UNIVERSIDADE DO SUL DE SANTA CATARINA

CARLOS EDUARDO POLEGATO

MODELAGEM ÁGIL SCRUM:
ADEQUANDO EM BUSCA DA MELHORIA CONTÍNUA

Florianópolis

2012

CARLOS EDUARDO POLEGATO

MODELAGEM ÁGIL SCRUM:
ADEQUANDO EM BUSCA DA MELHORIA CONTÍNUA

Monografia apresentada ao curso de especialização em Engenharia de Projetos de Software, da Universidade do Sul de Santa Catarina, como requisito parcial para a obtenção do título de Especialista em Engenharia de Projetos de Software.

Orientadora: Prof. Vera Schuhmacher, MEng.

Florianópolis

2012

CARLOS EDUARDO POLEGATO

MODELAGEM ÁGIL SCRUM:
ADEQUANDO EM BUSCA DA MELHORIA CONTÍNUA

Este Trabalho de Conclusão de Curso foi julgado adequado à obtenção do título de Especialista em Engenharia de Projetos de Software e aprovado em sua forma final pelo Curso de Especialização em Engenharia de Projetos de Software da Universidade do Sul de Santa Catarina.

Florianópolis, 01 de março de 2012.

____

Profa. e orientadora Vera Schuhmacher, MEng.
Universidade do Sul de Santa Catarina

TERMO DE ISENÇÃO DE RESPONSABILIDADE

MODELAGEM ÁGIL SCRUM:
ADEQUANDO EM BUSCA DA MELHORIA CONTÍNUA

Declaro, para todos os fins de direito, que assumo total responsabilidade pelo aporte ideológico e referencial conferido ao presente trabalho, isentando a Universidade do Sul de Santa Catarina, a Coordenação do Curso de Especialização em Engenharia de Projetos de Software, a Banca Examinadora e o Orientador de todo e qualquer reflexo acerca desta monografia.

Estou ciente de que poderei responder administrativa, civil e criminalmente em caso de plágio comprovado do trabalho monográfico.

Florianópolis, 01 de março de 2012.

____

CARLOS EDUARDO POLEGATO

À minha namorada Juliana Tessari, aos meus pais Francisco Carlos Polegatto e Jicelma Pereira Vasconcelos Polegatto.

AGRADECIMENTOS

Agradeço à Deus, à minha namorada que compreendeu os finais de semana e noites dedicadas ao estudo e pela força nos momentos decisivos, à minha família que me apoiou e incentivou desde o início, aos amigos de trabalho que colaboraram com a pesquisa e à empresa por permitir que a equipe fosse exposta e alvo principal de minha pesquisa.

RESUMO

Neste trabalho monográfico é apresentado um estudo envolvendo técnicas e artefatos de diferentes modelos de engenharia de software, bem como sua aplicação para corrigir pontos negativos no modelo utilizado por uma empresa tradicional. Nesta pesquisa, realizou-se um estudo dos principais modelos ágeis e tradicionais utilizados pelo mercado de desenvolvimento de software, suas características, aplicabilidade, técnicas e artefatos. Identificado as principais técnicas e artefatos dos modelos, foi realizado um estudo em uma equipe específica, identificado seus pontos falhos e sugerido o melhor encontrado em cada modelo, de acordo com os fatores negativos encontrado no processo de desenvolvimento de software da equipe.

Palavras-chave: Modelagem ágil. Scrum. Programação Extrema. Qualidade de Software. Produtividade.

ABSTRACT

This monograph presents a study of techniques and artifacts of different models of software engineering and its application to correct the negative points in the model used by a traditional company. In this research, we carried out a study of the main models used by the agile and traditional software development market, its characteristics, applicability, techniques and devices. When identified the main techniques and artifacts of the models, the study was conducted in a specific team, identified their defective points and suggest the best found in each model, according to the negative factors encountered in the process of software development team.

Keywords: Agile modeling. Scrum. Extreme Programming. Software Quality. Productivity.

LISTA DE SIGLAS

UML – Unified Modeling Language

XP – Extreme Programming

DAS – Desenvolvimento Adaptativo de Software

DSDM – Método de Desenvolvimento Dinâmico de Sistemas

FDD – Desenvolvimento por Características

WIP – Working in Progress

SLA - Service Level Agreements

RUP – Rational Unified Process

SH – Software House

TDD – Test-Driven Development

SUMÁRIO

1 INTRODUCÃO **12**
1.1 DELIMITAÇÃO DO TEMA 14
1.2 PROBLEMATIZAÇÃO 14
1.3 OBJETIVOS GERAIS 15
1.3.1 Objetivo geral *15*
1.3.2 Objetivos específicos *15*
1.4 JUSTIFICATIVA 16
1.5 PROCEDIMENTOS METODOLÓGICOS 17
1.5.1 Método de abordagem *17*
1.5.2 Método de procedimento *18*
1.5.3 Técnicas de pesquisa *18*
2 MODELOS DE AGÉIS **19**
2.1 PROGRAMAÇÃO EXTREMA 21
2.1.1 Praticas da programação extrema *25*
2.1.1.1 Jogo do planejamento 25
2.1.1.2 Entregas freqüentes 26
2.1.1.3 Metáfora 26
2.1.1.4 Projetos simples 26
2.1.1.5 Testes 27
2.1.1.6 Refatoração 27
2.1.1.7 Programação em pares 28
2.1.1.8 Propriedade coletiva 28
2.1.1.9 Integração contínua 28
2.1.1.10 Semana de 40 horas 29
2.1.1.11 Cliente presente 29
2.1.1.12 Padrão de codificação 30
2.2 SCRUM 30
2.2.1 Artefatos do Scrum *34*
2.3 KANBAN 37
3 MODELOS TRADICIONAIS **43**
3.1 RATIONAL UNIFIED PROCESS (RUP) 44
3.2 ICONIX 53
4 ESTUDO DE CASO **56**
4.1 EQUIPE DE MANUTENÇÃO EVOLUTIVA 57
4.1.1 Planejamento *58*
4.1.2 Codificação *60*
4.1.3 Geração de versão *61*
4.1.4 Testes Gerais *62*
4.1.5 Retrospectiva *62*
4.1.6 Liberação *63*
4.1.7 Outras práticas e artefatos *63*
4.2 EQUIPE DE MANUTENÇÃO CORRETIVA 64
5 PROPOSTA DE MELHORIA SUGERIDA **67**

6 CONCLUSÕES 79

REFERÊNCIAS 82

LISTA DE FIGURAS

Figura 1: Práticas reforçam umas as outras 25

Figura 2: Gráfico burndown 35

Figura 3: Quadro Scrum 36

Figura 4: Quadro Kanban 37

Figura 5: Gráfico de Fluxo Cumulativo. 39

Figura 6: Gráfico Tempo de Ciclo. 40

Figura 7: Gráfico Taxa de Defeitos 41

Figura 8: Gráfico Itens Bloqueados 41

Figura 9: Arquitetura RUP 46

Figura 10: Papéis envolvidos e artefatos produzidos na disciplina modelagem de negócio 48

Figura 11: Papéis envolvidos e artefatos produzidos na disciplina requisitos. 48

Figura 12: Papéis envolvidos e artefatos produzidos na disciplina de análise e design.. 49

Figura 13: Papéis envolvidos e artefatos produzidos na disciplina implementação. 49

Figura 14: Papéis Os artefatos desenvolvidos como produtos das atividades de teste e avaliação agrupados por papel de responsabilidade. 50

Figura 15: Os papéis envolvidos e os artefatos produzidos na disciplina Implantação 50

Figura 16: Os papéis envolvidos e os artefatos produzidos na disciplina Gerenciamento de Configuração e Mudança. 51

Figura 17: Os papéis envolvidos e os artefatos produzidos na disciplina Gerenciamento de Projeto. 51

Figura 18: Os papéis envolvidos e os artefatos produzidos na disciplina de Ambiente. 52

Figura 19: Visão macro do ICONIX. 54

Figura 20: Cronograma Equipe Manutenção Evolutiva. 58

1 INTRODUCÃO

Em um nicho de mercado cada vez mais competitivo, empresas tradicionais de software necessitam de processos de produção bem definidos e objetivos que garantam a vida útil de seu produto. Baseados em modelos ou metodologias, os processos definem qual a forma de organização e gerenciamento que cada projeto será trabalhado. Modelos ou metodologias tratam - se de abordagens organizadas por meio de passos estabelecidos para atingir objetivos específicos. Segundo Rezende (2006, p. 105) “Metodologia é um roteiro que permite uma ou várias técnicas por opção dos desenvolvedores do sistema de informação ou software”.

Para Rezende (2006), o modelo deve auxiliar o desenvolvimento do projeto, sistema ou software, de modo que os mesmos atendam as necessidades do cliente ou usuário, com os recursos disponíveis e dentro de um prazo ideal estabelecido em conjunto. O modelo deve ser de toda a organização e para a organização, de maneira que seja elaborado e utilizado por todos. Também deve ser revisado, atualizado e complementado na medida do desenvolvimento do projeto.

Na visão de Pressman (2006), processo de desenvolvimento é um roteiro que ajuda a criar a tempo um resultado de alta qualidade. Fornece organização, estabilidade e controle para uma atividade que pode, se deixado sem controle, se tornar caótica, no entanto, uma abordagem precisa ser ágil, precisa exigir apenas atividades, controle e documentações adequadas a equipe e projeto a ser produzido.

Existem vários modelos de processos definidos com características e aplicações próprias. Pressman (2006) caracteriza os modelos prescritivos como sendo aqueles que buscam a ordem e estrutura para desenvolvimento de software, realizam um mesmo conjunto de atividades genéricas de arcabouço, comunicação, planejamento, modelagem, construção e implantação. O mesmo autor destaca os seguintes modelos prescritivos:

- Modelo em cascata sugere progressão linear das atividades de arcabouço que é frequentemente inconsistente com a realidade

moderna. Possui aplicabilidade em situações em que os requisitos são bem definidos e estáveis.

- Modelo incremental produz software como uma série de incrementos.
- Modelo evolucionário é projetado para acomodar modificações, adotados para serem aplicados ao longo de todas as atividades de engenharia de software, desde o desenvolvimento de conceitos até atividades de manutenção de longo prazo.
- Modelo baseado em componentes enfatiza a reutilização.
- Processo unificado é um processo centrado na arquitetura, iterativo e incremental. Projetado como um arcabouço para métodos e ferramentas UML (Unified Modeling Language).

O ambiente moderno de desenvolvimento de software é apressado e mutável, requisitos sofrem alterações constantes e novos requisitos surgem no desenvolvimento de projetos. Engenharia de software ágil, conforme Pressman (2006), combina uma filosofia e um conjunto de diretrizes de desenvolvimento. A filosofia encoraja a satisfação do cliente e a entrega incremental do software, enquanto que, as diretrizes enfatizam a entrega em contraposição à análise, ao projeto e a comunicação ativa e contínua. Assim a engenharia de software ágil representa uma alternativa razoável para certas categorias e tipos de software a serem desenvolvidos em projetos.

De acordo com Pressman (2006), modelo ágil ressalta quatro tópicos chaves: a importância de equipes auto organizadas que tem controle sobre o trabalho que executam, comunicação e colaboração entre os membros da equipe, reconhecimento de que modificações representam oportunidade e uma ênfase na entrega rápida do software.

Para Pressman (2006), modelos ágeis estão em foco e encontramos alguns como: XP (Extreme Programming), DAS (Desenvolvimento Adaptativo de Software), DSDM (Método de Desenvolvimento Dinâmico de Sistemas), Scrum, Crystal, FDD (Desenvolvimento por Características). Scrum e Extreme Programming são os modelos mais aplicáveis para projetos comuns de desenvolvimento de software, cada modelo possui características próprias e aplicabilidades distintas.

Modelos ágeis não extinguem de seu processo, etapas de documentação ou qualquer outro artefato que colabore com o entendimento. Modelos ágeis também não definem que todas as regras devam ser seguidas em sua essência e que não possam ser adequadas ou extinguidas. Modelos ágeis estimulam a comunicação direta entre os membros e organização da equipe, a criação/uso de documentos, artefatos ou técnicas que realmente são necessários a cada cenário de fabrica. Promove a utilização dos recursos apenas ao que é necessário, propondo sempre ganho em tempo e assertividade nos planejamentos, melhoria da qualidade, satisfação do cliente a cada entrega do produto.

1.1 DELIMITAÇÃO DO TEMA

Um estudo envolvendo modelos de processos de engenharia de software, abordando técnicas, artefatos e características visando a aplicação em uma equipe de desenvolvimento de software com seu processo baseado em Scrum.

1.2 PROBLEMATIZAÇÃO

Empresas de desenvolvimento de software, buscam em modelos de processos a solução para otimizar recursos e aperfeiçoar técnicas de produção. Modelos ágeis surgem para o ganho de tempo e difusão do conhecimento, entretanto, quando aplicados em sua essência em empresas tradicionais com elevado número de funcionários e alta complexidade do produto, encontram impedimentos contraditórios ao modelo, que impedem que o ganho em sua aplicação seja real.

1.3 OBJETIVOS GERAIS

Para a construção da monografia serão elaborados os seguintes objetivos:

1.3.1 Objetivo geral

Estudar e apresentar técnicas de diversos modelos de desenvolvimento de software, afim de, preencher pontos negativos na aplicação do modelo ágil Scrum em empresas tradicionais.

1.3.2 Objetivos específicos

- Estudar técnicas de modelos de processo ágeis e modelos tradicionais.
- Identificar técnicas trabalhadas em uma equipe de desenvolvimento com possíveis perca de desempenho e qualidade.
- Estudar técnicas e procedimentos que possam suprir pontos falhos na aplicação do modelo ágil Scrum em uma equipe de desenvolvimento.
- Propor e apresentar a junção de técnicas de diferentes modelos, visando um ganho efetivo em sua aplicação.

1.4 JUSTIFICATIVA

Através de estudos detalhados de vários modelos ágeis e tradicionais, pode – se maximizar o ganho de produção em uma empresa, utilizando apenas o que condiz com sua realidade e o que lhe promove ganhos. Existem soluções para vários problemas em vários modelos, porém sabemos que cada empresa possui sua realidade e forma de trabalho. Uma característica definida como positiva em um modelo, pode ser positiva apenas para determinadas empresas, em uma empresa semelhante, essa mesma característica pode ser definida como ponto negativo. Devemos entender cada característica de cada modelo e a realidade de cada empresa para podermos propor ganhos e qualidade no processo de desenvolvimento.

Customizando e unificando técnicas de diferentes modelos, pode - se obter ganhos de desempenho, redução de custos e assertividades em prazos e estimativas de projeto. Essa tarefa implica diretamente na qualidade do desenvolvimento e no trabalho dos desenvolvedores. Trabalhando com um modelo correto e padronizado, se obtém maiores acertos diminuindo excesso de trabalho para projetos mal previstos, diminuindo bruscamente trabalhos já realizados e reduzindo consequentemente o custo total de um projeto. Um modelo que a informação esteja acessível a todos, um banco de conhecimento para que todos tenham o entendimento total e correto do projeto.

É comum que empresas busquem ganho em desempenho e agilidade no desenvolvimento de software. Seguindo esses objetivos, empresas apontam as metodologias ágeis como à solução. Em um cenário real, sabemos que existem alguns pontos divergentes que podem comprometer todo o planejamento ágil de um projeto como desnivelamento de conhecimento da equipe, erros de especificação e arquitetura, mudanças de requisitos, mudanças de prioridades e até mesmo erros gerados no próprio desenvolvimento iterativo do produto.

Os riscos citados interferem diretamente em qualquer planejamento, com isso é gerado um trabalho desordenado e maçante. Como todos os planejamentos têm prazos a cumprir, fatalmente algumas fases de projetos são reduzidas

drasticamente a fim de propiciar a entrega no prazo estipulado, porém interferindo negativamente na qualidade. Tais interferências, qualidade comprometida, prazos apertados vão se tornando rotineiros e sempre aumentando exponencialmente à medida que cada novo ciclo da iteração do desenvolvimento é iniciado.

Conhecendo esses riscos podemos utilizar e adequar técnicas de modelos tradicionais como documentação, diagramas UMLs, programas de disseminação de conhecimento, gráficos e indicadores, técnicas de testes e qualidade aos modelos ágeis com objetivo de suprir necessidades e fragilidades de alguns modelos quando aplicados em uma grande equipe e um produto complexo.

O modelo adequado a uma empresa grande e complexa poderá facilitar o trabalho de empresas menores, provendo soluções em produção de software para demais empresas da área de tecnologia. Mesmo que não aplicado por inteiro, alguma técnica estudada pode ser agregada a processos de outras empresas. É de conhecimento que cada empresa possuí suas particularidades, justamente por isso que modelos podem ser adequados e adaptáveis para diferentes cenários.

1.5 PROCEDIMENTOS METODOLÓGICOS

Esta monografia será realizada com os tipos de pesquisa exploratória e explicativa. Explorando problemas relacionados aos modelos de software buscando evidenciar e justificar suas ocorrências.

1.5.1 Método de abordagem

Como método de abordagem será utilizado o pensamento dedutivo, visando utilizar o modelo existente para abranger diversas técnicas de outros modelos.

Com intuito de análise e identificação de fatores positivos para melhoria no processo de desenvolvimento, a pesquisa será de natureza qualitativa.

1.5.2 Método de procedimento

Dentre os métodos de procedimentos, será abordado o método monográfico e estudo de caso aplicado a uma equipe de desenvolvimento pertencente à empresa SH. O estudo de caso irá abranger as técnicas e processos de desenvolvimento praticados atualmente pela equipe.

1.5.3 Técnicas de pesquisa

Como técnicas de pesquisa serão utilizadas pesquisas bibliográficas e pesquisa de campo baseado em uma equipe de desenvolvimento de software pertencente à empresa SH. A pesquisa bibliográfica será realizada mediante livros, artigos e sites da internet.

2 MODELOS DE AGÉIS

Conforme Pressman (2006), em essência, modelos ágeis foram desenvolvidos em um esforço para vencer as fraquezas percebidas e reais da engenharia de software convencional. O desenvolvimento ágil pode fornecer importantes benefícios, mas não é aplicável a todos os projetos, produtos, pessoas ou situações.

Na visão de Jacobson (2002), o acolhimento de modificações é o principal guia para agilidade. Os engenheiros de software devem reagir rapidamente se tiverem de acomodar as rápidas modificações. Para Pressman (2006), agilidade é mais do que uma resposta efetiva a modificação, encoraja estruturas e atitudes da equipe que tornam a comunicação mais fácil. Enfatiza a rápida entrega de software operacional e dá menos importância para produtos de trabalhos intermediários; adota os clientes como parte da equipe de desenvolvimento; reconhece que o planejamento em mundo incerto tem seus limites e que um plano de projeto deve ser flexível.

A Aliança Ágil (2011) define 12 princípios para aqueles que querem alcançar agilidade:

1. Nossa maior prioridade é satisfazer ao cliente desde o início por meio de entrega continua de software valioso.

2. Modificações de requisitos são bem vindas, mesmo que tardias. Os processos ágeis aproveitam as modificações como vantagens para a competitividade do cliente.

3. Entrega de software funcionando frequentemente, de preferência em curto espaço de tempo.

4. O pessoal do negócio e do desenvolvimento deve trabalhar juntos durante todo o projeto.

5. Construção de projetos em torno de indivíduos motivados. Forneça – lhes o ambiente e apoio que precisam e confie que eles farão o trabalho.

6. O método mais eficiente de levar informação para uma equipe de desenvolvimento é a conversa face a face.

7. Software funcionando é a principal medida do progresso.

8. Processos ágeis promovem desenvolvimento sustentável. Os patrocinadores, desenvolvedores e usuários devem ser capazes de manter um ritmo constante, indefinidamente.

9. Atenção contínua a 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.

De acordo com Pressman (2006), a agilidade pode ser aplicada a qualquer processo de software, basta que o processo seja projetado de modo que permita à equipe de projeto adaptar tarefas e aperfeiçoa – las, conduzir o planejamento, eliminar tudo exceto o trabalho mais essencial e mantê – los simples, e enfatizar estratégia de entrega incremental.

Conforme Fowler (2002, citado por PRESSMAN, 2006, p. 60), qualquer processo ágil de software é caracterizado de modo que atenda três suposições chave sobre a maioria dos projetos de software:

- É difícil prever antecipadamente quais requisitos de software vão persistir e quais serão modificados. É igualmente difícil prever como as prioridades do cliente serão modificadas à medida que o projeto prossegue.
- Para muitos tipos de software, o projeto e a construção são intercalados. É difícil prever o quanto de projeto é necessário antes que a construção seja usada para comprovar o projeto.

- Análise, projeto, construção e testes não são tão previsíveis como gostaríamos.

Com toda essa política de agilidade, atualmente encontramos verdadeiros debates sobre os benefícios e a aplicabilidade do desenvolvimento ágil em contraposição aos processos mais convencionais de engenharia de software. Highsmith (2002) descreve os extremos ao caracterizar o sentimento do campo pró – agilidade: “Os metodologistas tradicionais são um punhado de bitolados que preferem produzir documentação perfeita a um sistema funcionando que satisfaça as necessidades do negócio”. Em contrapartida, ele descreve a posição do campo da engenharia de software: “Os metodologistas levianos são um punhado de gloriosos hackers que terão uma grande surpresa quando tiverem de ampliar seus brinquedos para chegar a um software que abranja toda a empresa”.

Não existe modelo único correto e respostas absolutas, há muitos modelos propostos, cada qual com sua abordagem. O melhor e o que há muito a ser ganho é considerar o melhor de cada proposta. Nas subseções seguintes, apresentaremos um panorama dos modelos ágeis mais utilizados e que satisfazem os princípios de desenvolvimento ágil mencionados anteriormente.

2.1 PROGRAMAÇÃO EXTREMA

Programação extrema (XP) é uma metodologia ágil para equipes pequenas e médias que desenvolvem software baseados em requisitos vagos e que se modificam rapidamente. A XP enfatiza o desenvolvimento rápido do projeto e visa garantir a satisfação do cliente, além de favorecer o cumprimento das estimativas. Aborda o risco do desenvolvimento de software em todas as etapas do processo de desenvolvimento (BECK, 2000).

Segundo Beck (2000), a XP trata alguns riscos encontrados em desenvolvimento de software com as seguintes práticas:

Deslizes no cronograma: A XP exigi ciclos curtos de entrega, fazendo com que a extensão de qualquer deslize seja limitada. Dentro de cada ciclo de

entrega, a XP planeja tarefas com duração de no máximo três dias, para que o time possa resolver problemas até mesmo no período de um ciclo. As tarefas que compõem um ciclo são ordenadas de acordo com sua prioridade, assim, qualquer função que não for contemplada na versão será de prioridade menor.

Projeto cancelado: A XP pede ao cliente que escolha o menor release que faça mais sentido considerando o negócio, para que haja menos coisas que possam dar errado antes de se ir para fase de produção e para que o valor do software seja o maior possível.

Deterioração do sistema: Testes são realizados e mantidos em conjuntos, os quais são executados e reexecutados após cada modificação para assegurar o padrão de qualidade. A XP sempre mantém o sistema em excelente condição, não é permitido que código redundante seja acumulado.

Taxa de erros: São realizados testes sob perspectiva tanto do programador, desenvolvendo testes a cada função, quanto do cliente, desenvolvendo testes a cada funcionalidade do sistema.

Negócio mal compreendido: A XP convida o cliente para ser uma parte integrante do time. A especificação do projeto é continuamente refinada durante o desenvolvimento.

Modificações no negócio: Com ciclos curtos, modificações durante um desenvolvimento de um release são menores.

Falsa riqueza de funções: A XP insiste aos programadores que apenas as tarefas com maior prioridade sejam abordadas.

Rotatividade da equipe: Os programadores aceitam a responsabilidade de estimarem e completar seus próprios trabalhos. A XP retorna o tempo real despendido, para que as estimativas possam ser melhoradas.

Na visão de Beck (2000), em um ciclo de desenvolvimento XP notamos características como a programação em pares, pares de programadores programam juntos; o desenvolvimento é voltado a testes, primeiro testamos e depois codificamos, a funcionalidade só é concluída quando todos os testes rodem; os pares não fazem apenas os casos de testes rodarem, eles também evoluem o

sistema, acrescentam valor à análise, ao projeto e a implementação do sistema; a integração ocorre imediatamente após o desenvolvimento do sistema, incluindo o teste de integração.

De acordo com Beck (2000), desenvolvimento de software deve conter quatro variáveis: custo, tempo, qualidade e escopo. Essas variáveis se interagem e escopo deve ser a variável de maior importância. O foco deve estar sempre no escopo, este que será responsável direto pelo custo, tempo e qualidade. Sempre que necessário este pode e deve ser reduzido.

A XP abrange quatro valores, conforme é detalhado por Beck (2000):

Comunicação: Através de práticas como teste de unidade, programação em pares e estimativas de tarefas, a XP procura manter a comunicação certa e fluindo. O efeito de testar, programar e estimar é a comunicação entre programadores, clientes e gerentes. Isso não quer dizer que a comunicação não seja obstruída. As pessoas ficam com medo, erram e se distraem. A XP emprega um "treinador" cujo trabalho é perceber quando as pessoas não estão se comunicando e restabelecer o vínculo entre elas.

Simplicidade: A XP aposta em que é melhor fazer uma coisa simples hoje e pagar um pouco mais amanhã para fazer alguma modificação se for necessário do que fazer uma coisa mais complicada que talvez nunca seja usada.

Feedback: Um problema comum entre programadores é o otimismo, para este problema o feedback é a solução. Na XP o feedback funciona em diferentes escalas de tempo. Primeiramente em escala de dias e minutos. Os programadores escrevem testes de unidade e têm feedback concreto do estado do sistema. Quando "histórias" são escritas e avaliadas imediatamente, assim temos o feedback concreto da qualidade das "histórias". A pessoa que rastreia o progresso observa o término das tarefas, para dar ao time o feedback da probabilidade de eles terminarem no tempo planejado. Funciona também em escalas de semanas e meses quando cliente e testadores escrevem testes de funcionalidade para todas as "histórias", quando é realizada uma revisão do cronograma e quando o sistema é colocado em produção.

Coragem: Se você não tiver definido os três primeiros valores, coragem por si só não resolverá problemas. No entanto, quando combinada com comunicação, simplicidade e feedback concreto, a coragem se torna extremamente valiosa.

Além dos quatros valores destacados anteriormente, Beck (2000, p. 49) alega que "Se os membros do time não se importarem uns com os outros e com o que eles estão fazendo, a XP estará condenada. Da mesma forma, a maioria das outras abordagens de software também estará". Estes valores nos dão critérios para uma solução de sucesso, no entanto, são muito vagos para nos ajudar a decidir quais praticas usar, é preciso transformar os valores em princípios concretos que possam ser usados. Um princípio é mais concreto, ou você tem feedback rápido ou não. Os princípios fundamentais são: feedback, simplicidade presumida, mudanças incrementais, aceitação das mudanças e alta qualidade (BECK, 2000).

Beck (2000), também define outros princípios, menos fundamentais:

- Ensinar aprendendo;
- Investimento inicial pequeno;
- Jogar para ganhar;
- Experimentação concreta: testar, utilizar, experimentar o sistema;
- Comunicação honesta e franca;
- Trabalhar a favor dos instintos do pessoal, e não contra eles;
- Aceitação de responsabilidades;
- Adaptação local: programador decidi como desenvolver;
- Viajar com pouca bagagem: artefatos devem ser poucos e valiosos.
- Métricas genuínas: níveis de detalhamento não são suportados.

Com todas as informações anteriores obtemos então a matéria prima para o desenvolvimento ágil XP: histórias, quatro valores, princípios e quatro atividades básicas (codificar, testar, ouvir e projetar). Na visão de Beck (2000), nosso trabalho é estruturar as quatro atividades sob a ótica da longa lista de princípios por vezes contraditórios.

2.1.1 Praticas da programação extrema

A metodologia XP é baseada em 12 práticas que serão apresentadas detalhadamente posteriormente. Nenhuma prática consegue se manter por si só (com exceção dos testes). O diagrama da figura 1 resume as práticas, onde uma linha conectada a duas práticas mostra que elas reforçam uma a outra.

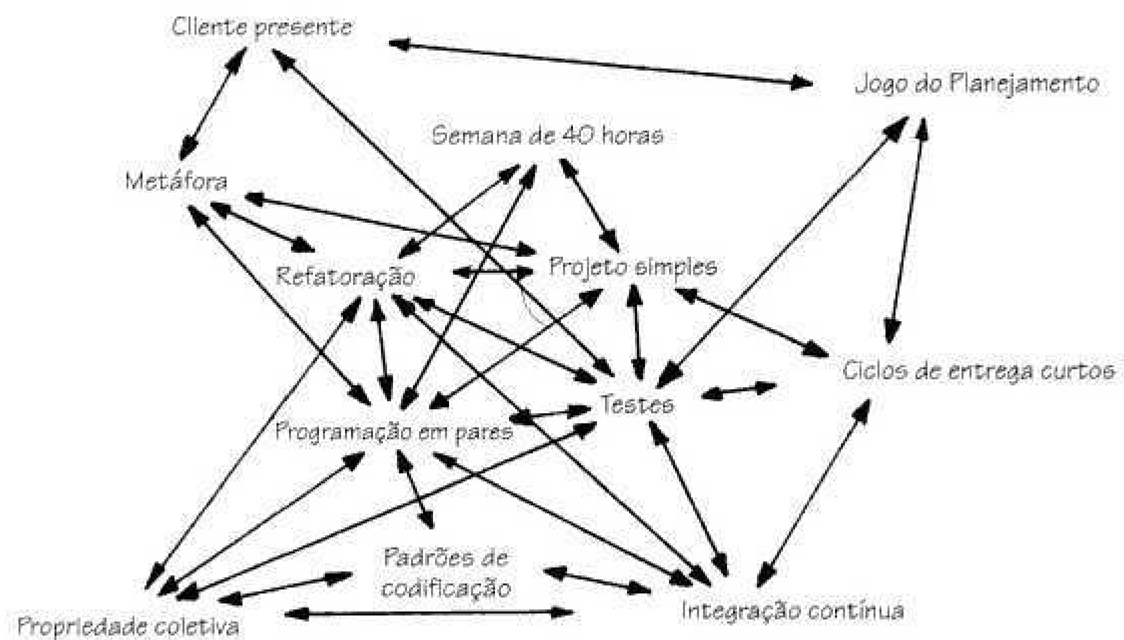

Figura 1: Práticas reforçam umas as outras. **Fonte:** Beck, 2000.

2.1.1.1 Jogo do planejamento

Nele é determinado o escopo da próxima versão considerando as prioridades de negócio e estimativas técnicas. Na definição do escopo da próxima versão deve prevalecer um diálogo evolutivo entre o possível e o desejado entre pessoas da área de negócio e as pessoas técnicas (BECK, 2000).

Área de negocio precisa decidir sobre o escopo, prioridade, composição das versões e datas de entrega. No entanto não conseguem tomar essas decisões sem auxilio técnico, necessitam da área técnica para auxílio na tomada de decisões.

Pessoas técnicas fornecem informações de estimativas, conseqüências, processo de como será trabalhado e o cronograma detalhado. O escopo é onde irá constar quais “histórias” serão realizadas em prioridades de acordo com os riscos identificados pelos programadores e relevância de cada uma para o cliente (BECK, 2000).

2.1.1.2 Entregas freqüentes

Versões simples são colocadas rapidamente em produção, novas versões são sempre liberadas em ciclos curtos. As versões devem ter o menor tamanho possível, contendo os requisitos de maior valor para o negócio por completo. Também não podemos reduzir drasticamente uma versão para ser entregue em um curto período com funcionalidades incompletas. Versões curtas nos propiciam identificação e solução mais rápida para problemas de projeto encontrados nos desenvolvimentos dos requisitos (BECK, 2000).

2.1.1.3 Metáfora

Simples história de como o sistema funciona, compartilhado por todos e responsável por guiar o desenvolvimento. Às vezes a metáfora é simplória, às vezes requer algumas explicações, entretanto elas são apenas metáforas por que não querem dizer literalmente o que é o sistema, apenas ajuda todos os envolvidos a entenderem os elementos básicos e seus relacionamentos (BECK, 2000).

2.1.1.4 Projetos simples

Sistema deve ser projetado da maneira mais simples possível em qualquer momento. Complexidade desnecessária deve ser removida assim que

descoberta. Um projeto correto é aquele que executa todos os testes, não tem lógica duplicada, expressa todas as intenções importantes para os programadores e tem o menor número possível de classes e métodos. Todos os demais códigos, classes ou métodos que podem ser removidos sem violar as regras, podem prejudicar o sistema, aumentar o risco e prejudicar a entrega. O futuro é incerto e acrescentar funcionalidades baseadas em especulações não é a melhor solução, acrescente quando realmente for preciso (BECK, 2000).

2.1.1.5 Testes

Os programadores escrevem testes de unidade continuamente, os quais devem ser executados sem falhas para que o desenvolvimento prossiga. Os clientes escrevem testes demonstrando que as funções estão terminadas. O resultado é que o programa se torna cada vez mais confiável e com o tempo ele se torna capaz de aceitar modificações. É preciso escrever um teste para cada método que for programado, os testes é que indicarão o término de uma atividade (BECK, 2000).

2.1.1.6 Refatoração

Os programadores reestruturam o sistema sem alterar seu comportamento a fim de remover duplicidade, melhorar a comunicação, simplificar e acrescentar flexibilidade. Ao implementar uma função do programa, os programadores se perguntam se existem uma maneira de alterar o programa existente para fazer com que a adição de nova função seja simples. Depois que eles adicionaram, se perguntam como poderia simplificar o programa, mantendo todos os testes executando. Isso é chamado de refatoração (BECK, 2000).

É notável que isso signifique maior trabalho do que seria preciso apenas para acrescentar uma função. Mas ao trabalhar dessa forma, garantimos que poderá adicionar a próxima função com uma quantidade aceitável de esforço e todas as

seguintes também. No entanto, não refatoramos com base em especulações, refatoramos quando é preciso.

2.1.1.7 Programação em pares

Todo código produzido é escrito por dois programadores em uma máquina. Existem dois papéis em cada par, um parceiro, aquele que com teclado e mouse está pensando qual é o melhor jeito de implementar este método específico e o outro parceiro que está pensando estrategicamente (BECK, 2000).

A programação em pares é dinâmica. Se duas pessoas formam um par pela manhã, à tarde elas podem facilmente fazer duplas com outras pessoas. Se você for responsável por uma tarefa de uma área que não lhe é familiar, você pode pedir a alguém com experiência recente para fazer uma dupla. Mas na maioria das vezes, qualquer pessoa do time servirá como parceiro (BECK, 2000).

2.1.1.8 Propriedade coletiva

Qualquer um pode codificar qualquer código, em qualquer lugar do sistema, a qualquer momento. Na XP, todos são responsáveis pelo sistema inteiro. Nem todos conhecem todas as partes igualmente bem, mas todos sabem um pouco sobre cada parte. Se um par está trabalhando e vê uma oportunidade de melhorar o código, eles vão em frente e o melhoram se isso facilitar suas vidas (BECK, 2000).

2.1.1.9 Integração contínua

Integre e atualize as versões do sistema várias vezes por dia, cada vez que uma tarefa for terminada. O código é integrado e testado após algumas horas e

no máximo um dia de desenvolvimento. Uma maneira simples de fazer isso é ter uma máquina dedicada apenas para a integração (BECK, 2000).

Na visão de Beck (2000), integrar apenas um conjunto de modificações de cada vez é uma prática que funciona bem porque fica óbvio quem deve fazer as correções quando um teste falhar.

2.1.1.10 Semana de 40 horas

Trabalhe no máximo 40 horas por semana, nunca faça horas extras por duas semanas seguidas. Pessoas diferentes têm tolerâncias diferentes para o trabalho. Uma pessoa é capaz de agüentar 35 horas concentrada, outra 45 horas. Mas ninguém é capaz de agüentar 60 horas por semana durante muitas semanas e ainda estar disposto, criativo, cuidadoso e confiante (BECK, 2000).

Horas extras são sintoma de um problema sério no projeto. A regra do XP é simples, você não pode trabalhar uma segunda semana com horas extras. Por uma semana, tudo bem, vá adiante e trabalhe algumas horas extras. Se você chega a segunda e precisa adicionar horas para completar o objetivo, então temos um problema que não pode ser resolvido com horas extras (BECK, 2000).

2.1.1.11 Cliente presente

Um cliente deve ser incluído no time, disponível o tempo todo para responder questões. A maior objeção a essa regra é que usuários reais do sistema em desenvolvimento são muito valiosos para ser emprestado ao time. Os gerentes terão de decidir o que vale mais: ter o software acabado mais cedo e melhor ou acumular o trabalho de uma ou duas pessoas. Se ter o sistema não agrega mais valor ao negócio do que ter uma pessoa a mais trabalhando, talvez o sistema não devesse ser desenvolvido (BECK, 2000).

2.1.1.12 Padrão de codificação

Os programadores escreverão códigos respeitando as regras que enfatizam a comunicação através do código. Se programadores diferentes trabalham em partes diferentes do sistema, trocando constantemente de duplas e refatorando os códigos uns dos outros constantemente, você pode ter conjuntos diferentes de práticas de codificação. Com um pouco de prática, será impossível dizer qual pessoa do time escreveu que código (BECK, 2000).

Para Beck (2000), o padrão adotado deve exigir a menor quantidade de trabalho possível, consistente com a regra de sem código duplicado. O padrão deve enfatizar a comunicação.

2.2 SCRUM

Scrum é um framework dentro do qual pode - se empregar diversos processos e técnicas. Não é um processo ou uma técnica para o desenvolvimento de produtos. Emprega uma abordagem iterativa e incremental para aperfeiçoar a previsibilidade e controlar riscos, sustentado por três pilares: transparência, inspeção e adaptação (SCHWABER; SUTHERLAND, 2011).

O framework Scrum consiste em um conjunto formado por time, eventos com duração fixa, artefatos e regras (SCHWABER; SUTHERLAND, 2011).

Times Scrum são projetados para otimizar flexibilidade e produtividade. São auto – organizados, interdisciplinares e trabalham em iterações. Dentro de um time Scrum encontramos papéis bem definidos como Scrum master, product owner e a equipe que executa o trabalho propriamente dito (SCHWABER; SUTHERLAND, 2011).

Scrum master: É responsável por garantir que o time esteja aderindo aos valores, práticas e regras do Scrum. É responsável por ajudar o time, treinando-os e levando-os a serem mais produtivos e ao mesmo tempo, desenvolvendo produtos de

maior qualidade. Faz com que o time entenda a usar autogerenciamento e interdisciplinaridade. Ajuda o time a fazer o seu melhor em um ambiente organizacional que pode ainda não ser otimizado para o desenvolvimento de produtos complexos. No entanto, não tem a função de gerenciar o time, o time deve ser auto – organizável (SCHWABER; SUTHERLAND, 2011).

Product owner: É a única pessoa responsável pelo gerenciamento do backlog do produto e por garantir o valor do trabalho realizado. Mantém o backlog do produto e garante que está visível a todos de forma priorizada. Ninguém tem a permissão de dizer ao time para trabalhar em outro conjunto de prioridades (SCHWABER; SUTHERLAND, 2011).

Equipe: São desenvolvedores que transformam o backlog do produto em incrementos de funcionalidades potencialmente entregáveis em cada sprint. São interdisciplinares e auto – organizáveis, todos contribuem, mesmo que isso exija aprender novas habilidades (SCHWABER; SUTHERLAND, 2011).

Eventos com duração fixa criam regularidade. Temos como elementos de duração fixa, reunião de planejamento do release, reunião de planejamento da sprint, sprint, reunião diária, revisão da sprint e retrospectiva da sprint (SCHWABER; SUTHERLAND, 2011).

Reunião de planejamento do release: O planejamento do release requer estimar e priorizar o backlog do produto. O propósito é estabelecer um plano e metas que o time e o resto da organização possam entender e comunicar. Envolve questões como: Como podemos transformar a visão em um produto vencedor da melhor maneira possível? Como podemos alcançar a satisfação do cliente e o retorno do investimento desejado? O plano do release estabelece a meta, as maiores prioridades do backlog do produto, os principais riscos, as características gerais e funcionalidades que estarão contidas no release. Estabelece uma data de entrega e custos prováveis que deve se manter se tudo sair como planejado (SCHWABER; SUTHERLAND, 2011).

Ao se utilizar o Scrum os produtos são construídos iterativamente, de maneira que cada sprint cria um incremento do produto, iniciado pelo de maior valor e maior risco. Assim a cada sprint o produto se torna de mais valor para o cliente.

Cada incremento é um pedaço potencialmente entregável do produto completo (SCHWABER; SUTHERLAND, 2011).

Sprint: É uma iteração, são eventos com duração fixa. Contêm e consistem na reunião de planejamento de sprint, trabalho de desenvolvimento, revisão e retrospectiva da sprint. Ocorre uma após a outra, sem intervalos entre elas. É aconselhável que tenha curto tempo de duração, um mês de média, sprints longas podem fornecer riscos e experiências traumáticas quando falham (SCHWABER; SUTHERLAND, 2011).

Reunião de planejamento da sprint: Reunião no qual a iteração é planejada, consiste em duas partes onde a primeira defini o que será feito na sprint e a segunda é o momento onde o time entende o que será feito e como será desenvolvido esta funcionalidade (SCHWABER; SUTHERLAND, 2011).

O product owner participa da reunião juntamente com a equipe de desenvolvimento e apresenta o que é mais prioritário no backlog do produto, também pode ser convidadas pessoas para fornecer conselhos técnicos ou sobre o domínio em questão. As entradas para essa reunião são o backlog do produto, o incremento mais recente do produto, a capacidade do time e a decisão do quanto do backlog será selecionado. O time é responsável por avaliar e definir o quanto que será desenvolvido na sprint (SCHWABER; SUTHERLAND, 2011).

Com backlog do produto selecionado, a meta da sprint é delineada. A meta é o objetivo que será atingido através da implementação do backlog do produto, fornece a razão ao time pelo qual está se desenvolvendo o incremento (SCHWABER; SUTHERLAND, 2011).

O time define como transformará o backlog do produto selecionado em um incremento pronto. O time projeta o trabalho, enquanto projeta identifica tarefas. Essas tarefas são pedaços detalhados do trabalho necessário para converter o backlog do produto em software funcional. Tarefas são decompostas para que possam ser realizadas em menos de um dia, com a lista de tarefas temos um backlog da sprint (SCHWABER; SUTHERLAND, 2011).

O time se auto – organiza para se encarregar e se responsabilizar pelo trabalho contido no backlog da sprint, tanto durante a reunião de planejamento quanto no próprio momento de execução.

Revisão da sprint: Realizada ao final de uma sprint, o time e as partes interessadas colaboram sobre o que acabou de ser feito. É uma reunião informal, com a apresentação da funcionalidade, que tem a intenção de promover a colaboração sobre o que fazer em seguida (SCHWABER; SUTHERLAND, 2011).

O time discute sobre o que correu bem durante a sprint e quais foram os problemas enfrentados, além de como os problemas foram resolvidos.

Retrospectiva da sprint: Após revisão da sprint, o Scrum master encoraja o time a revisar, dentro do modelo de trabalho e das práticas Scrum, seu processo de desenvolvimento, de forma a torna – lo mais eficaz para a próxima sprint. Tem como finalidade inspecionar como ocorreu a ultima sprint em se tratando de pessoas, relações entre elas, dos processos e ferramentas. A inspeção deve identificar e priorizar os principais itens que correram bem e aqueles que poderiam ter deixado o andamento da sprint ainda melhor, incluindo a composição do time, preparativos para reuniões, ferramentas, definição de “pronto”, métodos de comunicação e processos. No final da reunião o time deve ter identificado medidas de melhoria factíveis que ele implementará na próxima iteração (SCHWABER; SUTHERLAND, 2011).

Reunião diária: O time se encontra diariamente para uma reunião que aborda o que cada um realizou desde a ultima reunião diária, o que cada um vai fazer até a próxima reunião diária e se encontrou impedimentos em seu caminho (SCHWABER; SUTHERLAND, 2011).

São reuniões curtas de aproximadamente 15 minutos, sempre no mesmo horário e no mesmo local, com todos participantes dispostos em pé. É uma inspeção do progresso na direção da meta. Geram uma melhora na comunicação do time, eliminam outras reuniões, identificam e removem impedimentos para o desenvolvimento, ressaltam e promovem a tomada rápida de decisões.

O processo Scrum em poucas palavras seria divida sua organização em equipes pequenas, multifuncionais e alto – organizadas. Divida o trabalho em uma

lista de entregáveis pequenos e concretos, classificando – os por prioridade e estimando o esforço relativo de cada item. Divida o tempo em iterações de duração fixa, onde seja possível obter códigos potencialmente entregável no final de cada iteração. Otimize o plano de entrega e o processo ao final de cada iteração (KNIBERG; SKARIN, 2009).

2.2.1 Artefatos do Scrum

Scrum utiliza quatro artefatos principais. O backlog do produto, backlog da sprint, burndown de release e um burndown de sprint.

Os requisitos que o time está desenvolvendo estão listados no backlog do produto. O product owner é responsável pelo backlog do produto, por seu conteúdo, por sua disponibilidade e por sua priorização. Este backlog está sempre se evoluindo à medida que o produto e o ambiente evoluem. É dinâmico, está constantemente mudando para identificar o que o produto necessita para ser apropriado, competitivo e útil. É uma lista de todas as características, funções, melhorias e correções de defeitos que constituem as mudanças que serão efetuadas no produto para releases futuras, priorizada e ordenada por risco, valor e necessidade (SCHWABER; SUTHERLAND, 2011).

O gráfico burndown do release registra a soma das estimativas dos esforços restantes do backlog do produto ao longo do tempo. O esforço estimado deve estar em qualquer unidade de medida de trabalho que o time Scrum e a organização tenham decidido usar. As unidades de tempo geralmente são sprints. O product owner mantém o backlog do produto e o burndown do backlog do release atualizados e publicados todo o tempo (SCHWABER; SUTHERLAND, 2011).

O backlog da sprint consiste nas tarefas que o time executa para transformar os itens do backlog do produto em incremento pronto. É elaborado durante as reuniões de planejamento da sprint, sendo todo o trabalho que o time identifica como necessário para alcançar a meta. É um retrato em tempo real

altamente visível do trabalho que o time planeja efetuar durante a sprint (SCHWABER; SUTHERLAND, 2011).

O time pode descobrir que mais ou menos tarefas serão necessárias, ou que determinada tarefa levará mais ou menos tempo do que planejado. À medida que novo trabalho surge, o time adiciona ao backlog da sprint. À medida que se trabalham nas tarefas ou que elas são completadas, as horas estimadas de trabalho restantes são atualizadas. Na visão de Schwaber e Sutherland (2011), somente o time pode modificar o backlog da sprint, modificar seu conteúdo ou as suas estimativas.

O gráfico burndown da sprint indica a quantidade restante de trabalho para que a meta seja atingida. Neste gráfico é exibida a linha de duração de todas as tarefas estimada durantes as reuniões de planejamento, ou seja, a meta da sprint, juntamente com o que está sendo realizado. Com operações matemáticas neste gráfico, acompanhamos diariamente o progresso da sprint e identificamos possíveis riscos, ajudando a tomar decisões o quanto antes (SCHWABER; SUTHERLAND, 2011).

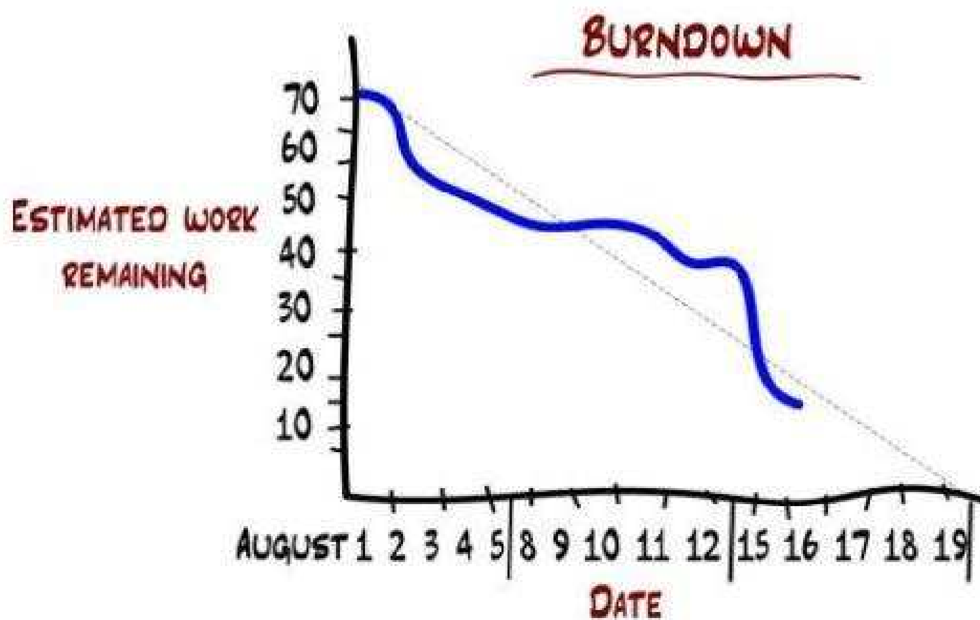

Figura 2: Gráfico burndown. **Fonte:** Schwaber; Sutherland, 2011.

O gráfico deve ser atualizado diariamente após as reuniões diárias e ficar disposto junto no quadro Scrum. Quadro Scrum é onde serão criados os cartões com as tarefas que irão fazer parte de uma Sprint e se tornarem parte de um produto entregável ao final desta. Através de colunas criadas de acordo com a necessidade de cada equipe, irá exibir o andamento de cada item durante o ciclo.

A figura 3 exemplifica o uso de um quadro Scrum.

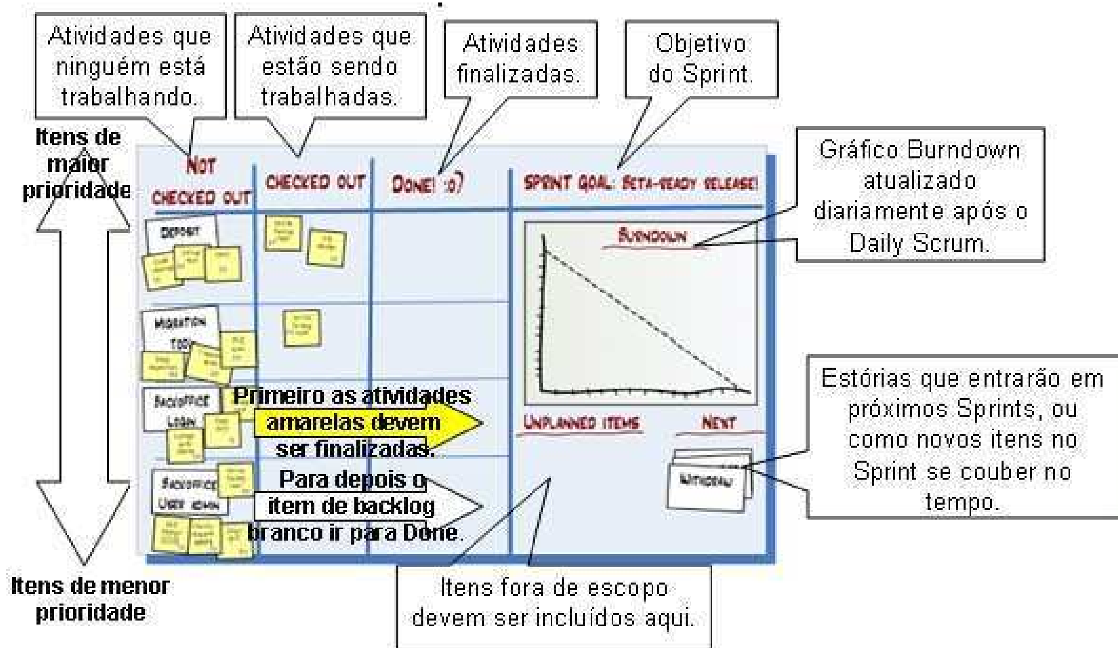

Figura 3: Quadro Scrum. **Fonte**: http://netfeijao.blogspot.com/2008/02/scrum-uma-abordagem-prtica.html

Um quadro Scrum pertence exatamente a uma equipe. Uma equipe é multifuncional, ela contém todas as habilidades necessárias para completar todos os itens contidos na iteração. É visível a qualquer pessoa interessada, mas somente quem faz parte da equipe pode edita – lo, é a ferramenta da equipe para gerenciar o seu compromisso para esta iteração (KNIBERG; SKARIN, 2009).

2.3 KANBAN

Kanban é uma técnica japonesa, integrada no conceito just in time, com origem nos cartões utilizados em fábricas automobilísticas, para solicitarem componentes a uma outra equipe pertencente a uma mesma linha de produção. Conforme Anderson (citado por KNIBERG; SKARIN, 2009, p. 10), Kanban é baseado em uma idéia simples, na qual atividades são limitadas. Uma atividade nova só pode ser iniciada quando outra atividade for liberada, os cartões do Kanban são sinais visuais indicando que novo trabalho pode ser iniciado e que atividade atual não coincide com o limite acordado.

Na visão de Anderson (citado por KNIBERG; SKARIN, 2009, p. 11), Kanban é uma abordagem para mudança gerencial, não é um processo ou ciclo de vida de gerenciamento de software. É uma abordagem para introduzir mudanças em um ciclo de desenvolvimento ou metodologia de gerenciamento.

Utilizado como um mecanismo de controle visual para acompanhar o trabalho à medida que ele fluí através de um quadro branco ou sistema de cartões eletrônicos, o Kanban vai além da transparência, expõe gargalos, filas, variabilidade e desperdício. Tudo que impacta o desempenho da organização em termos de quantidade de trabalho de valor entregue e o tempo de ciclo necessário para que seja entregue. Proporciona a visibilidade sobre os efeitos de suas ações ou falta de ações (Anderson, 2009, p. 12, apud KNIBERG; SKARIN).

Figura 4: Quadro Kanban. **Fonte:** Kniberg; Skarin, 2009.

Com a visibilidade de gargalos, desperdícios, variabilidade e seus impactos o Kanban proporciona a evolução incremental de processos existentes, evolução que é geralmente alinhada a valores ágeis e lean, através de discussões e melhorias identificadas e iniciadas rapidamente pela equipe.

Por ser uma técnica onde o time é que puxa o trabalho na medida de sua capacidade, e não o oposto, quando o trabalho é empurrado à equipe, segundo Anderson (citado por KNIBERG; SKARIN, 2009, p. 14), encoraja o comprometimento, tanto em priorização de trabalho novo quanto na entrega de trabalho existente. Tipicamente, os times irão concordar em uma cadência de priorização para atender as partes interessadas e decidir a próxima coisa em que trabalhar.

A utilização do Kanban é realizada em poucos passos, basta visualizar o fluxo de trabalho e o dividi – lo em partes, escreva cada item em um cartão e insira – o em um quadro branco. Neste quadro, use colunas nomeadas para acompanhar o andamento dos itens, as colunas mais comuns são: a fazer, em desenvolvimento, em teste e feito. Com o quadro pronto, limite o trabalho em progresso (WIP – work in progress), associe limites explícitos para quantos itens podem estar sendo realizados em cada coluna definida no quadro. Acompanhe o tempo de execução da tarefa, otimize o processo para tornar o tempo de execução o menor e o mais previsível possível (KNIBERG; SKARIN, 2009).

O Kanban é uma técnica adaptativa, deixa quase tudo em aberto, possuí como restrição apenas: Visualize seu fluxo de trabalho e limite suas atividades em andamento, com isso se torna altamente poderoso. Muitas equipes misturam e combinam técnicas de diferentes modelos, que produz com uma maior frequência casos de sucesso (KNIBERG; SKARIN, 2009).

Mesmo o Kanban não definindo papéis como Scrum, nada impede que você não possa ou não deva ter um papel de um product owner. Devemos apenas tomar o devido cuidado para não definirmos papéis que não agreguem valor ou entre em conflito com outros elementos do processo (KNIBERG; SKARIN, 2009).

No Kanban, iterações de duração fixa não são prescritas, podemos escolher quando melhorar o processo, planejar e entregar. Pode – se fazer estas

atividades em uma periodicidade regular ou por demanda, sempre que tivermos algo útil. Uma vez que tenhamos os limites das atividades em andamento devidamente estabelecidas, podemos começar a medir e prever o tempo de execução do ciclo, isto é, o tempo médio que cada item leva para cumprir o ciclo, com isso nos permite uma maior fidelidade ao cumprimento de SLA (Service Level Agreements) e realizar planos de entrega mais realistas (KNIBERG; SKARIN, 2009).

Em casos de itens que variam muito de tamanho, podemos considerar limites em pontos definidos dentro da história, ou qualquer outra unidade de medida que facilite a equivalência de tempo entre os itens. Outra alternativa pode ser dividir em itens aproximadamente nos mesmos tamanhos. É mais fácil criar um fluxo regular com planos de entrega realistas, se tivermos itens uniformizados ou medidos por alguma unidade de medida que possa os diferenciar (KNIBERG; SKARIN, 2009).

Em Kanban nenhum gráfico em especial é prescrito, mas nada o impede de usar qualquer tipo de gráfico. O Diagrama de Fluxo Cumulativo é um gráfico que mostra o quanto seu fluxo é regular e como as atividades em andamento afetam o seu lead time. Este diagrama mostra o porquê e por isso aumenta a probabilidade de que a equipe e a gerência colaborem efetivamente. O Diagrama de Fluxo Cumulativo exibe a quantidade atual de trabalho em cada estágio do ciclo (KNIBERG; SKARIN, 2009). A seguir é apresentado um exemplo desse tipo de gráfico:

Figura 5: Gráfico de Fluxo Cumulativo. **Fonte:** Kniberg; Skarin, 2009.

Existem outros gráficos que favorecem o gerenciamento e fornece subsídios para melhoria contínua do processo como:

Tempo de Ciclo: Embora o Diagrama de Fluxo Cumulativo exiba os tempos de ciclo médio, uma análise por item individual pode trazer uma maior confiabilidade podendo ser úteis em termos de previsibilidade. Este gráfico pode lhe fornecer informações de tarefas complexas, tarefas urgentes “apaga incêndios”, problemas de qualidade (BOEG, 2011).

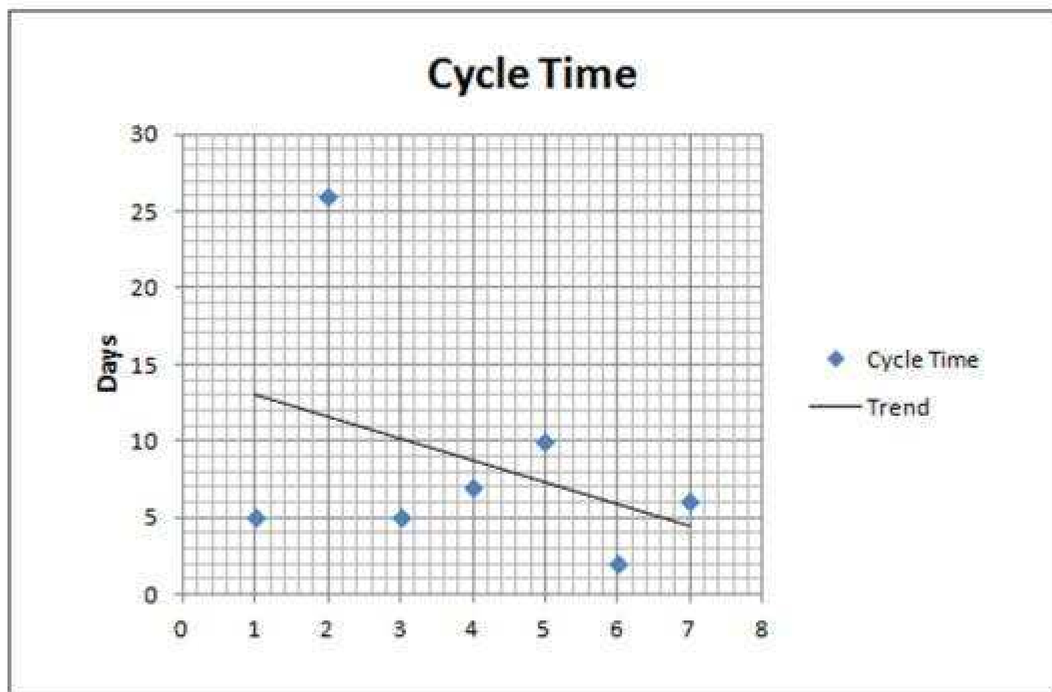

Figura 6: Gráfico Tempo de Ciclo. **Fonte:** Boeg, 2011.

Taxa de Defeitos: Defeitos são extremamente custosos em um projeto, manter um acompanhamento na quantidade de defeitos de um produto é ter certeza que o problema de qualidade não irá sair do alcance. Com gráfico de taxa de defeitos obtemos informações de aumento de número de taxa de defeitos, ou seja, perca de qualidade, como número de erros pode afetar o ciclo de tempo e impacto sobre o Diagrama de Fluxo Cumulativo (BOEG, 2011).

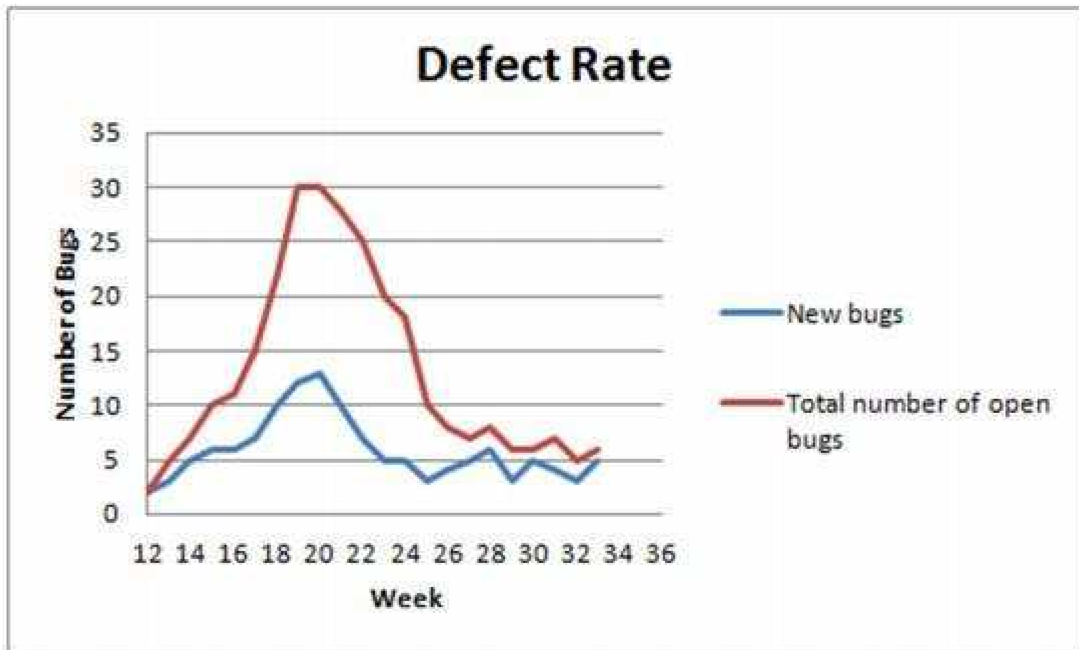

Figura 7: Gráfico Taxa de Defeitos. **Fonte:** Boeg, 2011.

Itens Bloqueados: Equipes terão itens bloqueados por períodos curtos ou longos por várias razões. Itens bloqueados produzem sérios efeitos em longo prazo e por isso devem estar sempre visíveis e serem acompanhados. O gráfico exibe o número de itens bloqueados e a capacidade da equipe em resolver rapidamente estas questões (BOEG, 2011).

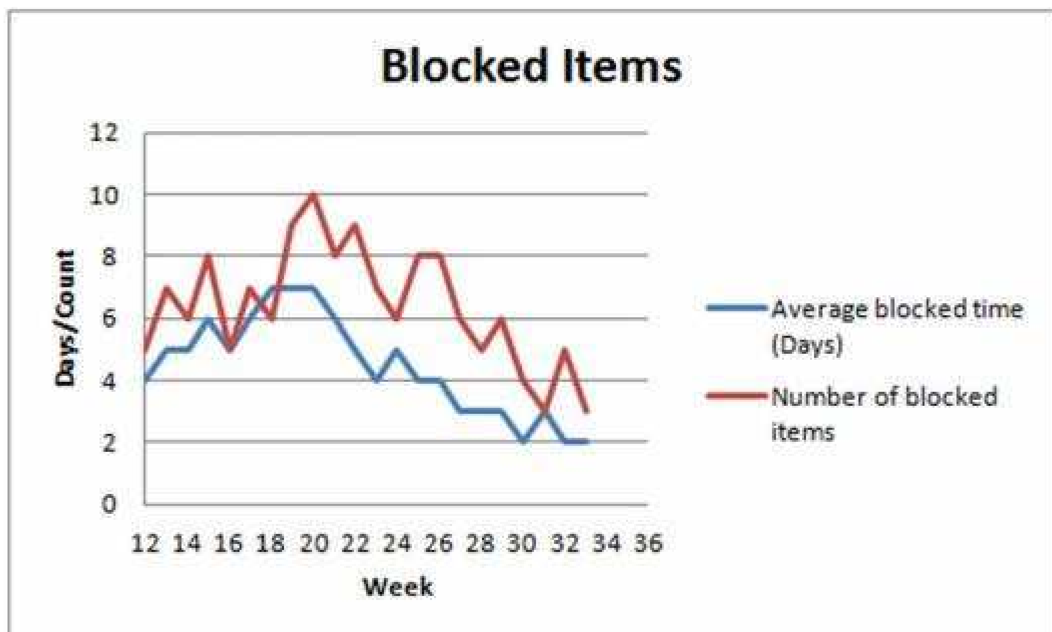

Figura 8: Gráfico Itens Bloqueados. **Fonte:** Boeg, 2011.

Evite achar um lugar separado para cartões bloqueados, estes tendem a serem facilmente esquecidos ou não receberem a devida atenção. É sugerido que um cartão seja identificado como bloqueado no próprio quadro, através de cor ou outro artefato que o diferencie dos demais visualmente (BOEG, 2011).

Também é importante que gráficos permaneçam visualmente juntos com o quadro Kanban, manter os gráficos em separado faz com que os mesmos não recebam a atenção necessária e que até caiam no esquecimento (BOEG, 2011).

3 MODELOS TRADICIONAIS

Segundo os argumentos de Cockburn (2002), modelos prescritivos de processo têm uma deficiência importante: eles esquecem as fragilidades das pessoas que constroem software de computadores. As pessoas exibem grande variedade de estilos de trabalho e diferenças significativas em nível de habilidade, criatividade, regularidade, consistência e espontaneidade. Modelos de processos podem tratar as fraquezas comuns das pessoas com disciplina ou tolerância e a maioria dos modelos prescritivos escolhem a disciplina. Ele afirma "como a consistência em ação é uma fraqueza humana, metodologias de alta disciplina são frágeis" (COCKBURN, 2002).

Se modelos de processos tendem a funcionar, eles precisam fornecer um mecanismo realístico para encorajar a disciplina necessária, ou precisam ser caracterizados de um modo que mostre tolerância com as pessoas que fazem o trabalho de engenharia de software. Invariavelmente práticas tolerantes são mais fáceis de serem adotadas e sustentadas pelo pessoal de software, mas elas podem ser menos produtivas. Como na maioria das situações, negociações devem ser consideradas (COCKBURN, 2002).

Encontramos facilmente diversos modelos e metodologias de desenvolvimento de software que são eficazes no gerenciamento dos mais diversos tipos de projetos e que se caracterizam tanto pela tolerância quanto pela disciplina. Empresas têm se engessado em um modelo e passam grande parte do tempo se preocupando em cumprir passos deste, no que na entrega efetiva do produto. O ideal seria sempre pensarmos no que realmente é preciso e conhecer intimamente as características do produto a ser desenvolvido. Qual é o tamanho do meu projeto? Qual é a quantidade e disponibilidade dos meus recursos? Qual tipo de entrega tenho disponível? Qual a freqüência de alterações que tenho durante o processo de desenvolvimento? Qual complexidade do desenvolvimento? Com isso, qual metodologia irei utilizar? (COCKBURN, 2002).

Modelos tradicionais foram originalmente propostos para organizar o desenvolvimento de software que estava cada vez mais se tornando dinâmico,

complexo e mutável, foram propostos para colocar ordem em um caos. Estruturados e com um roteiro efetivo, serviram como padrão de engenharia de software por algum tempo, no entanto, o desenvolvimento de software e o produto ainda permanecem no limite do caos.

Definidos por Pressman (2006) como um conjunto de atividades, tarefas, marcos e produtos de trabalho que são necessários para realizar a engenharia de software com alta qualidade. Não são modelos perfeitos, mas efetivamente fornecem um roteiro útil para o trabalho de engenharia de software.

Referido por alguns como modelos de processos rigorosos, fornece alta estabilidade, controle e organização a uma simples atividade que pode se tornar bastante caótica.

Com as respostas anteriores, conseguimos definir as necessidades e identificar qual técnica, qual modelo, qual processo nos promove maior desempenho e qual o nível de qualidade e satisfação teremos após a implantação das mesmas.

3.1 RATIONAL UNIFIED PROCESS (RUP)

É uma metodologia de desenvolvimento iterativo, incremental e customizável, com estrutura formal e bem definida. Composta de conceitos, práticas e regras. Utilizada para reduzir o risco sem tornar o desenvolvimento menos eficiente (PISKE, 2003).

De acordo com Piske (2003), provê de uma maneira disciplinada à atribuição de tarefas e responsabilidades dentro de um time de desenvolvimento, incorpora as seis melhores práticas de desenvolvimento de software de acordo com as causas de sucesso apontadas pela indústria de software, a serem executadas durante todo o processo de desenvolvimento:

- Desenvolver iterativamente;
- Gerenciar requisitos;
- Utilizar arquiteturas baseadas em componentes;

- Modelar visualmente;
- Verificação contínua de qualidade;
- Controle de mudanças.

Ainda, segundo Piske (2003), o conceito de melhores práticas se entrelaça com quatro definições, sendo elas:

- Funções: grupos de atividades executadas;
- Disciplinas: áreas de esforço na engenharia de software.
- Atividades: Definições de como é construído e avaliado.
- Objetivos e artefatos: Resultado do trabalho, produzido ou modificado durante o processo.

Nesta metodologia temos uma arquitetura dividida em duas dimensões: horizontal e vertical. Na visão de Taub (2009), horizontal representa o tempo de vida de um projeto, os aspectos do ciclo de vida do processo de engenharia de acordo com o decorrer do projeto. Vertical representa os grupos de atividades lógicas que são realizadas durante o decorrer do tempo. Essa dimensão demonstra o aspecto estático do processo, que será composto por disciplinas, atividades, fluxos, artefatos e papéis.

A figura 9 apresenta os elementos básicos do RUP, exibindo suas dimensões, fases básicas e disciplinas. Cada fase representa um momento distinto dentro do ciclo de vida de um projeto, portanto dão maior ou menor foco em determinadas disciplinas de acordo com a necessidade de cada projeto no decorrer de sua execução. Em cada fase, temos as disciplinas, onde, cada uma possui atividades que serão executadas por um papel distinto no processo e poderão ou não gerar artefatos (TAUB, 2009).

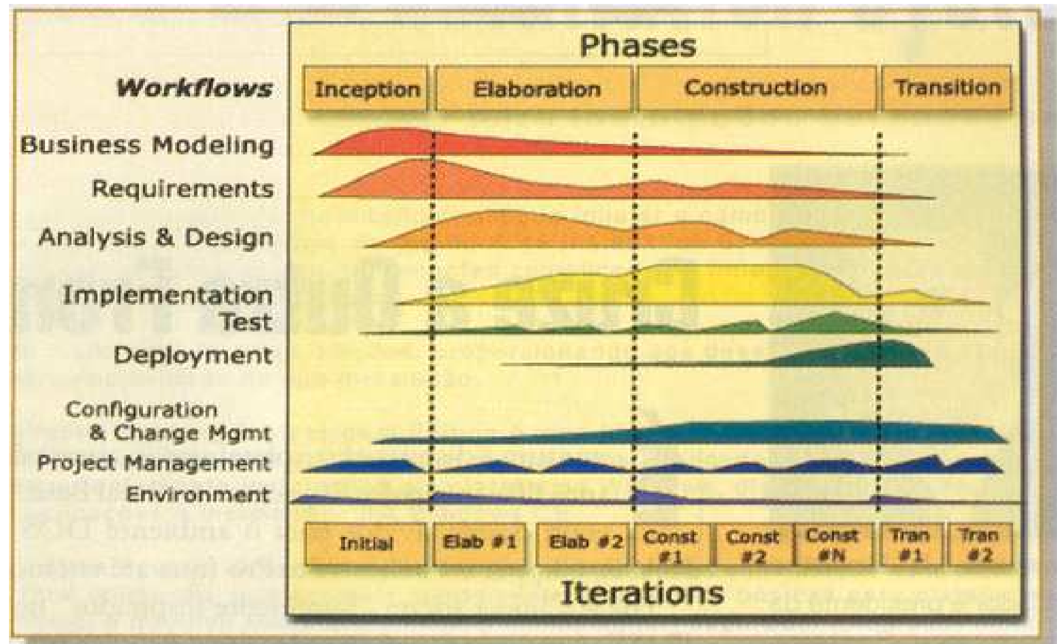

Figura 9: Arquitetura RUP. **Fonte:** Vianna, 2011

Cada fase é composta por uma ou mais iterações, o que se assemelha a um modelo espiral. Segundo Vianna (2011), essas iterações são curtas e abordam algumas poucas funções do sistema. Isto reduz o impacto de mudanças, pois quanto menor o tempo, menor a probabilidade de haver mudanças neste período.

A fase início (inception) é focada em identificar riscos de requisitos e negócios, possui foco em garantir que o projeto é possível e viável. Onde é realizada a discussão do problema, definição do escopo e estimativas. Deve – se nesta fase conseguir dos stakeholders um consenso relacionado aos objetivos do ciclo de vida do projeto. A fase início tem como objetivo estabelecer a visão do projeto, elencar os casos de uso críticos do sistema, exibir e demonstrar uma arquitetura candidata para atender a estes casos de uso, estimar custo e prazos para o projeto, estimar potenciais riscos e preparar o ambiente de suporte para o projeto (TAUB, 2009).

Com o propósito de analisar o domínio do problema, desenvolver o plano de projeto, estabelecer a fundação arquitetural e eliminar os elementos de alto risco, a fase de elaboração (Elaboration) segundo Taub (2009), estabelece uma base sólida para o design e implementação do sistema. A arquitetura deverá considerar os requisitos mais significantes e uma avaliação dos riscos. Os riscos a serem analisados são os de requisitos, tecnológicos, habilidades e políticos.

Considerada como a fase mais critica, Taub (2009) define como objetivos primários a garantia que a arquitetura, requisitos e planejamento do projeto estejam estáveis o suficiente; identificação de todos os riscos arquiteturais do projeto; estabelecimento de uma baseline arquitetural do projeto, demonstração que a arquitetura selecionada suportará os requisitos do sistema através de custo e prazo razoáveis e estabelecimento do ambiente de suporte ao projeto.

Na fase de construção (construction), fecham – se os requisitos. Compreende a modelagem e a fase de desenvolvimento em si, aquela em que o sistema efetivamente é programado, utiliza – se notação UML. A ênfase nesta fase é passarmos do desenvolvimento de propriedade intelectual criado nas fases anteriores para o desenvolvimento de um produto passível de entrega. Em seus objetivos primários, incluem minimização de custos de desenvolvimento, qualidade do produto, versões utilizáveis e desenvolvimento de um produto completo de maneira incremental e iterativa (TAUB, 2009).

A fase de transição (transition) tem como foco a garantia de um produto disponível para usuários finais. É o inicio da implantação do sistema para o usuário final, deve ser utilizado o lançamento de versões beta, operação paralela com o sistema legado, treinamento dos usuários e mantedores do sistema. Ao final desta fase os objetivos do ciclo de vida do projeto deverão ter sido alcançados e o projeto está prestes a ser finalizado (TAUB, 2009).

Durantes as fases temos as disciplinas definidas como workflows, cada uma dessas disciplinas possui atividades que serão executadas por um papel distinto no processo, os objetivos e focos de cada disciplina são caracterizados por Taub (2009) a seguir:

- **Modelagem de Negócio (Business Modeling):** alinhamento dos objetivos e expectativas de todos envolvidos no projeto.

Figura 10: Papéis envolvidos e artefatos produzidos na disciplina modelagem de negócio. **Fonte:** http://www.wthreex.com/rup/portugues/index.htm.

- **Requisitos (Requirements):** Limitar sistema de acordo com requisitos definidos em casos de uso, que são como uma base sólida para estimar custos e esforços. Todos stakeholders do projeto devem compreender e aceitar tudo o que o sistema deverá fazer.

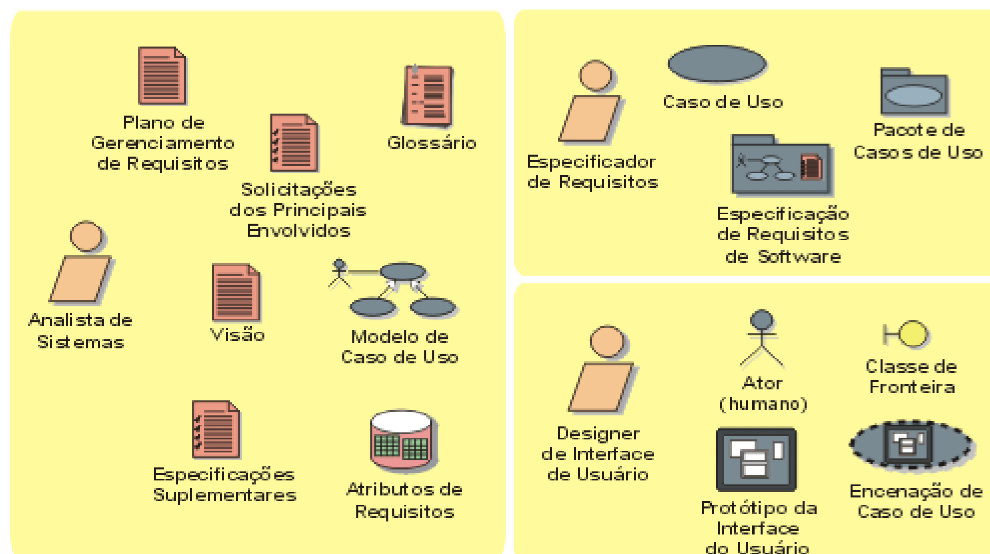

Figura 11: Papéis envolvidos e artefatos produzidos na disciplina requisitos. **Fonte:** http://www.wthreex.com/rup/portugues/index.htm.

.

- **Análise e Design (Analysis & Design):** Requisitos devem ser transformados em desenhos e especificações produzidas para serem seguidas na implementação de cada caso de uso do sistema.

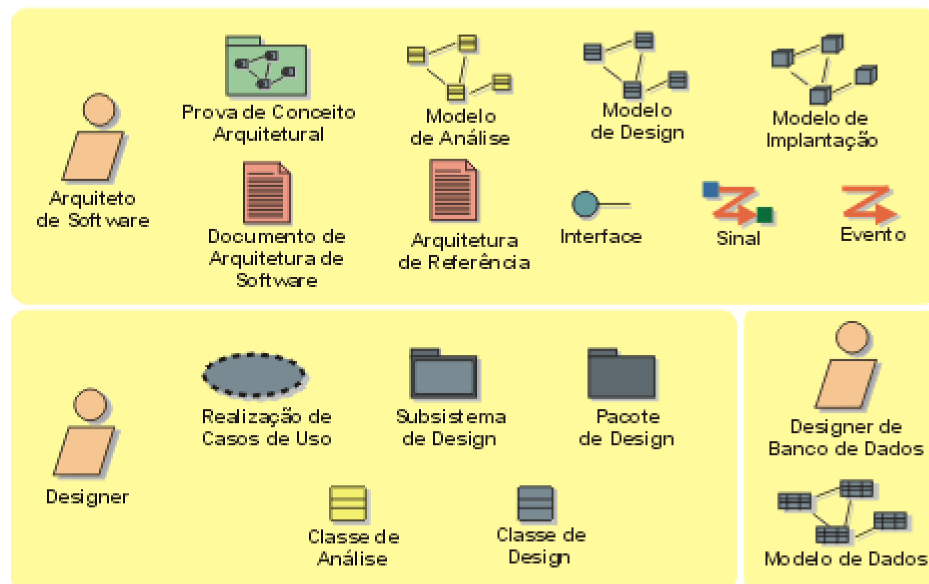

Figura 12: Papéis envolvidos e artefatos produzidos na disciplina de análise e design. **Fonte:** http://www.wthreex.com/rup/portugues/index.htm.

- **Implementação (Implementation):** Transformação de requisitos e modelos definido anteriormente em código fonte utilizável e testado unitariamente.

Figura 13: Papéis envolvidos e artefatos produzidos na disciplina implementação. **Fonte:** http://www.wthreex.com/rup/portugues/index.htm.

- **Testes (Test):** Encontrar, documentar e endereçar os defeitos encontrados na qualidade do software produzido.

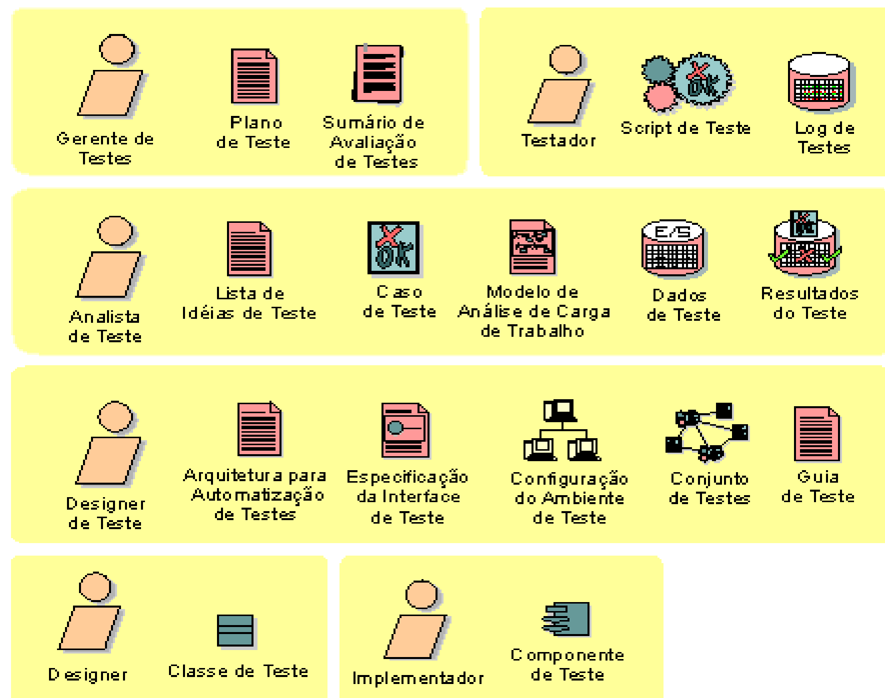

Figura 14: Papéis Os artefatos desenvolvidos como produtos das atividades de teste e avaliação agrupados por papel de responsabilidade. **Fonte:** http://www.wthreex.com/rup/portugues/index.htm.

- **Implantação (Deployment):** Garante que o software produzido estará disponível para os usuários finais.

Figura 15: Os papéis envolvidos e os artefatos produzidos na disciplina Implantação. **Fonte:** http://www.wthreex.com/rup/portugues/index.htm.

- **Gerenciamento de Configuração e Mudança (Configuration & Change Management):** Controla as mudanças e mantém a integridade de cada um dos artefatos produzidos no projeto. Cada

um deste deve ser identificado, auditado e possuir níveis de configuração e manutenção definidos.

Figura 16: Os papéis envolvidos e os artefatos produzidos na disciplina Gerenciamento de Configuração e Mudança. **Fonte:** http://www.wthreex.com/rup/portugues/index.htm**.**

- **Gerenciamento de Projeto (Project Management):** Equilibrar objetivos que competem entre si, gerenciar riscos, monitorar projeto e tratar regras que garantirão a entrega de um produto que irá atender às expectativas de clientes e usuários finais.

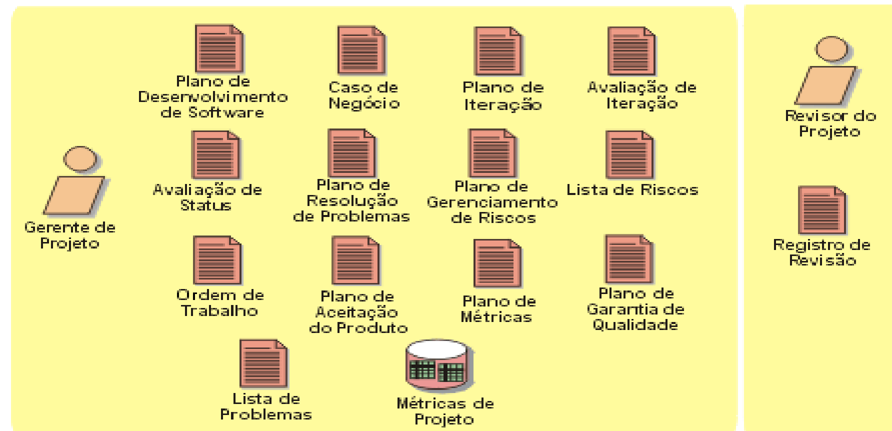

Figura 17: Os papéis envolvidos e os artefatos produzidos na disciplina Gerenciamento de Projeto. **Fonte:** http://www.wthreex.com/rup/portugues/index.htm.

- **Ambiente (Environment):** Garantir ambiente para que o processo e suas atividades possam ser executados.

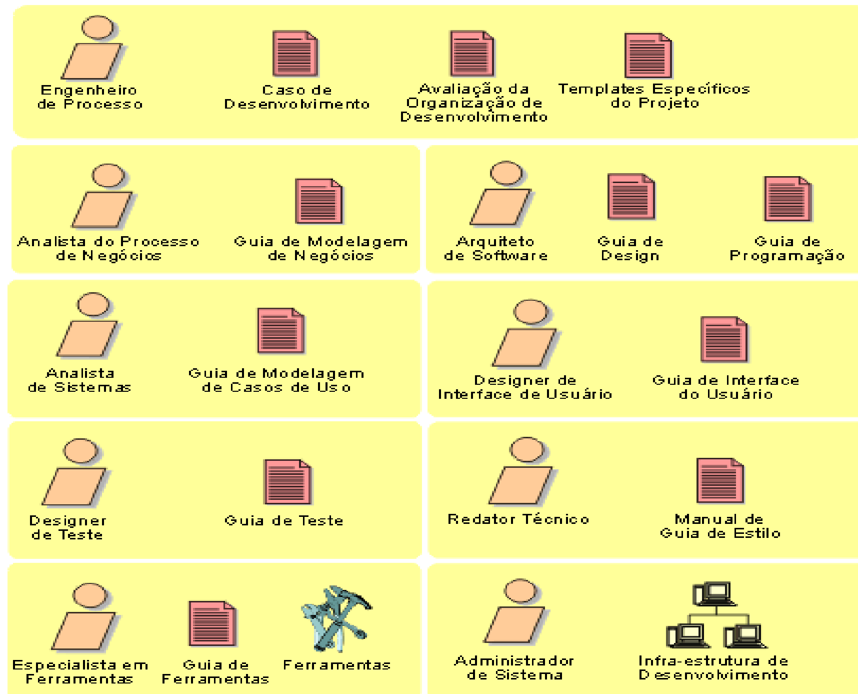

Figura 18: Os papéis envolvidos e os artefatos produzidos na disciplina de Ambiente. **Fonte:** http://www.wthreex.com/rup/portugues/index.htm.

Um papel é uma definição abstrata de um conjunto de atividades executadas e dos respectivos artefatos, desempenhados por uma pessoa ou um grupo de pessoas. Os papéis descrevem como as pessoas se comportam no negócio e quais são as responsabilidades que elas têm (WTHREEX, 2011).

Artefatos são produtos de trabalho final ou intermediário produzido e usado durante os projetos. Usados para capturar e transmitir informações do projeto. Para que o desenvolvimento de um projeto possa ser gerenciado, os artefatos são organizados em disciplinas (WTHREEX, 2011).

Nos projetos de software, garantia da qualidade é o ponto mais comum de falhas, por ser frequentemente não planejada inicialmente e algumas vezes tratadas por equipes e prioridades distintas. O RUP ajuda no planejamento da qualidade e cuida por todo o processo, assumindo que cada membro de uma equipe é responsável pela qualidade. Mudanças em planejamentos são inevitáveis, o RUP define métodos para controlar, rastrear e monitorar estas mudanças (IBM, 2011).

Mais processos nao siguinifica necessariamente melhorias, o mais eficaz é adaptar o processo as necessidades do projeto. O RUP fornece uma coleção de

processos que pode – se personalizar para abordar um conjunto diverso de necessidades de projetos e estilos de desenvolvimento (IBM, 2011).

3.2 ICONIX

Iconix é considerado um modelo puro, prático e simples, porem poderoso. Com um componente de análise e representação sólido e eficaz, é considerado como um processo de desenvolvimento de software. Não é um modelo tão complexo e burocrático como o RUP, não se gera tanta documentação, mas também não deixa a desejar no que se diz respeito à análise e design (MAIA, 2005).

Utilizando a linguagem de modelagem UML como flexível e aberta, podemos se necessário usar qualquer outro recurso da UML para complementar os recursos usados em suas fases. Possuí a característica de rastreabilidade. Mais precisamente, através de rastreabilidade de requisitos, podemos verificar em todas as fases os andamentos de requisitos e até mesmo sua manutenção, não existe um ponto em que o processo pode se distanciar dos requisitos do usuário (MAIA, 2005).

Divididos em dois grandes setores, que podem ser desenvolvidos em paralelo, conforme figura a seguir.

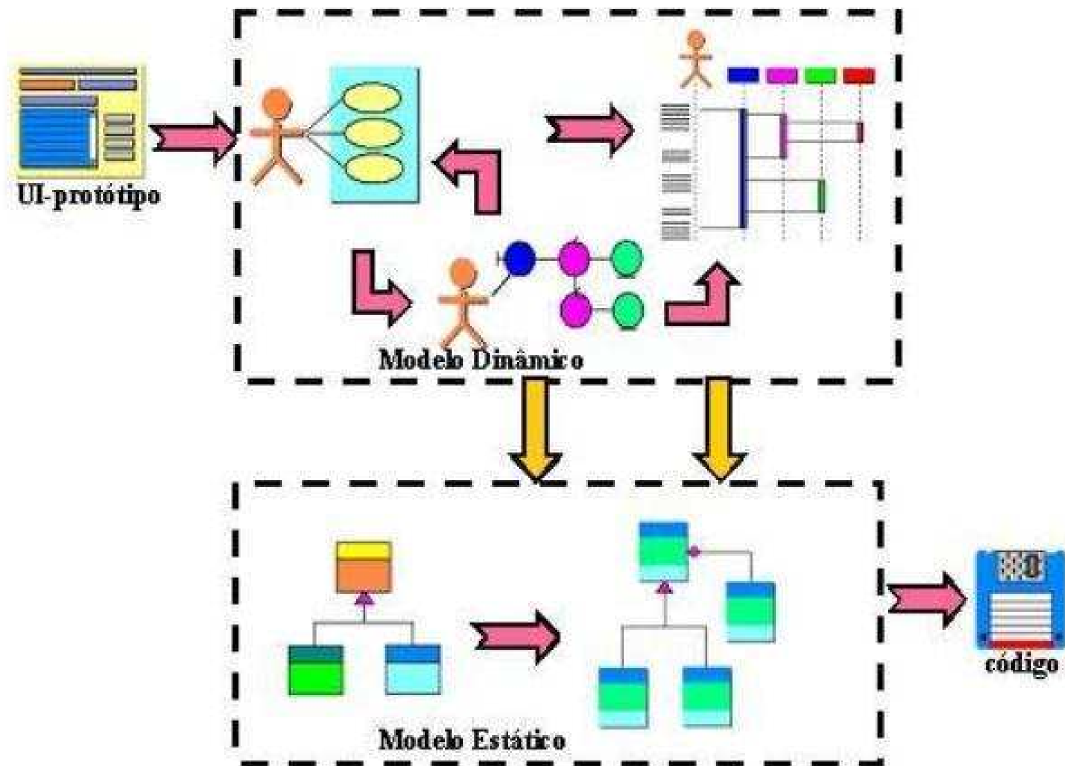

Figura 19: Visão macro do ICONIX. **Fonte:** www.guj.com.br/articles/172.

O modelo estático é formado pelos Diagramas de Domínio e Diagrama de Classes que modelam o funcionamento do sistema sem nenhum dinamismo e interação com o usuário, deve ser refinado incrementalmente durante as iterações sucessivas do modelo dinâmico. Já o modelo dinâmico mostra o usuário interagindo com o sistema através de ações onde o sistema apresenta alguma resposta ao usuário em tempo de execução (MAIA, 2005).

Modelo do Domínio: Na visão de Maia (2005), Modelo de Domínio é uma parte essencial do processo de Iconix que constrói uma porção estática inicial de um modelo que é essencial para dirigir a fase de design a partir de casos de uso. Basicamente, consiste em achar classes, substantivos, verbos que se tornarão objetos, atributos e associações em um Diagrama de Domínio (MAIA, 2005).

Modelo de Casos de Uso: Representam as exigências do usuário em um sistema novo ou alterações baseadas em um sistema existente. Detalha de forma clara e legível toda funcionalidade do sistema. Em sua descrição deve conter o curso normal do caso de uso e todos os cursos alternativos (MAIA, 2005).

Modelo de Análise Robusta: Tem o objetivo de conectar a parte de análise com a parte de projeto, assegurando que a descrição dos casos de uso estejam corretas e dentro da idéia proposta. Focaliza construir um modelo analisando as narrativas de texto de caso de uso, identificando um conjunto de objetos que participarão de cada caso de uso (MAIA, 2005).

Diagrama de Seqüência: Tem como objetivo construir um modelo dinâmico entre o usuário e o sistema. Para tal devemos utilizar os objetos e suas iterações identificadas na análise robusta. É o projeto de como realmente o software irá funcionar, tem como papel principal mostrar o funcionamento do sistema em tempo de execução. Exibe a troca de mensagens entre os objetos de acordo com a descrição do caso de uso (MAIA, 2005).

Modelo de Classe: Representa as funcionalidades do sistema de modo estático. Descreve os tipos de objetos presentes no sistema e os vários tipos de relacionamentos estáticos entre eles. Também mostram as propriedades e as operações de uma classe e as restrições que se aplicam à maneira de como os objetos são conectados (MAIA, 2005).

O processo Iconix trabalha a partir de um protótipo de interface onde se desenvolvem os diagramas de caso de uso baseados nos requisitos levantados. A partir dos diagramas de caso de uso se faz a análise robusta para cada caso de uso. Com os resultados obtidos é possível desenvolver o diagrama de seqüência e posteriormente povoar o modelo de domínio já revisado com métodos e atributos descobertos (MAIA, 2005).

A fase de codificação não é considerada na área de análise do Iconix, esta fase se torna responsável pela correta interpretação dos artefatos produzidos para um produto de software de qualidade (MAIA, 2005).

4 ESTUDO DE CASO

Neste capítulo iremos apresentar o processo de desenvolvimento de software de uma equipe pertencente a uma empresa tradicional de Florianópolis, Santa Catarina, Brasil. Iremos mencionar o nome da empresa através de um nome fictício chamado Software House (SH). O estudo de caso limita se apenas a área de desenvolvimento e testes da equipe, não abrangendo demais áreas como suporte ao cliente, levantamento de requisitos, projeto e negócio.

São citados alguns pontos positivos e negativos de acordo com a visão de alguns membros da equipe. Não se pode assegurar que todos são de fato como descritos, pois não existe fator concreto que demonstre e os comprove. São descritos apenas de acordo com a experiência e sentimento dos envolvidos.

A empresa SH é umas das maiores empresa de sistema de gestão do Brasil, desenvolvendo soluções específicas para segmentos específicos de negócio. Com mais de 1200 clientes no Brasil e no exterior, conta atualmente com cerca de aproximadamente 1000 colaboradores dispostos em várias equipes e segmentos.

A equipe de desenvolvimento estudada é uma equipe que segue o modelo Scrum, composta por dezesseis colaboradores, dispostos em um coordenador de desenvolvimento, um mentor técnico, doze analistas implementadores e quatro analistas de testes. Os implementadores e analistas de testes são divididos em duas equipes: equipe de manutenção evolutiva e equipe de manutenção corretiva. Analistas de projeto, analista de negócio e analista de requisitos tratamos como pertencentes a uma outra área paralela ao desenvolvimento, chamamos de equipe de negócio e, portanto não iremos entrar em detalhes sobre as atividades desta equipe.

O coordenador de desenvolvimento e o mentor técnico ambos atendem as equipes de manutenção corretiva e manutenção evolutiva. São acionados pelas equipes ou pelos desenvolvedores e tem o papel de auxiliar ou fornecer subsídios para que a equipe possa seguir o melhor caminho.

O mentor técnico é responsável por auxiliar os desenvolvedores em atividades complexas de desenvolvimento, em avaliações de especificações, em pré estimativas e outras atividades que a equipe possa encontrar dificuldades.

O coordenador de desenvolvimento é responsável pela gestão das equipes, acompanhamento dos trabalhos realizados, impedimentos técnicos e pessoais. Realiza trabalhos estratégicos para garantir sempre o atendimento das demandas necessárias e propiciar o melhor rendimento de sua equipe.

4.1 EQUIPE DE MANUTENÇÃO EVOLUTIVA

Tudo se inicia com o artefato gerado por uma equipe de negócio, chamado de backlog do produto. Este artefato é composto por especificações onde é descrito o que deve ser feito, servindo em paralelo como uma documentação para o sistema. Através deste backlog de produto é realizada uma priorização de atendimentos pelo product owner, e então, desta priorização temos o backlog da sprint.

As especificações são documentos que contêm detalhes das regras a serem alteradas ou novas funcionalidades a serem criadas, contêm uma solução mais formal do problema. Servem como base para o entendimento e quebra da atividade em tarefas. Contém poucos detalhes técnicos, abrange mais especificamente a parte do negócio, descendo a camada técnica apenas em reestruturação de modelo de dados.

Embora seja um material que gere bastantes subsídios aos desenvolvedores, pode ser melhorada usando outras técnicas ou ferramentas que promovem soluções mais próximas da implementação e posteriormente a rastreabilidade, sendo a falta de rastreabilidade a maior dificuldade para futuras manutenções no sistema.

A equipe de evolução trabalha em sprints seqüenciais com duração de 45 dias cada. Este ciclo é dividido nas seguintes fases, de acordo com a figura 20:

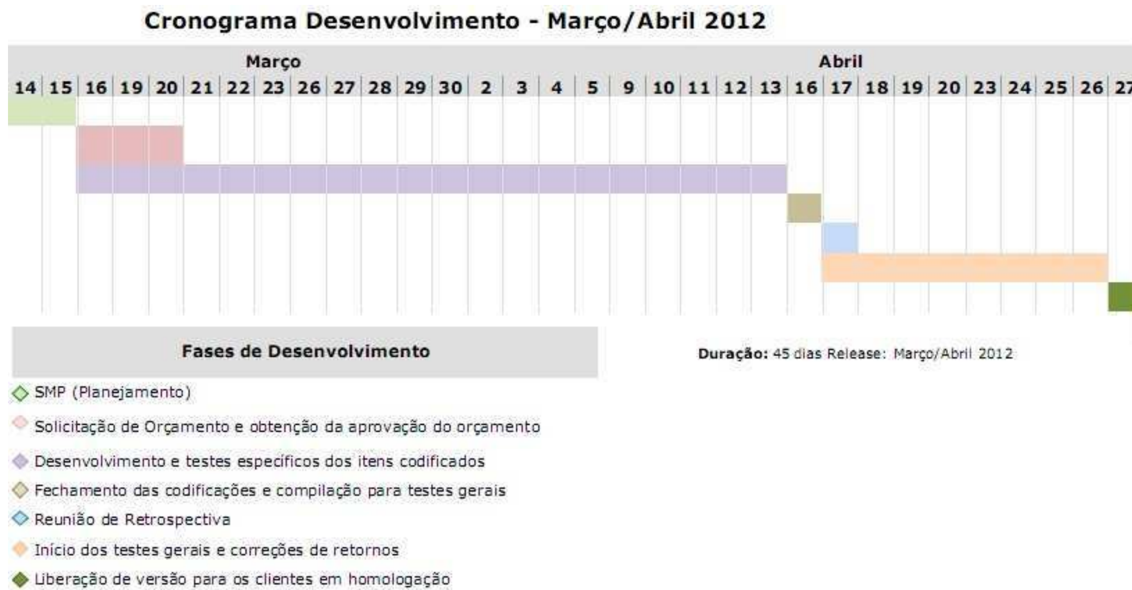

Figura 20: Cronograma Equipe Manutenção Evolutiva**. Fonte:** Cedido pela equipe**.**

4.1.1 Planejamento

Para kniberg (2007, p. 15): “O propósito da reunião de planejamento da sprint é dar à equipe informação suficiente para trabalharem em paz por algumas semanas, e para dar ao product owner confiança suficiente para deixá-los fazerem isto”.

Presidida pelo Scrum master a reunião de planejamento tem uma duração de 2 á 3 dias, com participação obrigatória de todos os desenvolvedores da equipe de evolução e pelo menos um analista de teste. Outras pessoas como mentor técnico, analistas pertencentes a equipe de negócio e até mesmo o product owner podem ser convidados. A equipe realiza os convites por especificação, de acordo com complexidade e especialidade dos membros disponíveis, sendo estes liberados após a conclusão da especificação discutida.

Com o backlog da sprint em mãos e os participantes necessários presentes, começam então o entendimento e o detalhamento de cada especificação em tarefas. A equipe procura sempre dividir em tarefas que não ultrapassem um

tamanho que atinja mais do que 8 horas para ser concluído. Com cada tarefa estimada, é gerado então um valor total que a especificação custará em tempo. Esse mesmo procedimento é feito até que todas as especificações contidas no backlog da sprint sejam contabilizadas.

As estimativas são realizadas de acordo com conhecimento e experiência dos desenvolvedores no assunto. Cada um fala um valor que acha condizente com o que deve ser feito e o maior valor citado é o estipulado para a tarefa.

Com todos os valores das especificações, tem – se então a informação correspondente ao tamanho da sprint. Através de uma operação matemática temos a capacidade da equipe que corresponde ao número de desenvolvedores x número de horas efetivas por dia x número de dias de desenvolvimento. Com esses dois valores, é verificado se o backlog da sprint atende à capacidade da equipe, caso não atenda o product owner é acionado para adicionar ou remover itens neste backlog.

Com essa pratica de reunião de planejamento a equipe consegue obter boas estimativas e um bom entendimento do que irá ser realizado, porém também são identificados alguns pontos que podem ser melhorados, sendo o tempo de reunião o maior fator negativo. Para kniberg (2007), reuniões longas geram cansaço e desestimula a equipe que quer iniciar logo o desenvolvimento, reuniões de planejamento não devem passar de 8 horas. Assim, adicionar uma hora ou agendar uma nova reunião para o dia seguinte, não é uma boa alternativa, uma vez que com a equipe cansada e desestimulada, o aproveitamento real não será o dos melhores.

Devido ao tempo de reunião prolongado, cansaço dos participantes e algumas vezes ansiedade para seu término, algumas tarefas importantes não são identificadas ou até mesmo não são compreendidas por todos os membros da equipe, tornando – se um risco eminente para o planejamento. Em especificações grandes, que demandam um elevado número de horas para sua avaliação, também são encontrados os mesmos problemas. Isso acaba gerando ansiedade da equipe para que um novo assunto seja abordado. Quanto maior a especificação, maior a probabilidade de passar uma tarefa despercebida, gerando assim na fase de desenvolvimento um item não previsto.

4.1.2 Codificação

Com uma duração de aproximadamente 3 semanas, a codificação inicia – se após a reunião de planejamento. As especificações divididas em tarefas são dispostas no quadro Scrum onde os desenvolvedores irão atacar as tarefas de acordo com a ordem das especificações definidas no backlog da sprint. Caso uma especificação prioritária contenha muitas tarefas, vários desenvolvedores podem trabalhar em paralelo na lista de tarefas, para garantir o desenvolvimento completo da especificação.

O quadro Scrum utilizado é uma ferramenta eletrônica, o qual contém as colunas de backlog da sprint, execução, testes, finalizado e impedimento. Na coluna impedimento são colocados as tarefas que por algum motivo não podem ser continuadas, o Scrum master é o responsável por acompanhar esta coluna e dar condições para que a tarefa possa seguir o fluxo.

Tratamos itens não planejados e itens não previstos, criando novas tarefas em destaque, para cada encontrado disponibilizamos no quadro a nova tarefa com uma cor diferente das demais. Através deste quadro é realizado toda a movimentação das tarefas pelas colunas e o gerenciamento da sprint.

O desenvolvedor é responsável pela execução da tarefa, assim como o teste da mesma. A tarefa neste momento é movida da coluna de backlog para coluna em execução. Após o desenvolvimento e os testes do implementador estarem concluídos, a tarefa é movida para coluna de testes. Quando todas as tarefas pertencentes a uma especificação são concluídas, é sinalizado no quadro que a especificação pode ser testada pela equipe de testes alocada na equipe de evolução. A equipe de testes realiza o teste e caso não encontrem inconformidades a tarefa é movida para coluna de finalizado. Quando a equipe de teste encontra problemas, o sinalizador de tarefa pronto para teste é retirado, uma nova tarefa com cor diferente é colocada na coluna backlog e então esta segue o fluxo normal.

Quanto à prática de andamento das tarefas no quadro e a execução das mesmas, não existem grandes problemas. As tarefas são executadas prioritariamente, assim é garantido que a maior prioridade esteja no release. Apesar

de na equipe haver especialistas em determinadas áreas, herança de processos antigos de desenvolvimento praticados na empresa, algumas vezes, de acordo com as prioridades outras pessoas acabam executando tarefas nas quais não são especialistas, gerando assim um aprendizado de regras de negócio e promovendo aos poucos a disseminação do conhecimento. A equipe tem o auxilio de um mentor técnico para casos nos quais o implementador encontre dificuldades, independente delas serem técnicas ou de negócio. O estado de finalizado só é alcançado após o código ser devidamente testado pelo implementador e pela equipe de testes.

Um dos maiores problemas encontrados, que interfere diretamente na equipe de evolução durante a fase de desenvolvimento, são itens não planejados, ou seja, erros emergenciais que acabam interferindo no planejamento ou até mesmo, alterações de requisitos que comprometem a estimativa dada anteriormente. Tais interferências comprometem por completo o sucesso de uma sprint, gerando um esforço extremo para que a sprint volte ao planejado, através de horas adicionais ou inclusão de novos recursos.

Mesmo contando com uma equipe separada para correção de erros, algumas vezes, devido à urgência e particularidade do problema, temos que recorrer ao especialista no assunto, independente se este está alocado na equipe de correção ou evolução.

4.1.3 Geração de versão

Quando todas as tarefas executadas na sprint se encontram na coluna finalizados ou o dia limite para o término do desenvolvimento é alcançado, temos então o fim da fase de desenvolvimento e inicio da fase de testes por todo o sistema, chamamos esta fase de testes gerais na versão. A versão é rotulada e disponibilizada para testes e homologações. Com a duração de um dia, nesta fase todos os desenvolvedores sincronizam tudo o que está pendente, realizam todas as atividades que possam interromper a geração da versão e após isso, temos uma versão rotulada e fechada.

4.1.4 Testes Gerais

Com a duração de 1 a 2 semanas, a fase testes gerais compreende os testes por todo o sistema, focando diferentes clientes para que a versão possa ser liberada com o máximo de qualidade possível. Erros encontrados pela equipe de testes são corrigidos em paralelo pelos desenvolvedores participantes da sprint até que a versão se torne estável.

Como fator negativo nesta etapa, tem - se a grande quantidade de erros encontrados, fazendo com que a energia da equipe se volte para as correções. Esses erros são oriundos de vários fatores como: sincronizações de correções feitas em versões anteriores e que não foram testadas na versão atual em desenvolvimento, impacto de alterações ou novas funcionalidades nas demais regras existentes e algumas vezes erros não identificados na própria especificação. O grande número de erros muitas vezes ultrapassa o tempo limite para a correção, gerando problema para o inicio em seqüência de uma próxima sprint.

4.1.5 Retrospectiva

Após a geração da versão, juntamente com o inicio da fase de testes gerais, ocorre à reunião de retrospectiva da sprint, com uma duração de aproximadamente 4 horas. A reunião de retrospectiva conta com a participação de todos os desenvolvedores podendo se estender o convite a qualquer outro interessado. Na visão de Kniberg (2007), a coisa mais importante em uma retrospectiva é assegurar – lhe que ela aconteça.

Com o inicio da reunião os participantes expõem de acordo com seu sentimento o que deu errado na sprint, o que pode ser melhor e as ações que podem ser tomadas para melhorar a próxima iteração. Nas reuniões de retrospectiva atuais, não existe uma ordem definida de exposição de idéias, ou seja, não existe um circulo em que cada membro expõe os pontos negativos, positivos e ações na

sua visão, os pontos são citados aleatoriamente entre os participantes sobre cada questão abordada.

Todos os pontos são anotados e servem com referencia para próxima iteração. Quanto à reunião de retrospectiva, a equipe cumpre bem o objetivo e o tempo recomendado, o que podemos levar como ponto negativo é o aproveitamento escasso de tudo o que foi discutido como lição aprendida para a próxima sprint. Muitas vezes, nem todas as ações recomendadas são seguidas, algumas caem no esquecimento fazendo com que alguns erros se tornem rotineiros.

4.1.6 Liberação

Após todas as etapas citadas anteriormente, temos então a liberação da versão para o cliente. A partir deste ponto, entram em cena outras equipes e consequentemente o término da sprint de desenvolvimento.

4.1.7 Outras práticas e artefatos

Como prática definida como sendo essencial pelo Scrum, a equipe realiza as reuniões diárias, nas quais são tratados assuntos pertinentes a sprint. Apenas os desenvolvedores são os participantes e cada um comenta o que está fazendo, o que fez da ultima reunião diária até o momento e quais são as próximas atividades que irá realizar até a próxima reunião diária. Esta reunião ocorre sempre no mesmo horário e no mesmo local, com duração de aproximadamente 15 minutos. Algumas vezes outros assuntos são abordados, fugindo até mesmo do objetivo da reunião, porem já é identificado pelo Scrum master como uma melhoria a ser tomada nas próximas reuniões.

Não se pratica a Reunião de review, como é definida pelo Scrum. No final da sprint toda equipe deve ser reunida juntamente com o product owner e uma apresentação de todos os itens terminados é realizada. Esta apresentação ocorre

por cada implementador ao término de sua atividade, o implementador responsável convida o product owner e o especificador, apresenta em sua própria maquina o resultado. Uma apresentação como essa, menos formal e não no final da sprint, pode acontecer de gerar uma sprint com muitas atividades 99% prontas, a apresentação no final força que apenas atividades totalmente concluídas possam ser apresentadas, não irá restar mais tempo para ajustes. Outro fator que pode ocorrer é possíveis ajustes pedido pelo product owner ou especificador, já que ainda resta tempo de desenvolvimento, este pode identificar na apresentação melhorias para o requisito.

Segundo Kniberg (2007), uma apresentação de sprint bem feita, apesar de parecer um desperdício de tempo e de recursos, obtemos ganhos como:

- A equipe ganha crédito por suas realizações.
- Outros aprendem o que sua equipe está fazendo.
- A apresentação atrai feedback vital dos stakeholders.
- Apresentações são (ou deveriam ser) um evento social nas quais equipes diferentes podem interagir umas com as outras e discutir seu trabalho.
- Sem apresentações, nós continuamos recebendo imensas pilhas de atividades 99% prontas.

Como o principal artefato de acompanhamento de andamento da sprint, a equipe atualiza e mantém disponível diariamente o gráfico burndown.

4.2 EQUIPE DE MANUTENÇÃO CORRETIVA

Assim como a equipe de evolução inicia suas atividades com uma lista de especificações priorizadas, a equipe de manutenção corretiva inicia com uma lista de erros priorizados. A equipe utiliza o Kanban para gerenciar o trabalho.

A equipe de manutenção corretiva, por utilizar Kanban não possui um Scrum master, mas tem o papel de um responsável pela avaliação dos itens de erro e pela monitoração para que os conceitos Kanban sejam seguidos.

Utilizam um sistema de controle interno, que substitui o quadro físico Kanban, onde cada item de erro é representado por diversos status, através de uma analogia com o quadro Kanban, todos os status correspondem a uma coluna. Seguindo esta lógica, existe as colunas *aguardando avaliação desenvolvimento*, *aguardando implementação*, *em implementação*, *aguardando outros*, *aguardando geração de versão*, *aguardando testes, em testes* e *encerrado*.

A lista de erros priorizados é formada pela equipe de suporte ao cliente. Identificado um erro no cliente, a equipe suporte realiza o primeiro contato, confirma a situação de erro e então prioriza este item na lista. Quando um novo item é priorizado, o responsável pelas avaliações na equipe de correção é notificado e uma atividade pendente fica em seu nome com o status de *aguardando avaliação desenvolvimento*, temos então analogamente a inserção de um item no quadro. Após avaliação, é realizada uma estimativa prévia do item baseada em experiência e históricos de erros semelhantes, se necessário ocorre uma repriorizacao junto aos analistas de suporte e então o status é atualizado para *aguardando implementação*. A partir deste momento, o item está disponível para que os desenvolvedores membros da equipe de manutenção corretiva possam executa – lo.

Com os itens avaliados e ordenados, os desenvolvedores vão pegando sempre o de maior prioridade, passando para o status de *em implementação* e realizando a correção necessária. Após a correção o status deste item é alterado para *aguardando geração de versão*. Geração de versão siguinifica que os analistas de testes alocados na equipe devem gerar uma versão para que testes sejam realizados. A versão é gerada por um analista previamente definido e então o item é alterado para *aguardando testes*, assim qualquer analista de teste pode ir até o quadro, pegar a tarefa de maior prioridade e realizar os testes passando o status para *em testes*. Caso o teste executado esteja de acordo com o esperado o item é encerrado e caso se encontre inconformidades o item é passado para *aguardando implementação* e volta a percorrer o mesmo fluxo.

O status de *aguardando outros* é semelhante a um impedimento, depende de terceiros para que o item possa continuar a caminhar normalmente pelo fluxo. O responsável pelas avaliações na equipe deve cuidar e monitorar estes itens.

Seguindo a regra Kanban, temos um WIP (Work in progress) limitado a uma atividade por desenvolvedor ou analista de testes. Mas atualmente esta limitação é problemática e não é tão seguida, provocando alguns efeitos colaterais indesejados. Mesmo com WIP definido, algumas vezes, devido ao dinamismo das prioridades e inserção de novas atividades no quadro, alguns desenvolvedores param a sua correção atual para dar vazão ao novo item de maior prioridade, fazendo com que algumas vezes se acumulem itens em implementação. O efeito colateral desta abordagem é que algumas vezes o primeiro item que estava sendo desenvolvido acaba ficando por mais tempo do que deveria em desenvolvimento, sendo que se fosse seguido à regra da limitação de trabalho, provavelmente já estaria no estado de encerrado. Isso faz que muitas estimativas definidas no início sejam quebradas, se tornando problemáticas em contratos SLA.

A equipe de correção atualmente não realiza nenhum tipo de reunião pré definida. Iniciou – se até com reuniões diárias, provindas do Scrum, porem não foram identificados ganhos reais e as reuniões acabaram sendo descontinuadas. Reuniões na equipe acontecem aleatoriamente, apenas quando é identificada a necessidade de abordar algum assunto com todos os membros da equipe.

A equipe de correção também não utiliza nenhum tipo de artefato gráfico para demonstrar ou gerenciar as atividades. Não possui indicadores de fácil acesso ou manuseio, todos os dados são registrados no sistema de controle, mas as consultas e visualizações disponíveis prejudicam o gerenciamento.

A equipe de correção não participa de nenhuma das reuniões praticadas pela equipe de evolução. A troca de informação entre as equipes ocorre através de rodízios de alguns membros sempre ao término de uma sprint.

5 PROPOSTA DE MELHORIA SUGERIDA

Neste capítulo será apresentado e sugerido técnicas ou adaptações para melhorar a qualidade e desenvolvimento da equipe pertencente à empresa SH. Algumas sugestões já estão sendo aplicadas à equipe em fase inicial e até o momento sem resultados comprovados, apenas através de feeling e sentimentos dos participantes. Entre as sugestões em fase inicial de aplicação, nem todas são fruto do estudo desta pesquisa, algumas chegaram até a equipe através de consultores externos a empresa, porem todas serão descritas seguindo os conceitos encontrados nos materiais pesquisados.

É notável através do estudo realizado no capitulo 4 que a equipe adotou o conceito Scrum, realizou as adaptações de acordo com sua realidade e tem praticado os artefatos principais, entretanto, sabemos e como o próprio desenvolvimento ágil define, o melhoramento contínuo sempre deverá ocorrer a cada iteração. As sugestões citadas estarão ligadas justamente ao melhoramento contínuo, não irá se tratar de mudanças de processo ou metodologia, mas sim uma busca de qualidade através da inclusão de novos artefatos, técnicas ou ferramentas que são usadas por outras práticas de engenharia de software, e que se relacionam com as dificuldades vivenciadas pela equipe de desenvolvimento da empresa SH.

Para uma equipe Scrum ser realmente bem sucedida, ela deve ir além da adoção das partes básicas e altamente visíveis do Scrum e se comprometer com mudanças reais na maneira de abordar o trabalho de criar um produto. Equipes com resultados sólidos e produtivos, podem obter melhorias substanciais quando realizam mudanças em suas práticas técnicas. O Scrum não prescreve práticas de engenharia específicas, isso seria incompatível com sua filosofia básica, o que o Scrum faz é exigir que as equipes entreguem um código de alta qualidade potencialmente funcionando no fim de cada sprint. Se as equipes puderem fazer isso sem mudar suas práticas técnicas, que assim seja. A maioria das equipes, no entanto, descobrem e adotam novas práticas técnicas porque isso facilita muito a realização de seus objetivos (COHN, 2011).

Seguindo a lógica de trabalho desempenhada pela equipe de manutenção evolutiva, o inicio ocorre com as especificações, mesmo as especificações não sendo desenvolvidas pela equipe estudada, Cohn (2011) defini que para o sucesso na utilização do Scrum ser alcançado, todas as equipes devem estar alinhadas com as práticas Scrum e trabalharem seguindo suas práticas.

Na empresa SH ambas as equipes utilizam o Scrum, no entanto em especial a equipe de negócio teria um ganho acrescentando técnicas do RUP e ICONIX em sua documentação sem abandonar o princípio ágil do Scrum. Inicialmente devido à complexidade de gerar mais documentos e uma maior burocracia, o tempo utilizado seria contra qualquer princípio ágil, mas futuramente com o reaproveitamento e rastreabilidade oferecidos na utilização destes conceitos, a agilidade voltaria a ter seu ganho. Grande parte do esforço de uma empresa em se tornar ágil é encontrar o equilíbrio apropriado entre antecipação e adaptação, sem a necessidade de abandonar etapas que fornecem ganhos.

Em uma reunião de planejamento realizada pela equipe de negócio, as atividades deveriam ser divididas igualmente como ocorre em uma mesma reunião de planejamento pela equipe de desenvolvimento. Especificações grandes podem gerar erros de planejamento pela equipe e propiciar até mesmo os erros discutidos no capítulo anterior na equipe de evolução. Poderiam ser divididas em pequenas especificações que contenham algo entregável no final de sua codificação, e não necessariamente em toda a funcionalidade do sistema, para que futuramente juntando, teríamos a funcionalidade completa.

O Scrum não extingue a documentação, apenas encoraja a agilidade. Documentações e acesso fácil a esta para futuras alterações promovem agilidade. Seguindo alguns diagramas realizados no RUP e ICONIX, como diagrama de casos de uso, diagramas de seqüência e de interação, ambas as equipes teriam maior chance de identificar e tratar riscos. A equipe de especificadores realiza especificações sem detalhamentos técnicos, gerando risco, como exemplo um impedimento técnico com outra regra existente no sistema ou até mesmo um impedimento de tecnologia que algumas vezes só é identificado na fase de codificação. Dessa maneira o impacto do risco é propagado tanto na equipe de especificação, que terá que alterar a documentação e propor nova solução, quanto

na equipe de desenvolvimento, que tem seu planejamento quebrado devido a incrementos ou alterações de tarefas não planejadas.

Com a redução de tamanho e maior detalhamento em nível técnico das especificações, a reunião de planejamento da equipe de evolução que hoje possuí a duração de 2 a 3 dias sofreria uma redução consequentemente em seu tempo. Não nos referindo a esta redução de tamanho como sendo o principal causador do alto tempo de planejamento, a equipe também deve se adequar e procurar maior organização para reduzir o tempo. Uma divisão da equipe de manutenção evolutiva, também ajudaria a reduzir o tempo e obter alguns outros ganhos.

Equipes grandes têm a vantagem de contar com membros experientes, habilidades e especialidades das mais variadas, não correndo o risco de perda de uma pessoa chave. Por outro lado, há mais vantagens em equipes pequenas, como as citadas por Cohn (2011) a seguir:

Há menos ociosidade social: Tendência das pessoas se esforçarem menos quando acham que outras pessoas assumirão responsabilidades.

Interação construtiva: A probabilidade de ocorrer é maior, em equipes com mais de 10 pessoas existe a dificuldade para estabelecer sentimentos de confiança, responsabilidade mútua e coesão.

Menos tempo gasto em coordenação: Equipes pequenas gastam menos tempo coordenando os esforços de seus membros.

Ninguém fica para trás: Em equipes grandes, há uma participação menor em atividades e discussões em grupo. Da mesma forma, a disparidade no nível de participação entre os membros da equipe aumenta.

Maior satisfação aos membros: Contribuições são mais visíveis e significativas.

É menos provável que ocorra uma especialização: Em grandes projetos, pessoas têm maior probabilidade de desempenharem papéis distintos.

Conforme Mark e Hertel (2003, p. 7, apud Cohn, 2011, p. 202) segue a conclusão de uma pesquisa sobre tamanho de equipes:

> Membros de equipes menores participavam mais ativamente de sua equipe; eram mais comprometidos com ela; tinham mais consciência dos objetos da equipe; entendiam melhor as personalidades, papéis e estilos de comunicação de outros membros da equipe; e relatavam níveis mais altos de afinidade. Os dados também mostram que equipes maiores são mais cuidadosas na preparação de pautas, reuniões se comparadas com equipes menores.

A equipe de manutenção evolutiva conta com aproximadamente 8 integrantes em uma sprint, mesmo esse não sendo um número considerado como grande por alguns autores de Scrum, não é um número pequeno que não possa ser dividido em duas equipes. E seguindo as vantagens citadas acima, é notável um ganho para todos os integrantes da equipe de desenvolvimento da empresa SH. É constatado pelos participantes da equipe algumas das vantagens citadas acima no formato atual, mas também as vantagens seriam mais evidenciadas com a divisão em duas equipes menores de evolução. É fato que no formato atual o Scrum master despende um maior tempo em coordenação do que em codificação e que a equipe erroneamente assimila o mesmo como o responsável pelo sucesso da sprint.

Duas equipes evolutivas, gerariam as vantagens citadas, bem como um escopo menor para a sprint, devido a capacidade da equipe ser dividida, havendo consequentemente então uma redução do tempo de planejamento.

Analisando a fase de codificação, poderíamos acrescentar técnicas combinadas do Scrum e XP, de acordo com Kniberg (2007), o Scrum é focado nas práticas de gerenciamento e organização, enquanto o XP da mais atenção às tarefas de programação, um complementa o outro.

Seguindo abordagem XP, poderíamos utilizar e obtermos ganho em nossa fase de desenvolvimento utilizando técnicas como desenvolvimento baseado em testes, refatoração, integração contínua e programação em pares, ambos adequados a nossa realidade.

O teste do implementador funciona hoje da seguinte maneira: os implementadores selecionam uma parte do programa para manipular, escrevem o código, tentam compilar, corrigem todos os erros de compilação, percorrem o código em um depurador e então fazem tudo de novo. Em uma abordagem TDD (Test-

Driven Development), na visão de Cohn (2011), o trabalho é realizado em ciclos muito curtos de identificação e automatização de teste de falha, criação de um código suficiente para passar neste teste e limpeza do código conforme necessário antes de começar tudo de novo. Este ciclo é repetido em curto espaço de tempo, em vez de após algumas horas.

O TDD garante que nenhum código não testado entre no sistema, a abordagem atual supostamente até garante o mesmo, mas não é o que acontece de fato, quando os implementadores terminam de desenvolver um requisito a pressão ou ansiedade para iniciar um novo pode ser grande, logo escrevem testes apenas para um subconjunto da nova funcionalidade.

Também pode ser considerado como uma prática de projeto, afinal, os testes que o implementador escreve e a ordem em que são escritos guiam o projeto de desenvolvimento de um requisito. Não há nada no TDD que o trata como prática de projetos pequenos, projetos complexos também podem utilizar o TDD, adaptando – se à arquitetura. Ser ágil é encontrar o equilíbrio correto entre a antecipação e adaptação (COHN, 2011).

Segundo George e Williams (2003, apud COHN, 2011, p. 180), usar o TDD demora 15% a mais do que não usá – lo. Mas também segundo, Sanchez, Williams e Maximilen (2007, apud COHN, 2011, p. 180), o número de erros encontrados diminui entre 20% e 38%. Portanto o TDD pode demorar mais inicialmente, mas o tempo será devolvido à equipe na forma de menos correção e manutenção.

Apesar das vantagens, é difícil de aprender o TDD, o melhor é um implementador que não conhece aprender trabalhando de maneira pareada com um que saiba, ou então que os dois aprendam em conjunto. Se estiver estagnado de testes manuais e quiser automatizar, comece construindo coisas que façam com que os testes de regressão fiquem mais rápidos, depois considere automatizar o teste efetivo (KNIBERG, 2007).

Sabemos que a programação em pares não deve ser forçada, as pessoas devem querer experimentar por si mesmas. Porém, para obter as suas vantagens é preciso incentiva – las e fornecer as ferramentas corretas. Algumas pessoas não se

sentem confortáveis e outras não gostam até experimentarem, por isso encorajar e incentivar é a melhor forma. Não deve ser aplicada diariamente, se torna cansativa, os pares devem ser trocados (KNIBERG, 2007).

Com essa informação, é recomendável que a empresa estimule e incentive a programação em pares na equipe. Que permitam aos implementadores estimarem tempo em suas reuniões de planejamento para utilizar este artefato, não precisa ser obrigatoriamente em todas as atividades da sprint, deixe que os participantes escolham as atividades que achar necessário e com os parceiros que se sentirem confortáveis. Em curto prazo terá um custo maior para a empresa, mas este custo adicional será compensado com a maior qualidade levando a menores custos de manutenção posteriormente. Se o problema é tempo, então precisamos mais ainda de programação em pares, a previsão tem maior possibilidade de sair conforme planejado e com qualidade notável. Se o problema for complexo e o implementador quiser pensar no problema sozinho, os parceiros podem se separar para pensarem, posteriormente se unem e compartilham as idéias sugeridas.

De acordo com Kniberg (2007), a programação em pares aumenta o foco equipe, nivela o conhecimento da equipe, produz o sentimento de código coletivo, é eficaz na solução de problemas difíceis ou em partes desconhecidas do sistema, bem como, ajuda na aplicação de outras práticas.

Uma alternativa válida para iniciar a programação em par e ao mesmo tempo estimular é a revisão de código. Também promove a disseminação do conhecimento e aumenta a qualidade. Na Revisão de código um implementador revisa o trabalho realizado pelo outro, opinando em design, layout, estruturas e etc. Erros podem ser identificados e melhorias podem ser sugeridas através desta técnica. No caso da equipe de evolução, como foi sugerido neste capítulo a divisão da mesma, os implementadores poderiam estar realizando revisão de código em membros de outra equipe e vice – versa, assim perderíamos o prejuízo de cada uma das equipes não saber o que a outra equipe está produzindo, uma vez que ambas pertence a uma equipe global de evolução.

No inicio de uso da revisão de código, pode haver um maior tempo gasto para finalização da atividade, existe uma fase de adaptação em que o retorno pelo

revisor acaba sendo em maior número. Mas ao decorrer do tempo, a tendência de retornos é diminuir muito.

Para o sucesso do TDD, um fator crucial é a refatoração. Esta ajuda a impedir que o código se deteriore. É recomendado que a refatoração ocorra constantemente durante a codificação e não ao final de período ou em tempos definidos. Se uma refatoração precisar de muitas horas homens, então é recomendado que esta seja um item de backlog para uma próxima sprint. Se a equipe for identificando muitos trechos que necessitam de refatoração nos códigos já existentes, a equipe pode criar uma lista com itens pendentes de refatoração.

Com uma lista de itens de refatoração, poderíamos criar o que chamamos de sprint melhorias, em que a cada período definido com o product owner, uma sprint de melhorias possa ser executada. Essa sprint pode ocorrer em uma iteração menor do que a definida de 45 dias, poderia como exemplo ter uma duração de 15 dias, para que os clientes também não fiquem por muito tempo sem receber uma nova funcionalidade do produto e ocorrendo a cada 3 ou 4 meses, ficando esse período a critério do product owner e sempre ajustado de acordo com o crescimento e importância da lista de refatoração.

Essa abordagem não geraria prejuízos à empresa em termos de faturamentos, seria mais um fator que diminuiria o alto custo em manutenção. Considere a definição clássica à medida que um software é modificado na visão de Brooks (1995, citado por COHN, 2011, p. 180), segue a conclusão de uma pesquisa sobre tamanho de equipes:

> Todos os reparos tendem a destruir a estrutura e aumentar a entropia e a desordem do sistema. Cada vez menos os esforços são gastos na correção de falhas no projeto original; cada vez mais são gastos na correção de falhas introduzidas por correções anteriores. À medida que o tempo passa, o sistema fica cada vez menos ordenado. Cedo ou tarde, as correções param de surtir efeito. Cada avanço vem acompanhado de um retrocesso. Embora, em princípio, usável definitivamente, o sistema esgota – se como base para o progresso.

E para que a abordagem TDD faça sentido e que a equipe possa ser realmente ágil, ferramentas de integração contínua devem ser disponibilizadas. A única sobrecarga para esta técnica está em instalar e manter o ambiente computacional no ambiente de servidor de build, mas o retorno será garantido economizando tempo em problemas de integração.

Para sistemas complexos, em que é necessário varias horas para execução de testes de integração por completo, Cohn (2011), sugere dividir o conjunto de testes. O teste é dividido em dois: um inicial e um teste completo. O primeiro é executado após o primeiro check in e incluí todos os scripts de teste da sprint atual e um conjunto de scripts de etapas anteriores. O segundo é executado uma vez a cada intervalo de tempo e incluí todos os scripts de testes de todas as etapas.

Com o uso das técnicas XP, citadas acima no ciclo Scrum trabalhado pela equipe de manutenção evolutiva, os maiores problemas encontrados pela equipe relatados no capítulo 4, se não extinguidos, pelos menos teriam uma grande melhoria. Melhorias em relação à qualidade da versão, que atualmente gera um grande número de erros e itens não planejados, diminuição de especialistas na equipe e consequentemente um maior cumprimento nos planejamentos definidos.

O próximo passo de melhorias propostas aborda a reunião de retrospectiva. Como vimos esta não apresenta maiores problemas, mas segue algumas sugestões que podem torna - lá cada vez melhor:

- Reunião de retrospectiva não diz respeito a apenas como uma equipe pode fazer um bom trabalho em uma próxima sprint. É indicado que uma pessoa, Scrum master ou outro membro, participe de reuniões de outras equipes e aja como uma ponte de conhecimento, verificando lições aprendidas das demais equipes e observando possíveis melhorias para sua equipe.
- Não podemos introduzir uma nova mudança sempre que alguma pessoa reclamar de alguma coisa, muitas mudanças podem deixar as pessoas relutantes contra vários artefatos de contorno, que pode gerar outros problemas. Deve – se produzir planos de ações para problemas que atrapalham toda equipe, problemas isolados tendem a desaparecem naturalmente.
- Convide membros de outras equipes, principalmente membros da equipe de negócio e equipe de manutenção corretiva que estão intimamente ligados com a equipe de manutenção evolutiva.

Ambos tendem a contribuir e serem mais uma fonte de feedback, bem como levarem melhorias para suas equipes.

- Opiniões de o que foi bom, o que aconteceu de errado e plano de ação devem ser citadas por todos os membros da equipe, em forma de circulo, no qual cada um tem a sua vez. De outra forma alguns simplesmente optam por ficarem calados não expondo os seus sentimentos da sprint.
- Todo o resultado da reunião deve ser divulgado e exposto em lugar visível, para que todos vejam e relembrem das ações de melhorias que foram acordadas pela equipe. Para esta tarefa, seria interessante a função de um escriturário na reunião, responsável por anotar todos os pontos. Se a equipe trabalhar com rodízio de Scrum master, o indicado a essa função seria o próximo.

Um artefato poderoso, porém não aplicado de maneira correta na equipe e não seguindo os princípios básicos do Scrum, é a review da sprint ou revisão da sprint. A revisão deveria ocorrer sempre ao final e composta por todos integrantes da equipe, product owner e convidados da equipe de manutenção corretiva, onde uma apresentação de todas as funcionalidades 100% concluídas seria realizada.

Além de ser um artefato poderoso que agrega vantagens como feedback, garantia de produtos concluídos e validação do trabalho pelo product owner, fornece uma vantagem estratégica para essa equipe, uma vez que as correções não serão realizadas pela própria. Com esse motivo, apresentando as funcionalidades para equipe de manutenção corretiva, estes aprendem o funcionamento de novos itens e acompanham a evolução do sistema. No caso de após a versão sair para o cliente e um erro for retornado, a equipe terá maior subsídios para agilizar a solução e maior conhecimento para evitar possíveis correções que geram outros erros.

As sugestões citadas não deixariam o trabalho da equipe com o seu máximo de produtividade e qualidade, sempre teremos o que melhorar em cada ciclo Scrum, mas obteríamos um salto potencial na obtenção destas melhorias. Há outros fatores menos impactantes que por mais simples que pareçam acabam promovendo ganhos pequenos e que somados melhoram a qualidade do produto e qualidade de trabalho.

Na vida real, nem sempre podemos estar em uma corrida, é preciso descansar, se você está sempre correndo, na verdade você está caminhando. O mesmo acontece no Scrum, em que as sprints são intensas e é preciso um descanso entre uma sprint e outra. Uma forma de promover o descanso é com “lab days”, são dias que os desenvolvedores podem realizar atividades como estudar ferramentas novas, realizarem workshops de assuntos de interesse da equipe, discussões de temas técnicos, enfim qualquer outra abordagem do interesse da equipe que mantenha o conhecimento atualizado. A empresa permite à equipe uma atualização de conhecimento e se torna um benefício atraente para contratação, onde ambos tanto empresa quanto colaboradores ganham (KNIBERG, 2007).

Reuniões diárias devem ser estimulantes aos participantes e não desmotivadoras. Reuniões diárias longas, que abordam outros assuntos e fogem de seus objetivos, desencorajam a equipe a realiza – lá. A equipe se preocupa em manter o planejamento, realizar a execução de suas tarefas de acordo com o planejado e, no entanto criam resistência ou não dão total atenção à reunião, pois estão mais interessadas e preocupadas em retornar a sua atividade. Com este contexto as reuniões têm quer ser breves e objetivas, apenas os itens definidos devem ser discutidos. Outros problemas ou qualquer outro tipo de informação devem ser tratados em particular fora da reunião e no caso em que abrange toda equipe, pode – se marcar uma outra reunião para que a reunião diária não se torne um fator desmotivante.

Sabemos que algumas interferências externas, alterações de requisitos ou erros de planejamento acabam comprometendo a sprint, gerando atraso e horas adicionais para que a sprint volte ao planejado. Atrasos em atividades isoladas, pertencentes a um responsável tendem a preocupar o membro e fazer com que este acelere seu desenvolvimento e algumas vezes podem comprometer a qualidade do release, em que as etapas de testes e validações podem ser extintas ou não realizadas por completo. Alguns participantes da equipe se sentem mais produtivos, sofrem menos interferência trabalhando remotamente. Quando atrasos isolados ocorrerem, onde o participante não dependa de mais pessoas ou informações para terminar a tarefa e se sinta confortável em trabalhar remotamente de sua casa, sob avaliação do Scrum master e do coordenador de equipe, autorizações para trabalho remoto poderiam ser concebidos a este participante para que a tarefa volte ao seu

tempo normal. É valido que se o profissional se sentir bem com a hipótese e for mais produtivo, terá maior confiança para retornar a tarefa a seu tempo planejado, sem comprometer a qualidade do código.

É normal na equipe que sprints durante seu andamento encontrem problemas que comprometam uma parte de seu planejamento e atrase algumas tarefas, porém existem ações para retornar ao planejado como adição de recursos ou até mesmo horas adicionais dos participantes. Através de históricos, poderíamos ter um tempo médio que sempre é despendido a mais para que cada sprint termine como planejado, através de estatísticas obteríamos o tempo mais próximo do real e aplicaríamos como risco na sprint, onde a equipe se comprometeria até o ponto de risco e não seria necessária a adição de novos recursos ou horas adicionais, visto que horas adicionais em excesso se torna um fator negativo tanto para produtividade quanto qualidade.

Com as abordagens sugeridas, a maior parte dos problemas atuais enfrentados pela equipe, citados no capitulo 4, podem não ser resolvidos por completo, mas existe uma grande chance de que melhorem potencialmente. Garantindo a qualidade sem perder em grande parte produtividade resolveria consequentemente problemas encontrados pela equipe de manutenção corretiva. Essa equipe estaria mais focada em cumprir prazos SLAs sem tanta interferência de o que chamamos de apagar incêndios.

Os gráficos estudados e citados no Kanban devem ser aplicados à equipe de manutenção corretiva com objetivo de facilitar a identificar tempos que cada atividade leva para ser cumprida. Gráficos de taxa de defeitos e itens bloqueados estariam guiando a equipe pelos indicadores de qualidade, enquanto que gráficos de fluxo cumulativo e ciclo de tempo ajudariam a estimar e definir prazos de entrega. Sabemos que erros têm diferentes tempos de execução de acordo com sua complexidade, pode – se então separar itens por complexidade, por exemplo, agrupando em níveis como fácil, médio e difícil em que para cada grupo obteríamos através dos gráficos o tempo que uma atividade leva a ser disponibilizada.

Testes hoje em correções são realizados somente nas versões em que o problema foi detectado. Para promover uma qualidade maior, os testes têm que ser realizados tanto na versão corrigida como na versão superior sincronizada, chamada

pela equipe de versão topo. As versões na maioria das vezes estão com estruturas diferentes e por isso precisam que ambas sejam testadas. Para que a correção possa se efetivar por completo, também é necessário que se teste para diferentes bancos de dados trabalhados pela equipe. Desta maneira estaríamos garantindo também a qualidade de correções. A equipe não realiza atualmente todos os testes citados devido a falta de tempo e falta de mão de obra, mas é provado por todos na maior parte dos matérias citados nesta pesquisa, que o quanto antes identificar o problema menor será o custo, com isso, despender mais tempo baixando a produtividade para realizar todos os testes não irá significar menor faturamento para empresa.

6 CONCLUSÕES

Muitas empresas de software buscam agilidade em seu processo de desenvolvimento, baseadas em casos de sucesso com equipes ágeis que produzem software com alta qualidade, menor custo e atendendo as necessidades dos usuários. Algumas equivocadamente seguem padrões sugeridos por modelos ágeis acreditando que apenas esse passo é suficiente para alcançar a tão esperada agilidade, no entanto modelos ágeis são difíceis, a transição completa e os resultados que as empresas esperam são muito mais difícil do que elas imaginavam.

Não podemos simplesmente adotar um modelo ágil e segui – lo fielmente, cada empresa possui sua realidade e o modelo deve se adaptar para obter os melhores resultados possíveis. Com este trabalho, identificamos as melhores práticas nos modelos mais utilizados, independente deles serem ágeis ou tradicionais, e sugerimos aplicação em uma empresa de acordo com sua realidade. Nem toda boa prática de um modelo será uma boa prática para empresa, é preciso estudar, testar e adaptar.

Modelos de software buscam sempre a melhoria contínua, é considerado que por mais satisfatório que seu resultado seja sempre temos o que melhorar e buscamos por isso. Técnicas são bem vindas e devem ser utilizadas, não existe restrição à adição de técnicas ou artefatos ao processo de trabalho, apenas não podemos esquecer do maior objetivo que é tornar - se ágil e não adicionarmos e nem burocratizarmos em excesso.

Modelos ágeis não definem quais artefatos ou técnicas devemos utilizar para tornarmos ágeis, simplesmente apresentam os artefatos e técnicas que propiciam a grande parte das equipes ganhos potencias em agilidade e qualidade. Não apresenta um passo a passo ou uma receita de bolo, por isso são tão difíceis, devemos sempre procurar técnicas ou artefatos que melhor se encaixem em cada equipe, uma vez que cada equipe sempre possuí suas próprias particularidades. Não existe limite de técnicas ou artefatos a serem utilizadas, o importante é um bom senso e um equilíbrio para que o pilar da qualidade não interfira no pilar de produtividade.

Com as técnicas estudadas, apesar de não obtermos dados concretos de ganhos com a pesquisa, as sugestões apresentam forte relação com os problemas apresentados, teoricamente resolvem os problemas e fornecem o ganho. Para que o sucesso da aplicação seja concreto, não basta apenas aplicarmos o uso, deve – se colocar em prática, ir acompanhando e adaptando. Também não podemos impor o uso, deve ser sugerido, plantar a idéia para que a equipe se interesse e se comprometa em realizá – la. O objetivo é mostrar os supostos ganhos e com isso ter subsídios para encorajar e incentivar a equipe. A coleta de dados e obtenção de resultados virá com o tempo e experiência.

O Scrum trabalhado pela equipe produz resultados satisfatórios para gerenciamento e acompanhamento, mas para a realidade da equipe na fase de desenvolvimento notamos que pode ser melhorado utilizando técnicas de outros modelos principalmente da Programação Extrema. Essas técnicas provam teoricamente suprir as necessidades encontradas, provendo a qualidade esperada.

Temos que acreditar em casos de sucessos das técnicas e trabalharmos confiantes, somente assim iremos obter resultados satisfatórios. As técnicas não são simplesmente receitas que bastam serem seguidas e o resultado irá aparecer, exige esforço em conjunto para que se adeque a equipe da melhor maneira possível.

Os modelos não impõem restrições, apenas definem objetivos e mostram as melhores maneiras de os obtermos, podemos sempre estar os incrementando da melhor maneira para que atendam as nossas necessidades.

6.1 TRABALHOS FUTUROS

O principal objetivo é incentivar e aplicar as sugestões desta pesquisa. Após este trabalho será necessário acompanhar pontos negativos e positivos do trabalho para que constantemente melhoremos o processo de desenvolvimento, sempre buscando qualidade.

Paralelamente, pesquisando e identificando novas técnicas e artefatos para problemas já existentes e problemas futuros que poderão ocorrer. O mundo de

software é mutável, teremos que sempre estar nos adaptando e atualizado com as melhores práticas disponíveis.

Foram apresentadas sugestões e dados estatísticos sobre seus ganhos e benefícios, porem para aplicar as técnicas sugeridas, é necessário um estudo e aprofundamento de cada uma.

REFERÊNCIAS

Aliança Ágil em: <http http://www.agilealliance.org>. Acesso em: 05 mar 2012.

BECK, Kent. **Programação extrema explicada** – Porto Alegre : Artmed, 2000.

BOEG, Jesper. **Priming Kanban** – C4Media Inc. 2011. Disponível em <http://www.infoq.com/agile>. Acesso em: 14 jan. 2012.

SCHWABER, Ken; SUTHERLAND, Jeff. **Guia do Scrum,** 2011. Disponivel em <http://www.infoq.com/br/agile>. Acesso em: 10 jan. 2012.

COCKBURN, Alistair. **Agile Software Development** - Massachusetts : Addison - Wesley, 2002.

COHN, Mike. **Desenvolvimento de Software com Scrum** – Bookman, 2011.

HIGHSMITH, Jim. **Agile Software Development Ecosystems**, Massachusetts: Addison - Wesley, 2002.

IBM em: <ftp://ftp.software.ibm.com/software/pdf/br/RUP_DS.pdf>. Acesso em: 25 fev. 2012.

JACOBSON, I.. **A Resounding Yes to Agile Process - But Also More**, Cutter, IT Journal, vol. 15, No. 1, Janeiro 2002, páginas 18-24.

KNIBERG, Henrik. **Scrum e XP Direto das Trincheiras** - C4Media Inc. 2007. Disponível em <http://www.infoq.com/br/agile>. Acesso em: 14 mar. 2012.

KNIBERG, Henrik; SKARIN, Mattias. **Kanban e Scrum – Obtendo o melhor de ambos** – C4Media Inc. 2009. Disponível em <http://www.infoq.com/br/agile>. Acesso em: 14 jan. 2012.

MAIA, José Anízio Pantoja. **Construindo Software com Qualidade e Rapidez Usando ICONIX.** Artigo, 2005. Disponível em: <http://www.guj.com.br/articles/172>. Acesso em: 25 fev. 2012.

PISKE, Otávio Rodolfo. **RUP – Rational Unified Process.** Dissertação, 2003. Disponível em: <http://www.angusyoung.org/arquivos/artigos/trabalho_rup.pdf>. Acesso em: 20 fev. 2012.

PRESSMAN, Roger S.. **Engenharia de Software** – 6 ed. – São Paulo : McGrawHill, 2006.

REZENDE, Denis Alcides. **Engenharia de software e sistemas de informação** – 3 ed. rev. e ampl. – Rio de Janeiro : Brasport, 2006.

TAUB, Adilson Junior. **Rational Unified Process - RUP**. Artigo, 2009. Disponível em: <http://www.baguete.com.br/artigos/731/adilson-taub-junior/04/11/2009/rational-unified-process-rup>. Acesso em: 20 fev. 2012.

VIANNA, Mauro. **Conheça o Rational Unified Process - RUP**. Artigo, 2011. Disponível em: <http://www.linhadecodigo.com.br/artigo/79/conheca-o-rational-unified-process-rup.aspx>. Acesso em: 20 fev. 2012.

WTHREEX em: <http://www.wthreex.com/rup/portugues/index.htm>. Acesso em: 20 Fev 2012. Acesso em: 20 fev. 2012.