos e a prioridade de seus itens do *Product Backlog*, cada um desses itens agora será dividido em partes menores para o *Sprint Backlog*. Ou seja, uma lista especificando os passos necessários para implementar uma funcionalidade do sistema.

Logo após o *Sprint Backlog* estar concluído, o total de horas trabalhadas é comparado com o total previsto anteriormente no *Product Backlog*. Caso haja uma diferença significativa, a

equipe SCRUM deve negociar novamente com o cliente o número correto de horas, para que o *Sprint* seja realizado com maior probabilidade de sucesso.

3.9.3.3 Burndown Chart

O *Burndown Chart* é um gráfico que mostra a quantidade de trabalho cumulativo restante de um *Sprint*, dia por dia. Neste gráfico, a altura indica a quantidade de tarefas do *Sprint Backlog* não completadas, e o comprimento são os dias. Com isto, podemos visualizar facilmente se um trabalho está tendo progresso, completando as tarefas, enquanto vemos que as colunas do gráfico vão caindo em sua altura. Se um ciclo de desenvolvimento, ou *Sprint*, tem a duração de 30 dias, haverão 30 colunas juntas. As colunas têm que diminuir ao longo do comprimento e antes ou até o 30o. dia não poderá haver colunas, indicando que todos as tarefas foram completadas e o *Sprint* foi um sucesso.

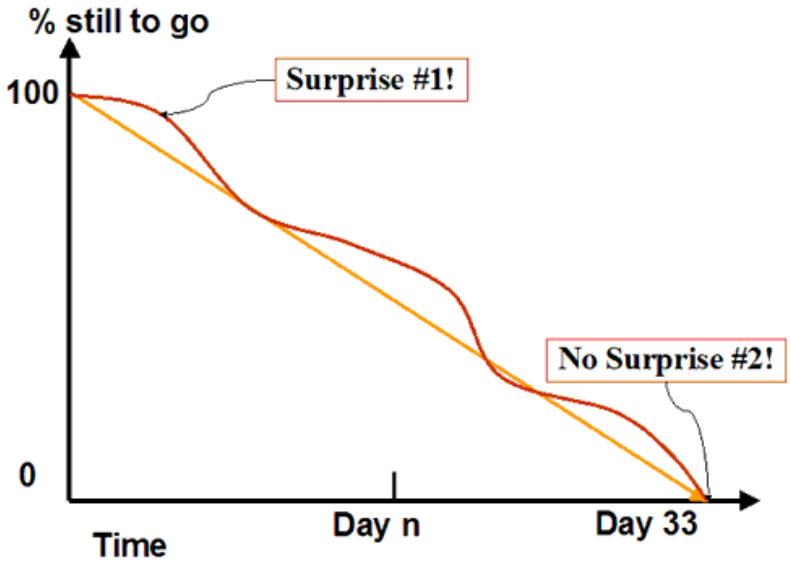

Figura 8: Exemplo de Burndown chart [14]

Durante as reuniões diárias do *Sprint*, o *ScrumMaster* acompanha os membros da equipe para ver o que foi terminado. Assim, dia a dia, ele ajusta o *Burndown Chart* e a qualquer momento todo o time pode ter uma ideia de como o processo está andando. O mesmo gráfico pode ser mostrado para o Proprietário do Produto ver que está tudo indo bem.

Por essas razões, o *Burndown Chart* é um dos principais recursos de medição do processo de desenvolvimento e um diferencial para a metodologia SCRUM.

3.10 A Metodologia de Desenvolvimento RUP

A metodologia RUP (*Rational Unified Process*) é um processo que fornece uma metodologia disciplinada de desenvolvimento utilizando um conjunto de ferramentas, modelos e entregáveis. A RUP se diferencia das demais metodologias por ser proprietária de uma empresa, no caso a *IBM Rational*.

A metodologia padronizada pode ser atrativa para grandes organizações que necessitem de uma linguagem comum e ferramentas padronizadas para toda a organização. A metodologia RUP utiliza UML para comunicar os requisitos, arquiteturas e modelagens.

Figura 9: O modelo RUP [15]

A metologia RUP é uma metologia de desenvolvimento interativo com foco na redução dos riscos do projeto. Ela agrega um valor real à organização que necessita manter padrões tanto para as comunicações externas quanto para a comunicação com a equipe de desenvolvimento. Algumas características "negativas" desta metodologia são o aumento dos requisitos de documentação, a necessidade dos clientes possuírem algum entendimento de UML e ao fato de estar amarrada ao software proprietário da *IBM Rationa*l.

Um dos principais pilares do RUP é o conceito de *best practices* (melhores práticas), que são regras/práticas que visam reduzir o risco e tornar o desenvolvimento mais eficiente. O RUP define seis *best practices*, sendo elas:

- Desenvolver iterativamente
- Gerenciar requerimentos
- Utilizar arquiteturas baseadas em componentes
- Modelar visualmente

- Verificação contínua de qualidade
- Controle de mudanças

O RUP, ainda, entrelaça o conceito de *best practices* em quatro definições, sendo elas:

- Funções: grupos de atividades executadas.
- Disciplinas: áreas de esforço na engenharia de software.
- Atividades: definições de como (objetos/artefatos) é construído e
- Objetos/Artefatos: resultado do trabalho, produzido ou modificado durante o processo.

Além destas definições, esta metodologia de desenvolvimento divide o processo de desenvolvimento de software em quatro fases:

- Concepção: definição do escopo do projeto.
- Elaboração: elaboração básica do software.
- Construção: desenvolvimento.
- Transição: implantação.

3.11 RUP – Best Practices

O RUP tenta diminuir os riscos do desenvolvimento efetivamente deixar o desenvolvimento mais eficiente, por meio de seis práticas básicas a serem executadas durante todo o processo de desenvolvimento.

3.11.1 Desenvolver Iterativamente

Desenvolver iterativamente significa desenvolver em ciclos. Cada ciclo contém um objetivo que deve ser alcançado como lançamento de um pré *release* ou beta, correção de um erro, etc.

Esta prática acaba dando ao RUP uma série de vantagens, como a possibilidade de identificar/modificar requerimentos com mais facilidade, integração progressiva (quase contínua) de elementos ao software, ocasionando uma melhora no descobrimento e endereçamento de riscos; desenvolvimento iterativo provê aos gerentes maneiras de fazer mudanças táticas aos produtos; etc.

3.11.2 Gerenciamento de Requerimentos

Gerenciamento de requerimentos provê uma maneira prática de produzir, organizar, comunicar e organizar os requerimentos de um projeto. Adicionalmente, os casos de uso e cenários descritos nos processos são uma excelente forma de capturar e assegurar requisitos. O gerenciamento de recursos acarreta um melhor controle sobre projetos complexos, além de maior

qualidade e redução de custos.

O RUP é uma metodologia dirigida a casos de uso, de modo que é possível utilizar os mesmos casos de uso definidos no sistema como base para o resto do processo.

3.11.3 Utilizar Arquiteturas Baseadas em Componentes

Foca o desenvolvimento na modularização, por meio do uso de componentes, de modo a criar um sistema flexível, adaptável, intuitivamente entendível e reutilizável. O RUP define componentes como módulos não triviais e/ou subsistemas com uma função clara e específica. Entre os benefícios podemos citar a facilidade para identificar, isolar, manipular e desenvolver componentes é maior do que para um sistema inteiro; componentes podem ser desenvolvidos com a reutilização em mente; etc.

3.11.4 Modelar Visualmente

A modelagem visual permite melhor entender não só a concepção e a complexidade do sistema, mas também “dimensionar”, além de facilitar a identificação e solução de problemas.

3.11.5 Verificação Contínua de Qualidade

O RUP não toma a qualidade como responsabilidade de apenas uma pessoa ou grupo, mas como uma responsabilidade de todos os integrantes do projeto. A qualidade é focada especialmente em duas áreas:

- Qualidade de produto: a qualidade do produto sendo desenvolvido (software ou sistema) e todos as partes envolvidas (componentes, subsistemas, arquitetura, etc).
- Qualidade de processo: a qualidade dos processos dentro do projeto de desenvolvimento.

3.11.6 Controle de Mudanças

Como resultado de um processo de desenvolvimento iterativo, muitas são as mudanças ocorridas no decorrer do projeto. Controlar as mudanças durante todo o projeto é prática fundamental para manter a qualidade do projeto.

3.12 Fases de Desenvolvimento

O processo de desenvolvimento é dividido em ciclos, sendo que o ciclo de desenvolvimento é subdividido em 4 fases consecutivas.

Estas fases são concluídas tão logo uma *milestone* é alcançada. Uma *milestone* define uma etapa, na fase, na qual decisões críticas são feitas ou objetivos são alcançados.

3.12.1 Concepção

Concepção inicial do sistema, aonde é feita uma discussão sobre o problema, definição do escopo do projeto, estimativa de recursos necessários para a execução do projeto, etc. É nesta fase que é apresentado o plano de projeto, caso de uso inicial e o glossário do projeto, entre outros.

3.12.2 Elaboração

O propósito desta fase é analisar o domínio do problema, desenvolver o plano de projeto, estabelecer a fundação arquitetural e eliminar os elementos de alto risco. Os elementos de risco a serem analisados, nesta fase, são os riscos de requerimentos, tecnológicos (referentes a capacidade das ferramentas disponíveis), de habilidades (dos integrantes do projeto) e esta é a fase mais crítica de todas, pois ao final desta fase a engenharia é considerada completa e os custos para modificação do sistema aumentam a medida que o projeto avança. Do ponto de vista administrativo, é ao final desta fase que um projeto deixa de ser uma operação de baixo risco e baixo custo para se tornar uma operação de alto risco e alto custo.

3.12.3 Construção

Esta fase compreende a fase de modelagem e a fase de desenvolvimento em si, aquela em que o sistema é efetivamente programado. A fase de modelagem deve utilizar alguma notação definida pela UML.

3.12.4 Transição

A partir desta fase, o sistema já está pronto, começa a implantação do sistema para o usuário. Nesta fase deve ser utilizado o lançamento de versões beta, operação paralela com o sistema legado, treinamento dos usuários e mantenedores do sistema.[16]

4 A empresa e o cenário inicial

Em relação a metodologias, é conhecido que estas são capazes de potencializar e estruturar um ambiente de desenvolvimento, fazendo com que haja uma maior eficiência na entrega e na qualidade dos projetos. Porém, quando se fala em adotar uma metodologia nova, ou começar a usufruir de uma metodologia, há uma certa resistência e um certo impacto com esse acontecimento.

É exatamente nessa situação que este estudo se encontra. Ao analisar a situação da empresa, viu-se que a mesma possui um porte entre pequeno, porém com um crescimento grande nos últimos meses, devido o sucesso da plataforma desenvolvida por eles.

4.1 A empresa TEx Soluções em Informática

A TEx Soluções em Informática, é uma empresa de desenvolvimento situada no bairro de Moema, na zona sul de São Paulo. É mantida por quatro sócios, e possui 11 funcionários, sendo que no primeiro momento há uma pessoa responsável pelo suporte e atendimento e 10 desenvolvedores que tratam do sistema.

O sistema, chamado TELEPORT, é um sistema voltado para corretores de seguros. A princípio, um corretor para calcular uma cotação precisa pegar as informações do objeto em questão (um veículo, um local de risco) e inseri-las no kit da seguradora (o software que é responsável por calcular o seguro).

Como são várias seguradoras e vários kits, para efetuar uma comparação, o corretor deve abrir um kit, inserir as informações, efetuar o calculo, e repetir isso em todos os outros kits que o cliente desejar, gerando uma repetição do trabalho.

O sistema TELEPORT foi criado pensando em suprir a ausência de uma ferramenta multi cálculo de seguros totalmente Web. Utilizando da suíte Delphi, os desenvolvedores constroem páginas da web pelas telas construídas por meio do componente *IntraWeb VCL*, responsável por converter os objetos, classes, procedimentos e funções criadas no Delphi em código HTML e JavaScript.

Com apenas três anos de vida, a empresa começou a ter um grande crescimento, visto que poucos concorrentes forneciam um sistema deste porte totalmente web. A maioria, era um sistema que rodava na máquina dos corretores, sendo restrito à sua maioria apenas em clientes com Windows.

Com esse crescimento, aumentaram também as demandas de desenvolvimento e de suporte. Para suportar isso era preciso uma mudança de postura interna em relação a maneira de como é feita o desenvolvimento, e como é prestado o suporte.

Na situação inicial, a empresa não usufruía de uma metodologia, mas sim algo como uma mistura das chamadas metodologias “Codifica e Corrige”, e “Modelo Balbúrdia”, sendo as tarefas delegadas pelo gerente de desenvolvimento, sendo ele responsável pelo teste, validações, controle de versões e controle de prazos. Enfim, tudo de maneira manual e com prazos bem restritos para a execução das tarefas.

Isso gerava uma série de transtornos e estresse, visto que os prazos curtos não davam oportunidade do software ter uma rotina de testes adequadas, tampouco um ciclo de vida, visto que o sistema a todo momento ganhava novas funcionalidades e correções.

Em relação ao suporte ao sistema, a estrutura também não era provida de estrutura, pois apenas uma pessoa era responsável pelo atendimento on-line, atendimento pelo telefone e controle de chamados abertos, gerando assim, atrasos na entrega de correções e atendimentos às necessidades específicas dos clientes.

5 A proposta da estruturação da empresa

A empresa passava por um momento crítico, pois haviam vários projetos atrasados, o sistema possuía vários erros de funcionamento e o suporte não estava saturado, não conseguindo ter um andamento do fluxo de resolução de chamados. Tal desorganização é devido a um controle praticamente manual e nenhuma adoção de metodologia ou ferramenta.

Em um certo momento, propus juntamente com outro desenvolvedor, a adoção de uma metodologia e ferramentas gerenciais que auxiliam a administração do suporte e do desenvolvimento do sistema.

As mudanças foram propostas visando uma melhoria no andamento dos projetos e do suporte, se consistiam na adoção de uma metodologia e de ferramentas, após conversa sobre a situação atual da empresa e como ela poderia se encontrar com a adoção desses recursos.

5.1 Primeiras reuniões e apresentações

Com uma certa resistência, a primeira reunião foi constituída de uma introdução às ferramentas e às metodologias disponíveis e como elas funcionaram. Foram apresentadas as ferramentas Mantis e Tortoise SVN. A primeira ferramenta gerencia e executa rastreamento de chamados e *bugs*. A segunda, é uma ferramenta que serve como cliente de um servidor de versões, juntamente proposto na reunião.

5.2 Mantis Bug Tracker

Para o atendimento aos chamados do suporte, feito manualmente em uma planilha do Excel, foi proposto a adoção de uma ferramenta de gerenciamento de chamados.

Mantis Bug Tracker é um software Web baseado em PHP, e é um software livre que executa um rastreamento de *bugs* e chamados de suporte, podendo ser instalado em um servidor Windows, Linux, Mac OS, OS/2 e outros. É compatível com os bancos MySQL e PostgreSQL.

5.3 Servidor de Versões

O controle das versões do sistema é totalmente manual, sendo controlado apenas por

uma pasta com a data da entrega da tela alterada, em uma pasta compartilhada. Qualquer tela do sistema que é alterada, passa pela revisão dessa mesma pessoa, e as versões novas não passam por uma rotina de testes efetiva.

Para obter um controle mais eficiente, foi proposto a montagem de um servidor de versões SVN rodando em Linux, e o software cliente, chamado Tortoise SVN.

Como as máquinas têm configuração de ambiente Windows, foi proposto a adoção dessa ferramenta devido a fácil integração da ferramenta com o sistema operacional, e pela fácil configuração desta ferramenta.

5.4 A adoção da metodologia e o impacto causado

Após a apresentação das diversas metodologias e ferramentas, foi concluído que seria utilizada uma adaptação da metodologia Programação Extrema, tendo a execução do desenvolvimento sendo acompanhada de perto por um representante do cliente, visto que a maioria das tarefas eram relacionadas a uma lista de melhorias ou necessidades específicas do cliente.

Em resumo, um representante do cliente começa a fazer parte da equipe, tendo uma participação ativa na construção do sistema. Em três casos essa situação foi adotada, tendo em vista a resolução de uma lista de pendências ou melhorias em relação ao uso do sistema, apelidado de “Backlog”.

Este backlog contém todos os prazos, descrição da tarefa e qual o responsável pelo andamento da mesma. O cliente por sua vez, fica ao lado do desenvolvedor, dando suporte nas dúvidas sobre as novas funcionalidades e sobre as necessidades que possui.

5.4.1 O primeiro Backlog

O primeiro cliente a adotar esta metodologia foi a corretora Brookfield, com o principal escritório situado no Rio de Janeiro. A representante era uma corretora chamada Andréa, responsável por dar suporte aos desenvolvedores sobre as necessidades da empresa.

Em um primeiro momento, a cliente foi alocada em uma mesa ao lado de um dos três desenvolvedores envolvidos. Foram desenvolvidas novas funcionalidades voltadas à corretora. Após esse primeiro estágio, dois desenvolvedores foram alocados para a sede da corretora no Rio de Janeiro para entregar a fase final do projeto. Porém, como primeira experiência, houve um pequeno atraso na entrega do projeto. Mas mesmo assim, foi notada uma grande melhora na execução do projeto como um todo.

Em níveis não mensuráveis, notou-se a satisfação do cliente tendo em vista que as pendências foram resolvidas, embora tenha havido um pequeno atraso.

5.4.2 O segundo Backlog

Passado um mês, houve um segundo backlog, voltado à corretora Vertex, que está locada em vários pontos ao longo do estado de Minas Gerais. Nesse caso, o desenvolvimento foi mais voltado junto com os diretores de ambas as empresas, cuidando das especificações e repassando aos desenvolvedores.

Então, além do projeto ter sido executado no prazo estipulado de 3 meses, notou-se uma

relevante evolução com a simples adoção de uma metodologia na rotina de trabalho de desenvolvimento. Isso demonstra o quanto uma metodologia pode contribuir para melhorar o ritmo e a qualidade dos processos de trabalho.

Como esta corretora possui uma base de dados muito volumosa, conseguiu-se executar uma série de otimizações, pois o sistema era todo carregado com *query's* imensas a ponto de selecionar praticamente todas as colunas de várias tabelas, por exemplo. A maior parte das melhorias incidia no desempenho, o que acabou afetando o sistema como um todo, deixando-o mais rápido.

5.4.3 O terceiro backlog

Por fim, foi executado um terceiro e último backlog até o presente momento, foi voltado para a corretora Economize no Seguro, situada na zona Leste de São Paulo. Esta por sua vez, tinha uma necessidade de diversos ajustes, semelhante ao que havia acontecido com a corretora Brookfield, porém, voltados à parte financeira.

Já com a experiência das outras ocorrências, esta situação procedeu dentro do planejado, tendo a maioria das implementações executadas voltadas para todas as outras corretoras.

5.5 A organização do suporte

Durante a adoção da metodologia no desenvolvimento, houve também uma mudança na área de suporte da empresa, utilizando-se de uma ferramenta de organização de chamados, chamada ZenDesk. O que aumentou a produtividade e organização da área de suporte, tendo uma melhora em 60% no tempo de resposta aos chamados abertos.

6 Considerações Finais

Após a visível melhoria nas áreas de desenvolvimento e suporte, tendo uma agilidade na execução das diversas tarefas, e também na satisfação notável nos clientes, conclui-se que a adoção de metodologias em um ambiente de desenvolvimento, só tende a evoluir.

Inicialmente, é comum ocorrer uma certa resistência para mudar a filosofia de trabalho que se prolongou durante anos. Porém, essa mudança gera uma nova perspectiva de trabalho, trazendo novos meios de desenvolver o trabalho, proporcionando diversas melhorias e gerando menos estresse para ambos todos.

De um lado, os clientes e usuários, que não só dispões de um serviço com maior qualidade, como dispõem também de um suporte mais estruturado para atendê-los. E de outro, os profissionais de informática, que ganham com um melhor ambiente para exercerem suas tarefas de desenvolvimento, e também com um prazo mais bem aproveitado.

7 Referências Bibliográficas

POMPILHO, S., *Analise Essencial: guia prático de análise de sistemas*. Rio de Janeiro, Ciência Moderna, 2002.

REZENDE, D. A., *Engenharia de Software e Sistemas de Informação*. 2. ed. , Rio de Janeiro, Brasport, 2002.

WEBSTERS *Ninth Neiv Collegiate Dictionary*. 1998.

MCCONNELL, S. CODE COMPLETE. *Um guia prático para a construção de software*. 2. ed. São Paulo: Bookman, 2005.

NASA, *The standard waterfall model for systems development NASA webpage*, archived on Internet Archive March 10, 2005.

PARNAS, D., *A rational design process and how to fake it* (PDF) .

SOMMERVILLE, I., *Software engineering*. 5th. ed. Addison-Wesley, 1995.

PRESSMAN, R. S., *Software engineering: A practitioner's approach*. 4th. ed. McGraw-Hill, 1997. p. 22-53.

FALBO, R. A., *Integração de Conhecimento em um Ambiente de Desenvolvimento de Software*. Tese de Doutorado, COPPE/UFRJ, Rio de Janeiro, Brasil, 1998.

HUMPHREY, W. S., M*anaging the Software Process. Addison.* Wesley Publishing,Company, Massachussets, 1990.

SCHWARTZ, J. I. ,Construction of software. In: *Practical Strategies for Developing Large Systems*. Menlo Park: Addison-Wesley, 1st. ed., 1975.

REIS, C., *Caracterização de um Modelo de Processo para Projetos de Software Livre.* Tese de mestrado. Instituto de Ciências Matemática e Computação. São Carlos, São Paulo, Abril de 2001

SASS, G. G., *O Processo de Desenvolvimento Baseado em Componentes: O impulso das novas tecnologias*, São Paulo, 19 de agosto de 2005.

BOEHM, B. *A Spiral Model of Software Development and Enhancement*. IEEE Computer, vol.21, 5, May 1988, pp 61-72

LEITE, J. C. *Programação Extrema (XP). Engenharia de Software*, Natal, março. 2007. Disponível em: http://engenhariadesoftware.blogspot.com/2007/03/programao-extrema-xp.html/, Acessado em 10 set. 2011.

SCRUM Alliance, Disponível em: http://www.scrumalliance.org/, Acessado em 10 set. 2011.

AGILE Alliance, Disponível em: http://agilealliance.org/article/articles_by_category/17 , Acessado em 10 set. 2011.

CODE Project, Disponível em: http://codeproject.com/KB/architecture/scrum.aspx, Acessado em 10 set. 2011.

QUADROS, M., *Gerencia de Projetos de Softwares: Técnicas e Ferramentas*, Visual Books.

IBM, *Rational Software - Best Practices for Software Development Teams*, Disponível em: http://www.rational.com/, Acessado em 10 set. 2011.

IBM, *Rational Software - Rationa Unified Process*, Disponível em: http://www.rational.com/, Acessado em 10 set. 2011.

IBM, *The Rational Edge - RUP and XP, Part I - Finding Common Ground*, Disponível em: http://therationaledge.com,