UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

ANDREA RAYMUNDO BALLE

Análise de Metodologias Ágeis: Conceitos, Aplicações e Relatos sobre XP e Scrum

Trabalho de Graduação.

Prof. Dr. Leandro Krug Wives
Orientador

Porto Alegre, julho de 2011.

AGRADECIMENTOS

Agradeço a todos os que me apoiaram, em especial meus professores, que despertaram minha curiosidade e vontade de pensar diferente frente a dificuldades. Aos colegas, que se alegraram e sofreram junto comigo e que, por suas diferentes trajetórias profissionais, provaram que a Ciência da Computação é um campo de atuação vasto e repleto de oportunidades, seja qual for seu interesse.

Agradeço meus pais, minha família e amigos, pela paciência, carinho e por me lembrarem, em especial nos últimos seis meses, que existe vida além do Trabalho de Conclusão.

Agradeço a Flávio Steffens, da Woompa, Carlos Santin, da Endeeper, e Alexandra Ibaldo, da AG2, pela oportunidade e confiança de mostrarem seus métodos de trabalho. Sem o estudo prático, eu jamais teria uma visão tão ampla e precisa de como as metodologias ágeis são aplicadas.

Agradeço a todos que tiveram algum papel ao longo da minha trajetória na universidade: os colegas de trabalho e estágios, que tanto me ensinaram, os membros de comunidade de software livre, por mostrarem que não se deve esperar que algo seja feito se você mesmo pode fazê-lo, os desenvolvedores de jogos (de todos os tempos), pela inspiração e diversão que proporcionaram.

A todos os que me ajudaram a ser a pessoa e profissional que sou, o meu mais sincero muito obrigada.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS 6
LISTA DE FIGURAS 7
LISTA DE TABELAS 8
RESUMO 9
ABSTRACT 10
1 INTRODUÇÃO 11
2 METODOLOGIAS ÁGEIS 12
2.1 Scrum 17
2.1.1 Scrum Master 20
2.1.2 Sprint 22
2.1.3 Scrum Team 21
2.1.4 Product Owner 21
2.1.5 Daily Scrum ou Daily Meeting 22
2.1.5.1 Técnica Chickens and Pigs 22
2.1.6 Sprint Planning 22
2.1.7 Incremento de produto potencialmente entregável 23
2.1.8 Product Backlog 23
2.1.9 Sprint Backlog 24
2.1.10 Retrospectiva 24
2.1.11 User Stories 25
2.1.11.1 Card, Conversation, Confirmation 25
2.1.11.2 INVEST 26
2.1.12 Story Points 26
2.1.13 Estimativas 26
2.1.13.1 Planning Poker 26
2.1.14 Acceptance Criteria 27
2.1.15 Velocidade 28
2.1.16 Burndown Chart 28
2.1.17 Release 29
2.1.18 Review Meeting 29
2.1.19 Kanban ou Task Board 29
2.2 Extreme Programming 30
2.2.1 Planning Game 32
2.2.2 Small Releases 33
2.2.3 Metáfora 33
2.2.4 Design Simples 33

2.2.5 Refatoração 33
2.2.6 Testes 34
2.2.7 Pair Programming 34
2.2.8 Propriedade Coletiva 34
2.2.9 Continuous Integration 34
2.2.10 Semana de 40 horas (Susteinable Pace) 35
2.2.11 On-Site Customer 35
2.2.12 Coding Standards 35
2.2.13 Test-Driven Development 35
2.2.14 Time Coeso (Whole Team) 36
2.2.15 Testes de Aceitação (Customer Tests) 36
3 METODOLOGIAS ÁGEIS NO MERCADO 38
3.1 Relatos 38
3.1.1 Telessaúde 38
3.1.2 Software House 40
3.1.3 Sistema de Gestão de Ensino 41
3.1.4 Endeeper 433
3.1.5 Instituto Atlântico 44
3.1.6 SAF-ACS 47
3.1.7 Orbisat 48
3.1.8 Globo.com 49
3.1.9 Agência AG2 51
3.2 Análise de Campo 52
3.2.1 Woompa 52
3.2.2 Endeeper 55
3.2.3 AG2 Publicis Modem 58
3.3 Considerações 61
4 O OLHAR INVERSO: QUANDO AGILE NÃO É SOLUÇÃO 62
5 ANÁLISE E RESULTADOS 65
6 CONCLUSÃO 69
REFERÊNCIAS 71
ANEXO A: GUIA PARA INICIANTES 74
ANEXO B: CONCEITOS IMPORTANTES 77

LISTA DE ABREVIATURAS E SIGLAS

XP	*Extreme Programming*
GP	Gerente de Projeto
BR	Brasil
UFRGS	Universidade Federal do Rio Grande do Sul
BD	Banco de Dados
DDS	Desenvolvimento Distribuído de Software
QA	*Quality Assurance*

LISTA DE FIGURAS

Figura 2.1: Agile Flowchart (WELLS, 2009) 15
Figura 2.2: Esqueleto Scrum – Scrum Alliance (DUNCAN et al., 2005) 18
Figura 2.3: Processo do Scrum (GLOGER, 2007) 19
Figura 2.4: Exemplo de Product Backlog (COHN, 2002) 24
Figura 2.5: Exemplo de Burndown Chart (COHN, 2002) 28
Figura 2.6: Exemplo de Kanban (KNIBERG e SKARIN, 2010) 30
Figura 2.7: Práticas de Extreme Programming (JEFFRIES, 1999) 31
Figura 2.8: Planning/Feedback Loops de XP (WELLS, 2009) 32
Figura 3.1: Sprint Backlog no Task Board (BRUNHERA e ZANATA, 2010) 41
Figura 3.2: Refatoração do código (SAVOINE et al., 2009) 43
Figura 3.3: Gráfico de transções versus Story Points (MARÇAL et al., 2009) 46
Figura 3.4: Diagrama de fluxo de Scrum (PRESSMAN apud BARROS et al., 2009) .. 48
Figura 3.5: Kanban da Orbisat (MESQUITA, 2010) 49
Figura 3.6: Kanban, Burndown Chart e Product Backlog 53
Figura 3.7: artefatos da Retrospectiva de Sprint 54
Figura 3.8: Quadro Branco Colaborativo 54
Figura 3.9: Quadro (Task Board) e Burndown Chart da Endeeper 56
Figura 3.10: Kanban do Time de Arquitetura de Informação 59
Figura 3.11: Ambiente de trabalho da equipe de AI 61

LISTA DE TABELAS

Tabela 2.1: Manifesto para Desenvolvimento Ágil de Software 14
Tabela 3.1: Conversão de Story Points em complexidade de Use Cases 45

RESUMO

Os problemas atualmente enfrentados por empresas de desenvolvimento e departamentos de software estão intimamente ligados a prazos, entregas, comunicação e gerenciamento. Para contornar essas dificuldades e ter mais controle sobre os projetos, muitas empresas estão utilizando metodologias ágeis. No entanto, como o conceito é relativamente novo, existem muitas formas de aplicação para esse conjunto de práticas.

Este trabalho tem por objetivo explanar sobre metologias ágeis, seu surgimento, aplicação, conceitos e principais métodos. Serão discutidas as metodologias *Scrum*, que abrange gerenciamento de projetos de software, e *Extreme Programming*, no âmbito de desenvolvimento. A partir dessa avaliação preliminar, serão descritos e analisados alguns relatos de aplicação de metodologias ágeis, de forma a entender como a teoria está sendo aplicada.

O resultado desse trabalho dá indícios da forma como o mercado usa e encara as metodologias ágeis nos dias atuais. Com isso, pode-se rever conceitos da teoria, dando mais ênfase aos que são largamente utilizados com resultados satisfatórios, filtrando e propondo transformações aos que estão tornando-se obsoletos por falta de aplicação.

Palavras-Chave: metodologias ágeis, Scrum, XP

Analysis of Agile Methods: Concepts, Application and Reports about XP and Scrum

ABSTRACT

The problems faced today by development companies and software departments are closely tied to timelines, deliverables, communication and management. To overcome these difficulties and have more control over projects, many companies are using agile methodologies. However, as the concept is relatively new, there are many ways of applying for this set of practices.

This paper aims to explain about Agile Methodology its appearance, application concepts and main methods. Will be discussed the methodologies Scrum, covering project management software, and Extreme Programming, in the context of development. From this preliminary assessment will be described and analyzed some reports of the application of agile methodologies in order to understand how the theory is being applied.

The result of this work gives evidence of how the market today uses and faces agile methodologies. With this analysis, concepts of the theory can be reviewed, giving more emphasis to those which are widely used with satisfactory results, and proposing changes to the filtering that are becoming obsolete due to lack of application.

Keywords: Agile Methods, Scrum, XP

1 INTRODUÇÃO

Desenvolvimento de software é uma ciência complexa. Diversas variáveis estão em jogo: linguagens, ambientes, requerimentos, tempo, sendo que a principal delas é a relação estabelecida com o cliente. Afinal, é a necessidade do mesmo que deve ser suprida com o produto final.

Entretanto, nas abordagens tradicionais de desenvolvimento, como o *método em cascata, método em espiral* ou a *metodologia orientada a objetos*, o envolvimento do cliente em todo o processo de construção do software costuma ser mínimo. Geralmente, os analistas de negócios fazem toda a ponte entre o cliente e a equipe de desenvolvimento, e isso costuma acontecer nas fases iniciais, onde os requerimentos são desenvolvidos, e, depois, somente nas fases finais, de homologação do programa.

Com uma sociedade cada vez mais voltada para resultados rápidos, é um tanto complicado gerenciar um processo de desenvolvimento de software que se estende por meses sem dar retorno para o maior interessado, sendo que, no final do prazo, pode-se ter um produto que não atende completamente às expectativas.

Segundo Kent Beck (1999), "o desenvolvimento de software tem falhas na entrega e nos valores entregues. Essas falhas têm impactos econômicos e humanos enormes. É necessário achar uma maneira de desenvolver software com qualidade e entregas frequentes". Para isso, uma das soluções apresentadas é o uso de metodologias ágeis, um conjunto de métodos e práticas que oferecem respostas rápidas a mudanças, adaptando-se a realidade onde requisitos são mutáveis constantemente e necessita-se de uma qualidade crescente (BECK, 1999).

O objetivo do presente trabalho é estudar as metodologias ágeis em seus conceitos gerais e em especial os dois métodos mais largamente utilizados no presente momento: *Scrum* e *Extreme Programing*, ou *XP*. O primeiro é uma metodologia de gerenciamento, enquanto que o segundo é uma metodologia de desenvolvimento. Será feita uma identificação da forma como as metodologias ágeis estão sendo aplicadas em empresas, envolvendo um levantamento de quais pontos estão sendo utilizados tal como na teoria e quais são descartados, verificando assim a importância de especialização e o impacto dessa na rotina de profissionais e equipes. Para isso, serão realizadas análises de relatos considerando tanto artigos publicados, depoimentos em *blogs* e pesquisas com empresas e profissionais sobre como eles utilizam as metodologias ágeis – ou por que não as utilizam.

Com o estudo desenvolvido e a observação de que muitas das práticas das metodologias não são aplicadas da forma teórica, criou-se um material para auxiliar profissionais que estão ingressando nos estudos das Metodologias Ágeis. Com a análise dos relatos, pôde-se extrair as práticas que o mercado mais utiliza e nas quais deve ser depositado maior esforço em um primeiro momento de estudo da metodologia escolhida. Portanto, esse Guia para Iniciantes tem por objetivo aumentar a velocidade da curva de aprendizado desses profissionais, sem que os conceitos teóricos das metodologias estudadas se percam. Sendo um subproduto da pesquisa, o Guia para Iniciantes encontra-se nos Anexos do presente trabalho.

O documento está estruturado da seguinte forma. Na próxima seção são apresentados os conceitos de metodologias ágeis e uma explanação mais aprofundada em *Scrum* e *XP*. Na seção 3 serão descritos os relatos de uso de metodologias ágeis, análise de *papers* e observação de campo. Na seção 4 serão abordados casos e situações em que as metodologias ágeis não são indicadas ou não são utilizadas, evidenciando-se seus motivos e soluções encontradas. Na seção 5 serão avaliados os resultados obtidos nas seções 3 e 4 e se dará a análise de como está a aplicação das metodologias ágeis no cenário atual, abordando-se o foco que as empresas dão, para quais práticas, quais os denominadores comuns e as maiores dificuldades enfrentadas. A seção 6 conclui o trabalho, com as considerações finais e trabalhos futuros.

2 METODOLOGIAS ÁGEIS

As metodologias ágeis são parte da Engenharia de Software, a área de conhecimento voltada para a especificação, desenvolvimento e manutanção de sistemas de software. De acordo com Pressman (2007), a Engenharia de Software abrange três componentes básicos: *métodos*, que proporcionam os detalhes de como construir um software, tratando do planejamento, estimativas, análises de requisitos e arquitetura, entre outros; *ferramentas*, que sustentam cada um dos métodos; e *procedimentos*, que definem a sequência em que os métodos são aplicados, e fazem o elo entre os métodos e as ferramentas.

Existem diversas definições quando tratamos de metodologias ágeis. Uma das mais aceitas é que *metodologias ágeis* são um conjunto de práticas que seguem os princípios do *Manifesto Ágil* (BECK et al., 2001).

Com a necessidade de tornar o desenvolvimento de software mais leve, flexível a mudanças e sem aumento exponencial de custos, em fevereiro de 2001, dezessete profissionais na área de software, sendo desenvolvedores, gerentes, entusiastas, se reuniram no Snowbird Ski Resort, em Utah, criando a *Agile Software Development Alliance*, mais conhecida como *Agile Alliance*. Dessa reunião, surgiu o *Manifesto Ágil para Desenvolvimento de Software*. Estavam presentes representantes de *Extreme Programming*, *Scrum*, DSDM (*Dynamic Systems Development Method)*, *Adaptive Software Development*, *Crystal*, *Feature-Driven Development*, *Pragmatic Programming*, e outros simpáticos à necessidade de uma alternativa ao desenvolvimento "orientado a documentação", tradicional e pesado (BECK et al., 2001).

Um importante ponto a ser notado é que as metodologias vieram antes do Manifesto. Algumas, inclusive, datam dos anos 80, como *Extreme Programming*. A reunião foi feita para proporcionar a troca de ideias, que acabaram chegando em alguns pontos comuns para todos esses métodos.

Estes profissionais de diversas áreas de formação, com pontos de vista diferentes sobre modelos e métodos de desenvolvimento de software, criaram um manifesto para encorajar melhores meios de desenvolver software (AMBLER, 2002). As principais ideias foram compiladas para expressar os princípios que guiam as metodologias ágeis.

Tabela 2.1: Manifesto para Desenvolvimento Ágil de Software

Valorizamos	**Mais que**
Indivíduos e interações	Processos e ferramentas
Software funcional	Documentação extensa
Relacionamento com o cliente	Negociação de contrato
Responder a mudanças	Seguir um planejamento

Fonte: BECK et. al., 2001

Além disso, o *Manifesto Ágil* é apoiado em doze princípios (BECK et al., 2001):

1. Nossa maior prioridade é satisfazer o cliente, através de entregas rápidas e contínuas gerando valor ao software.
2. Recebendo bem as mudanças dos requisitos, mesmo em estágios tardios do desenvolvimento. Processos ágeis devem admitir mudanças que trazem vantagens competitivas para o cliente.
3. Trabalhando para entregar software, em intervalos de 2 semanas até 2 meses, com preferência para que tenha uma curta escala de tempo.
4. Empresários e desenvolvedores devem trabalhar juntos diariamente durante todo o projeto.
5. Construa projetos com indivíduos motivados, dê-lhes o ambiente e o suporte que precisam, e confie neles para ter o trabalho realizado.
6. O método mais eficiente e efetivo de transmitir informação para a equipe de desenvolvimento está na conversa face a face.
7. Software funcionando é a principal medida para o progresso.
8. Processos ágeis promovem o desenvolvimento sustentável. Os patrocinadores, os desenvolvedores, e os usuários devem ser capazes de manter um ritmo constante indefinidamente.
9. Atenção contínua para uma excelência técnica e um bom design aumentam a agilidade.
10. Simplicidade – a arte de maximizar o valor do trabalho não feito – é essencial.
11. As melhores arquiteturas, requisitos, e design surgem a partir de equipes auto-organizadas.
12. Em intervalos regulares, as equipes devem refletir sobre como se tornaram mais efetivas. Em seguida, devem se aprimorar e ajustar de acordo com seu comportamento. Para isso, cada metodologia conta seus ciclos específicos e pode se apoiar em artefatos que auxiliam na avaliação do trabalho realizado e programação do que deverá ser realizado, seguido ou melhorado para os ciclos seguintes.

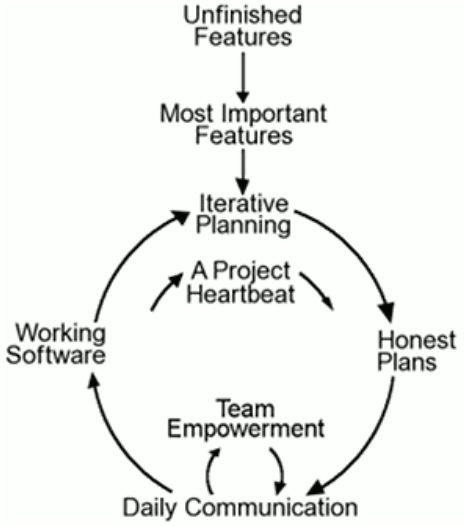

Figura 2.1: Agile Flowchart (WELLS, 2009)

Assim como existem diversas metodologias ágeis, também existem conjuntos de práticas ágeis que são aceitas por quaisquer das metodologias, podendo ser adotadas pelo times ágeis ou não. Essas práticas muitas vezes são uma reunião de conceitos de diversas metodologias, organizados de forma neutra para que possam viver concomitantemente com qualquer uma, ou nenhuma. *Agile Modeling* (AMBLER, 2002) é um processo baseado em práticas que descreve como ser um modelador efetivo. Essas práticas são muito baseadas em metodologias ágeis diversas, mas aplicadas à modelagem e vistas de forma mais ampla.

Agile Modeling ajuda a encontrar o meio-termo entre os extremos em que as formas de modelar podem cair, seja não existindo nenhum modelo, o que acaba significando retrabalho, como na modelagem excessiva, quando documentos e modelos demais são criados, o que atrasa o desenvolvimento. A proposta do *Agile Modeling* é modelar o suficiente para explorar e documentar o sistema de forma eficiente, mas não a ponto de diminuir a velocidade do projeto (AMBLER, 2002).

Sendo uma técnica e não uma metodologia, *Agile Modeling* pode ser aplicado por times que utilizam metodologias ágeis. Seus princípios e técnicas são partilhados por várias metodologias ágeis, como XP, Scrum, entre outras. No entanto, a técnica pode ser também aplicada por projetos que não utilizam metódos ágeis e desejam melhorar e simplificar seus modelos.

As metodologias ágeis entram em um contexto de mercado que Chris Anderson descreve como Freemium (ANDERSON, 2009). No caso, as metodologias foram desenvolvidas por empresas ou pessoas físicas, que não cobram nada pela sua utilização, em uma primeira análise, o que poderia caracterizar um Mercado Não-Monetário. Não há como negar que se pode simplesmente procurar informações na Web, onde milhares de sites e materiais de apoio são mantidos, inclusive por alguns dos participantes do Manifesto Ágil. Pode-se, inclusive, simplesmente seguir as diretrizes do próprio Manifesto, aplicando os princípios ágeis em qualquer setor.

Essa gratuidade das metodologias, que também não impõe uma ferramenta específica para a caracterização dos seus conceitos, vem atraindo diversas empresas, da mesma forma que uma conta gratuita (mas limitada) de um serviço na Web atrai muitos usuários. No entanto, há toda uma economia girando em torno das metodologias ágeis:

existem certificações oficiais, livros das mais diversas metodologias e práticas são lançados, cursos, workshops e palestras são muito requisitadas, em especial dos autores-chave das metodologias em questão e, como pode-se verificar com a análise de relatos de aplicação de metodologias ágeis (ver capítulo 3 deste trabalho), é comum que sejam contratados consultores para o auxílio da implantação da metodologia escolhida.

Dessa forma, a gratuidade das metodologias atrai mais consumidores, mas estes, sem uma orientação mais profunda e correta, acabarão muitas vezes aplicando a metodologia com uma interpretação errada. Assim, alguns desses consumidores procurarão orientação, seja em cursos, livros, consultorias, muitas vezes encabeçadas por autores consagrados, os mesmos que mantém materiais gratuitos que são a porta de entrada para o mundo de metodologias ágeis, o que caracteriza um mercado Freemium.

O Manifesto Ágil diz que software funcional é mais valorizado que uma documentação extensa. Isso acaba por gerar confusão, sendo que um dos erros básicos dos que estão começando a trabalhar com metodologias ágeis é acreditar que não devem manter documentação nenhuma. Metodologias ágeis podem comportar documentação, porém, sob de forma que não deixe de lado as características que definem o trabalho ágil.

Segundo Scott Ambler (2002), um documento é ágil se seguir uma série de critérios. Primeiramente, os documentos ágeis maximizam o investimento dos *stakeholders*, ou seja, o benefício que deles provêm é maior que o investimento em sua criação e manutenção. Por isso, descrevem somente informações que não mudam com facilidade, pois quanto maior a chance de uma informação mudar, menor o valor de um documento escrito sobre a mesma. Da mesma forma, um documento ágil deve capturar informações críticas e que não são facilmente deduzíveis.

Um documento ágil contém somente as informações para atingir o seu objetivo, ou seja, é o mais simples possível. E, mais do que isso, precisa ter um objetivo simples e facilmente identificável. Precisa ter um público específico e facilitar o trabalho do mesmo, sendo suficientemente precisa e detalhada (e não mais que isso). E, por fim, documentos ágeis são indexados de forma eficiente e precisa, pensando em seu público-alvo.

O conjunto de práticas que é o coração das metodologias ágeis pode e está sendo aplicado em empresas e projetos que não se dizem ágeis. Um bom exemplo é o método de trabalho da empresa *37signals*, criadora do *Ruby on Rails*, descrito pelos seus fundadores Jason Fried e David H. Hansson no livro *Rework* (2010). Apesar da *37signals* não adotar nenhum método em específico, alguns de seus capítulos descrevem práticas adotadas que estão em concordância tanto com o Manifesto Ágil como com práticas de *Scrum* e *Extreme Programming*.

Quando se diz que planejamentos devem ser feitos a curto prazo, que deve-se entregar o produto aos poucos e lançá-lo com rapidez e que *workaholics* não fazem bem para o negócio, podemos ver conceitos como *backlog*, incremento de produtos potencialmente entregável e de semana de 40 horas (que serão explanados adiante). Também existem algumas práticas da empresa que não se adaptam em nenhuma metodologia, pois foram desenvolvidas como forma de trabalho para o dia-a-dia da empresa. Em uma interpretação livre, poder-se-ia encaixar o método de trabalho da

37signals em uma combinação levemente modificada de Scrum com *Extreme Programming*, o que, como se perceber-a no capítulo 3, é a forma de trabalho mais comum das empresas que começam a adotar metodologias ágeis.

Esse exemplo prova que os métodos ágeis e seus conceitos podem ser adotados com flexibilidade, adaptando-se às necessidades da empresa e projeto, sem mesmo a necessidade de uma nomenclatura em específico. Ver-se-á nos relatos e em suas análises que existem projetos bem-sucedidos indo contra algumas práticas da teoria.

O foco deste trabalho é estudar a teoria nos mais diversos aspectos, e ver de que forma ela está sendo aplicada, seja por empresas e profissionais utilizando a forma totalmente gratuita, seja orientados por consultorias, workshops ou literatura.

O universo *agile* contém diversos conceitos que serão explorados neste trabalho. Muitos deles não são intuitivos, mas devem ser compreendidos com precisão, pois são de suma importância para o total entendimento e correta implementação das metodologias aqui apresentadas.

Tendo como objetivo principal desta monografia a interpretação, análise e classificação de pontos comuns na aplicação de metodologias ágeis, é importante saber quais são os conceitos mais relevantes e aos quais deve nos prender para um entendimento pleno, maior abrangênica e acuidade no que tange ao cenário atual da aplicação dos métodos e seus conceitos. Como o domínio de metodologias ágeis é muito amplo, são tratados os termos de mais relevância e abrangência, ou com foco específico em Scrum e *Extreme Programming*, por serem as metodologias mais amplamente utilizadas, muitas vezes conjuntamente, como visto mais adiante no capítulo 3, com a descrição dos relatos de aplicação.

2.1 Scrum

Scrum não é um acrônimo. O nome vem de mecanismos do *rugby* para colocar uma bola de volta no jogo: um círculo denso de pessoas e normalmente é separado pelos membros do time de *rugby* que brigam pela posse da bola. Esse termo foi utilizado pela primeira vez por Nonaka e Takeuchi (TAKEUSHI e NONAKA, 1986). No capítulo "*Moving the Scrum Downfield"* são descritas seis características que, usadas conjuntamente, "produzem um conjunto dinâmico" e são utilizadas pelas empresas de ponta no gerenciamento de seus projetos.

Segundo a *Scrum Alliance* (DUNCAN et al, 2005), Scrum é um framework ágil para a realização de projetos complexos. Scrum originalmente foi formalizado para projetos de desenvolvimento de software, mas funciona bem para qualquer escopo, complexo e inovador de trabalho. Um exemplo disso é o Centro de Estudos de Comportamento Reali (http://www.reali.com.br/), que apesar de não trabalhar com desenvolvimento de software em nenhum âmbito, utiliza os conceitos de Scrum para gerência de seus projetos. Um dos motivos para a abrangência de campos é a simplicidade do framework Scrum.

O trabalho de Scrum considera como bibliografia básica, que define as bases da metodologia, os livros *Agile Software Development with Scrum*, de Ken Schwaber e Mike Beedle e *Agile Software Management with Scrum*, de Ken Schwaber. Existem outros livros e materiais que adicionam características que, aos poucos, foram sendo

comumente usados com as bases da metodologia e que serão também aqui considerados, explicitando que essas técnicas são adendos.

Os termos utilizados na explicação a seguir são mais explanados adiante neste mesmo capítulo. Como o cerne da metodologia está intimamente ligado com esses termos e conceitos, eles serão explanados apropriadamente.

O *Scrum* destaca-se dos demais métodos ágeis pela maior ênfase dada ao gerenciamento do projeto. Reúne atividades de monitoramento e *feedback*, em geral, reuniões rápidas e diárias com toda a equipe, visando a identificação e correção de quaisquer deficiências e/ou impedimentos no processo de desenvolvimento (SCHWABER, 2004). O método baseia-se ainda em princípios como: equipes pequenas de, no máximo, sete pessoas; requisitos que são pouco estáveis ou desconhecidos; e iterações curtas. Divide o desenvolvimento em intervalos de tempos de no máximo, trinta dias, também chamados de *Sprints*.

O *Scrum* implementa um esqueleto iterativo e incremental através de três papéis principais (SCHWABER, 2004): *Product Owner*: representa os interesses de todos no projeto; Time: desenvolve as funcionalidades do produto; *ScrumMaster*: garante que todos sigam as regras e práticas do *Scrum*, além de ser o responsável por remover osimpedimentos do projeto.

Um projeto no *Scrum* se inicia com uma visão do produto que será desenvolvido (SCHWABER, 2004). A visão contém a lista das características do produto estabelecidas pelo cliente, além de algumas premissas e restrições. Em seguida, o *Product Backlog* (seção 2.1.8) é criado contendo a lista de todos os requisitos conhecidos. O *Product Backlog* é então priorizado e dividido em *releases*.

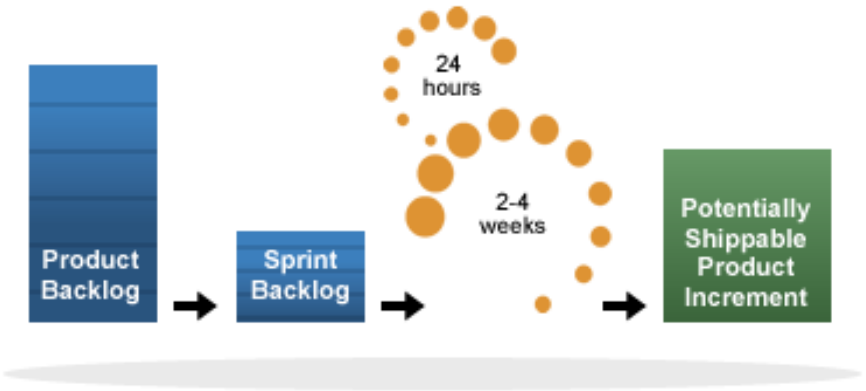

Figura 2.2: Esqueleto *Scrum* – Scrum Alliance (DUNCAN et al., 2005)

Projetos Scrum progridem em uma série de *Sprints*, que são iterações não maiores do que um mês. No início de um *Sprint*, os membros da equipe se comprometem a entregar um número de características que foram listadas no *Product Backlog*. No final do *Sprint*, esses recursos estão feitos - estão codificados, analisados e integrados no produto em desenvolvimento ou sistema. No final do *Sprint* há uma revisão durante a qual a equipe demonstra a nova funcionalidade para o *Product Owner* e outras partes interessadas que fornecem *feedback* que possa influenciar o próximo Sprint (COHN, 2002).

Schwaber (2004) explica que cada *Sprint* inicia-se com uma reunião de planejamento (*Sprint Planning Meeting*), na qual o *Product Owner* e o Time decidem em conjunto o que deverá ser implementado (*Selected Product Backlog*). A reunião é dividida em duas partes. Na primeira parte (*Sprint Planning* 1) o *Product Owner* apresenta os requisitos de maior valor e prioriza aqueles que devem ser implementados. O Time então define, colaborativamente, o que poderá entrar no desenvolvimento da próxima *Sprint*, considerando sua capacidade de produção. Na segunda parte (*Sprint Planning* 2), o time planeja seu trabalho, definindo o *Sprint Backlog*, que são as tarefas necessárias para implementar as funcionalidades selecionadas no *Product Backlog*. Nas primeiras *Sprints*, é realizada a maioria dos trabalhos de arquitetura e de infraestrutura. A lista de tarefas pode ser modificada pelo Time ao longo da *Sprint*, e as tarefas podem variar entre quatro e dezesseis horas para a sua conclusão.

No final da *Sprint* é realizada a reunião de revisão (*Sprint Review Meeting*) para que o Time apresente o resultado alcançado na iteração ao *Product Owner*. Nesse momento, as funcionalidades são inspecionadas e adaptações do projeto podem ser realizadas. Em seguida, o *ScrumMaster* conduz a reunião de retrospectiva (*Sprint Retrospective Meeting*), com o objetivo de melhorar o processo/time e/ou produto para a próxima *Sprint*.

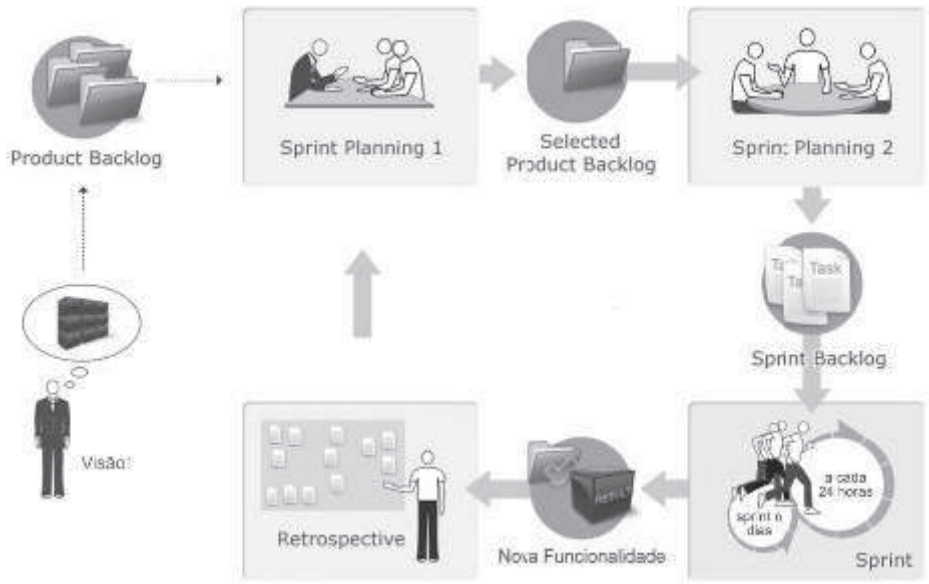

Figura 2.3: Processo do Scrum (GLOGER, 2007)

Scrum é permite manter o foco na entrega do maior valor de negócio, no menor tempo possível. Isto permite a rápida e contínua inspeção do software em produção. As necessidades do negócio é que determinam as prioridades do desenvolvimento de um sistema. Todos podem ver o software real em produção, decidindo se o mesmo deve ser liberado ou continuar a ser aprimorado por mais um *Sprint*.

A equipe Scrum é autoorganizável, e não há nenhum líder geral da equipe que vai decidir qual pessoa vai fazer qual tarefa ou como um problema será resolvido. Essas são questões que são decididas pela equipe como um todo. A equipe é multifuncional e apoiada por dois indivíduos específicos: um *Scrum Master* e um *Product Owner*. O

ScrumMaster pode ser visto como um treinador para a equipe, ajudando os membros da equipe a utilizar o framework Scrum. O *Product Owner* é o proprietário do produto e representa a empresa, clientes ou usuários, orientando a equipe de desenvolvimento na construção do produto correto.

O ambiente de trabalho de Scrum é extremamente importante. Ele deve ser aberto, para facilitar o diálogo da equipe e a auto-organização. O Time deve ter à mão sempre as melhores ferramentas possíveis para a realização do trabalho e há mais sucesso se todo o Time ocupar o mesmo espaço físico, ou seja, se trabalhar na mesma sala. No entanto, isso não é fator determinante e existem diversos relatos (um deles explanado no capítulo 3) que mostram sucesso com equipes distribuídas.

As práticas de Scrum evoluíram ao longo da sua aplicação em milhares de projetos. É recomendado que essas práticas sejam aplicadas estritamente, até que seja entendida a forma como Scrum funciona com a experiência. Quando Scrum estiver funcionando bem, podem ser feitos ajustes (SCHWABER and BEEDLE, 2001). Pode-se, no entanto, ver nos relatos descritos adiante que diversas vezes as equipes não aplicam essa recomendação, pelos mais diversos motivos, tendo adaptado a forma como Scrum é aplicado sem tê-lo experimentado em sua totalidade anteriormente.

O estudo desse trabalho foca com muita ênfase nessas práticas e conceitos, pois o básico da teoria é de conhecimento mais amplo, como visto no capítulo 3, de análise de relatos. As diferenças maiores se dão na aplicação de interpretação dos conceitos que, por serem numerosos e de diversas fontes, dão abertura para interpretações diversas. O próprio livro-base da metodologia, *Agile Software Development with Scrum* (SCHWABER and BEEDLE, 2002), organiza a metodologia como uma série de práticas que, juntas, formam o que se chama de Scrum.

Dessa forma, serão explanados diversos conceitos, ferramentas e artefatos da teoria de Scrum. Alguns deles estão na bibliografia básica da metodologia, outros foram descritos porteriormente por outros autores e são amplamente aplicados, sendo que, muitas vezes, são artefatos descritos pelo autor explicitamente para o uso conjunto. Todos são de extrema importância tanto para o entendimento pleno da metodologia como deste trabalho.

2.1.1 Scrum Master

O *Scrum Master* ocupa a posição normalmente ocupada pelo Gerente de Projeto. Enquanto um gerente de projeto tradicional é responsável por definir e gerenciar o trabalho, o *Scrum Master* é a pessoa responsável pelo processo Scrum, sua correta implementação e a maximização de seus benefícios.

O *Scrum Master* auxilia o time e a organização a adotarem o *Scrum*, educa o time, treinando-o e levando-o a ser mais produtivo e a desenvolver produtos de maior qualidade. Também ajuda o time a fazer o seu melhor em um ambiente organizacional que pode ainda não ser otimizado para o desenvolvimento de produtos complexos. Quando o *Scrum Master* ajuda a realizar essas mudanças, isso é chamado de "remoção de impedimentos". No entanto, o Scrum Master não gerencia o time; este é auto-organizável (SCHWABER e SUTHERLAND, 2009).

O *Scrum Master* deve conhecer os processos, práticas, artefatos e terminologia *Scrum* e saber como aplicá-los corretamente.

2.1.2 Sprint

Em esportes, um *Sprint* é uma corrida curta de alta velocidade. Essa definição encaixa-se muito bem ao *Sprint* da metodologia *Scrum*: um período de 30 dias durante o qual o time trabalha para transformar o *Product Backlog* (ver seção 2.1.11 para a definição) selecionado em um incremento de produto potencialmente entregável (SCHWABER, 2004). Esse período de tempo pode variar em alguns times, mas é sempre proporcional e *time-boxed*, ou seja, não prorrogável.

O *Sprint*, portanto, é um espaço curto de tempo em que o objetivo definido deve ser alcançado de forma rápida e precisa.

2.1.3 Scrum Team

Times de desenvolvedores transformam o *Backlog* do Produto em incrementos de funcionalidades potencialmente entregáveis em cada *Sprint*. Times são interdisciplinares: membros do time devem possuir todo o conhecimento necessário para criar um incremento no trabalho (SCHWABER, 2004). Membros do time frequentemente possuem conhecimentos especializados, como programação, controle de qualidade, análise de negócios, arquitetura, projeto de interface de usuário ou projeto de banco de dados. No entanto, o conhecimento que os membros do time devem compartilhar - isso é, a habilidade de trabalhar um requisito e transformá-lo em um produto utilizável - tendem a ser mais importantes do que aqueles que eles não dividem. Não há títulos em times, e não há exceções a essa regra. Os times também não contém subtimes dedicados a áreas particulares como testes ou análise de negócios (SCHWABER e SUTHERLAND, 2009).

Times também são auto-organizáveis. Ninguém - nem mesmo o *Scrum Master* - diz ao time como transformar o *Backlog* do Produto em incrementos de funcionalidades entregáveis. O time descobre por si só. Cada membro do time aplica sua especialidade a todos os problemas. A sinergia que resulta disso melhora a eficiência e eficácia geral do time como um todo (SCHWABER e SUTHERLAND, 2009).

A composição do time pode mudar ao final da *Sprint*. Toda vez que o time muda, a produtividade ganha através da auto-organização é reduzida. Deve-se tomar cuidado ao mudar a composição do time, pois a produtividade aumenta quando os membros do time já estão habituados tanto com a metodologia que está sendo utilizada como com o projeto, de forma que a curva para o aprendizado acaba por diminuir a velocidade (conceito adiante) que o time é capaz de atingir, como pode ser comprovado por alguns dos relatos que serão tratados nos próximos capitulos, com atenção especial ao relato da Globo.com (seção 4.1.8).

2.1.4 Product Owner

O *Product Owner* (geralmente alguém com um papel relacionado com o marketing ou um usuário-chave do desenvolvimento interno) prioriza o *Product Backlog*. Ele é o responsável por gerenciar o *Product Backlog* de forma a maximizar o valor do projeto.

O *Scrum Master* trabalha com os clientes e a gerência para identificar e designar um *Product Owner*. Espera-se dos *Product Owners* que eles saibam como conseguir

otimizar valor através do *Scrum*. Se eles não souberem, considera-se o Scrum Master responsável (SCHWABER e SUTHERLAND, 2009).

O *Scrum Team* olha para o *Product Backlog* priorizado, seleciona os itens de alta prioridade e se compromete a completá-los durante um *Sprint*. Esses itens se tornarão o *Sprint Backlog*.

Em troca de seu compromisso de completar as tarefas selecionadas (que, por definição, são o mais importante para o *Product Owner*), o *Product Owner* compromete-se a não lançar novas exigências à equipe durante o *Sprint*. Requisitos podem mudar (e a mudança é encorajada), mas apenas fora do *Sprint* (COHN, 2002).

2.1.5 Daily Scrum ou Daily Meeting

A *Daily Scrum* é uma reunião de curta duração realizada diarimente pelo *time*, na qual cada um dos membros sincroniza seu trabalho e progresso e reporta quaisquer impedimentos a fim de que o *Scrum Master* os remova.

A *Daily Scrum* é sempre feita no mesmo horário e no mesmo local durante as *Sprints* e tem duração máxima de 15 minutos (SCHWABER e SUTHERLAND, 2009). Durante a reunião, cada membro explica o que ele realizou desde a última reunião diária, o que ele vai fazer antes da próxima reunião diária quais obstáculos estão em seu caminho.

O *Scrum Master* garante que o time realize a *Daily Scrum* e o time é responsável por conduzi-la. O *Scrum Master* deve a manter a *Daily Scrum* com curta duração, reforçando as regras e garantido que as pessoas falem brevemente.

O sentido da *Daily Scrum* não é ser uma reunião de status, é uma inspeção do progresso na direção da meta do *Sprint*. Geralmente acontecem reuniões subsequentes para realizar adaptações ao trabalho que está por vir na *Sprint*. A intenção é otimizar a probabilidade de que o time alcance sua meta. Essa é uma reunião fundamental de inspeção e adaptação no processo empírico do Scrum.

2.1.5.1 Técnica Chickens and Pigs

Existe uma piada em que uma galinha e um porco estão conversando, e a galinha diz: "Vamos abrir um restaurante." O porco responde: "Boa ideia, mas como devemos chamá-lo?". "Que tal 'presunto e ovos'", diz a galinha. "Não, obrigado", disse o porco, "eu estaria comprometido, você só estaria envolvida".

A brincadeira serve para apontar a diferença entre aqueles que estão comprometidos com um projeto, ou seja, alguém ocupando um dos três papéis do *Scrum* (Time, Product Owner ou Scrum Master) e aqueles que só estão envolvidos, não tendo responsabilidades formais no processo *Scrum*. *Scrum* confere um estatuto especial para aqueles que estão comprometidos e muitas equipes impõem uma regra em que apenas aqueles que estão comprometidos (*pigs*) têm permissão para falar durante a *Daily Scrum*, enquanto os que estão envolvidos (*chickens)* apenas a assistem (COHN, 2002).

2.1.6 Sprint Planning

O *Sprint Planning Meeting* é uma reunião na qual estão presentes o *Product Owner*, o *Scrum Master* e todo o time, bem como qualquer pessoa interessada que esteja representando a gerência ou o cliente. O Sprint Planning é uma reunião de um dia, com

duração de 8 horas, que inicia um *Sprint*. A reunião é dividida em dois segmentos de 4 horas. Esses segmentos têm duração fixa, não podendo se extender (SCHWABER, 2004).

Durante o *Sprint Planning*, o *Product Owner* descreve as funcionalidades de maior prioridade para a equipe. A equipe faz perguntas durante a reunião de modo que seja capaz de quebrar as funcionalidades em tarefas técnicas, após a reunião. Essas tarefas irão dar origem ao *Sprint Backlog*. O time e o *Product Owner* definem um objetivo para o *Sprint*, que é uma breve descrição daquilo que se tentará alcançar no *Sprint.*

Depois do Sprint Planning Meeting, o time se encontra separadamente para conversar sobre o que eles escutaram e decidir quanto eles podem se comprometer a fazer no Sprint que será iniciado. Em alguns casos, haverá negociação com o *Product Owner*, mas será sempre responsabilidade da equipe determinar o quanto ela será capaz de se comprometer a fazer (SCHWABER, 2004).

Visto que a *Sprint Planning* foi feita com a duração de 8 horas para comportar o planejamento de *Sprints* de 4 semanas, existem times que personalizam essa duração quando seus *Sprints* são menores. Geralmente, a duração da *Sprint Planning* cai pela metade se o *Sprint* é de 2 semanas, como pode ser verificado em alguns dos relatos descritos no capítulo 3.

2.1.7 Incremento de produto potencialmente entregável

Incremento de produto potencialmente entregável é um conjunto de realizações que poderia ser liberado para cliente. O *Product Owner* toma a decisão sobre quando realmente a liberar o lançamento de qualquer funcionalidade ou produto.

2.1.8 Product Backlog

Backlog, em tradução literal, significa reserva, acúmulo. Isso já dá uma idéia conceito de *Product Backlog*.

Product Backlog é uma lista priorizada de requerimentos do projeto com estimativas (tempo, pontos, etc.) para torná-los funcionalidades completas. Estimativas em dias são mais precisas quanto maior for a prioridade do item no *Product Backlog* (SCHWABER, 2004). A lista evolui, mudando conforme as condições do negócio ou a tecnologia muda.

A *Product Backlog* é composta de itens que são os requerimentos funcionais, os não funcionais e os defeitos. Eles são priorizados em ordem de importância para o negócio e em ordem de dependências, e, então, estimados. A precisão da estimativa depende da prioridade e granularidade do item, com os itens de maior prioridade que podem ser selecionados no próximo *Sprint* sendo muito granulares e precisos.

	Item #	Description	Est	By
Very High				
	1	**Finish database versioning**	16	KH
	2	**Get rid of unneeded shared Java in database**	8	KH
	-	**Add licensing**	-	-
	3	Concurrent user licensing	16	TG
	4	Demo / Eval licensing	16	TG
		Analysis Manager		
	5	File formats we support are out of date	160	TG
	6	Round-trip Analyses	250	MC
High				
	-	**Enforce unique names**	-	-
	7	In main application	24	KH
	8	In import	24	AM
	-	**Admin Program**	-	-
	9	Delete users	4	JM
	-	**Analysis Manager**	-	-
	10	When items are removed from an analysis, they should show up again in the pick list in lower 1/2 of the analysis tab	8	TG
	-	**Query**	-	-
	11	Support for wildcards when searching	16	T&A
	12	Sorting of number attributes to handle negative numbers	16	T&A
	13	Horizontal scrolling	12	T&A
	-	**Population Genetics**	-	-
	14	Frequency Manager	400	T&M
	15	Query Tool	400	T&M
	16	Additional Editors (which ones)	240	T&M
	17	Study Variable Manager	240	T&M
	18	Haplotypes	320	T&M
	19	**Add icons for v1.1 or 2.0**	-	-
	-	**Pedigree Manager**	-	-
	20	Validate Derived kindred	4	KH
Medium				
	-	**Explorer**	-	-
	21	Launch tab synchronization (only show queries/analyses for logged in users)	8	T&A
	22	Delete settings (?)	4	T&A

Figura 2.4: Exemplo de Product Backlog (COHN, 2002)

2.1.9 Sprint Backlog

Uma lista de tarefas que define o trabalho do time por um *Sprint*. A *Sprint Backlog* é composta por itens, que são tarefas que o time ou um membro do time definiu como necessárias para transformar um item em uma funcionalidade do sistema.

Cada tarefa possui uma identificação daqueles responsáveis por realizá-la e o trabalho restante estimado na tarefa em qualquer dia durante o *Sprint*.

2.1.10 Retrospectiva

Retrospectiva é uma reunião, com duração fixa de três horas, onde *Scrum Master* encoraja o time a revisar, dentro do modelo de trabalho e das práticas do processo do *Scrum*, seu processo de desenvolvimento, de forma a torná-lo mais eficaz na próxima *Sprint*.

A finalidade da Retrospectiva é inspecionar como correu a última *Sprint*, identificar e priorizar os principais, itens que correram bem e aqueles que poderiam ser melhorados. No final da Retrospectiva da *Sprint*, o time deve ter identificado medidas de melhoria factíveis que ele implementará na próxima *Sprint*. Essas mudanças se tornam a adaptação para a inspeção empírica (SCHWABER e SUTHERLAND, 2009).

2.1.11 User Stories

Uma história, ou *User Story*, descreve uma funcionalidade que terá valor para um usuário do software. *User Stories* não são pertencentes ao *Scrum* em sua essência, mas já são tão comumente usadas com essa metodologia que diversos autores incluem seu conceito em tópicos iniciais de estudo de *Scrum*.

Um dos principais livros sobre histórias, *User Stories Applied*, de Mike Cohn (2004), já define seu uso ágil no subtítulo, "*for agile software development*". É importante notar que o livro tem prefácio de Kent Beck, um dos principais autores de *Extreme Programming* (ver seção 2.2), o que prova que, apesar de ser comumente usada em conjunto com Scrum, *User Stories* é uma técnica que pode ser combinada com qualquer metodologia ágil.

User Stories são compostas por três aspectos: uma descrição escrita da história, usada para planejamento e como lembrete; conversas sobre a história que servem para iluminar os detalhes; e testes que documentam os detalhes e podem ser usados para determinar quando uma história está completa (COHN, 2004).

As *User Stories* não devem ser muito grandes. Os detalhes podem ser descritos como histórias adicionais, e é melhor ter várias pequenas do que poucas muito grandes. O ideal é ter histórias que podem ser codificadas e testadas num período entre meio dia e duas semanas por um ou um par de programadores (COHN, 2004). Quando uma história é muito grande, ganha o nome de *Epic*, e pode ser dividida entre duas ou mais histórias de tamanho menor.

Os desenvolvedores geralmente quebram histórias tarefas para facilitar a distribuição do trabalho de implementação em toda a equipe, e permitir o acompanhamento dos progressos a nível de granularidade (SIMS, 2009). Entretanto, por serem técnicas, uma tarefa pode ser significativamente mais trabalhosa que outra. A integração torna-se muito complicada. Portanto, é melhor quebrar o trabalho em histórias do que em tarefas (JEFFRIES, 2010).

As histórias devem conter atributos que descrevem aspectos importantes da história, tais como seu número de *story points* (ver o conceito a seguir). Existem alguns conceitos que devem ser levados em consideração para a escrita de histórias de boa qualidade, que foram definidos por diversos autores. Eles são descritos a seguir.

2.1.11.1 Card, Conversation, Confirmation

Conceitos estabelecidos por Ron Jeffries (1999), que estabelecem aspectos críticos a serem tratados na história. Também chamado de 3C, esse aspectos devem ser lembrados no momento da criação da *User Story*.

- *Cards* — Histórias são escritas em cartões ou *post-its*. A sugestão dos cartões e *post-its* é, justamente, pelo fato de serem pequenos. E cartões pequenos naturalmente forçam histórias pequenas, de duas a três linhas no máximo.

- *Conversation* — A história escrita no cartão serve como um lembrete, uma maneira de identificar uma funcionalidade que foi conversada e discutida entre os clientes e desenvolvedores.

- *Confirmation* — Depois das funcionalidades terem sido discutidas e escritas nos cartões, o cliente define (implícita ou explicitamente) uma maneira de validar esse pedido. Geralmente essa confirmação é feita com testes de aceitação.

2.1.11.2 INVEST

Mnemônicos que descrevem atributos de uma história (WAKE, 2001). São eles:

- *Independent — User Stories* devem ser independentes uma das outras. Dependências entre histórias geram problemas de planejamento e priorização, sem falar que dificultam bastante nas estimativas.

- *Negotiable — User Stories* não são contratos, mas são lembretes para funcionalidades discutidas e negociadas entre o cliente e os desenvolvedores.

- *Valuable — User Stories* devem agregar valor para o cliente, para o negócio.

- *Estimatable* — Os desenvolvedores devem ser capazes de estimar o tamanho das *User Stories*. Geralmente *User Stories* incompletas ou muito grandes (complexas) são difíceis de serem estimadas. Portanto, devem ser discuscutidas pelo time e quebradas em *User Stories* menores quando necessário.

- *Small — User Stories* grandes dificultam as estimativas, bem como *User Stories* muito pequenas. Em algumas abordagens, o S significa "*sizable*", ou seja, do tamanho adequado, não sendo nem muito grande, nem demasiado pequena.

- *Testable — User Stories* devem ser possíveis de serem testadas. Um teste executado com sucesso prova que a *User Stories* foi desenvolvida com sucesso, atingindo as necessidades do cliente.

2.1.12 Story Points

Story Points é um método de estimativa (ver capítulo 2.1.13) ágil que ajuda o time a visualizar quando uma história estará terminada. Cada time define o tamanho do seu *story point.* Uma das forma é considerar que um *story point* é um dia ideal de trabalho, isto é, um dia sem interrupções como reuniões, *emails* ou ligações telefônicas (COHN, 2004).

Um *Story Point* é uma estimativa relativa de "tamanho" da atividade comparada com outra atividade no projeto. Portanto, espera-se que uma história de 10 pontos demore o dobro do tempo que uma história de 5 pontos (COHN, 2002).

2.1.13 Estimativas

Estimativas são formas de mensurar o tamanho de uma história, medida em *Story Points*, horas ou outra forma definida pelo time. São realizadas pelo *Scrum Team*, com base em comparação entre as histórias e experiências prévias, portanto, com o passar de tempo, e com equipes mais experientes, as estimativas se tornam mais precisas.

2.1.13.1 Planning Poker

Planning Poker (poker de planejamento) é uma abordagem para estimativa ágil. Para iniciar uma sessão de estimativa, o *Product Owner* ou o cliente lê uma *User Story*

ou descreve um recurso para os estimadores, que deve incluir todos na equipe. Cada estimador está segurando um baralho de cartas com valores. Os mais comumente utilizados são os primeiros números da sequência de Fibonacci (0, 1, 2, 3, 5, 8, 13, 20, 40 e 100), porém, podem ser utilizados outros valores que expressem a mesma ideia. Os valores representam o número de *Story Points*, dia ideal, ou outra unidade de estimativa que faça sentido para o time (SCHWABER e SUTHERLAND, 2010).

Os estimadores discutem a função, fazendo perguntas para o *Product Owner*, conforme necessário. Quando o recurso tiver sido amplamente debatido, cada estimador seleciona uma carta para representar a sua estimativa, sem mostrá-la aos demais. Todos os cartões são, então, revelados ao mesmo tempo. Se todos os estimadores houverem selecionado o mesmo valor, ele se torna a estimativa. Se não, os estimadores discutem as suas estimativas. As estimativas mais alta e baixa deve principalmente partilhar as suas razões. Após um debate, cada estimador seleciona novamente um cartão de estimativa e todas as cartas são novamente reveladas ao mesmo tempo (SCHWABER e SUTHERLAND, 2010).

O processo é repetido até que o consenso é alcançado ou até que os estimadores decidirem que a estimativa de um determinado item deve ser adiada até que informações adicionais possam ser adquiridas.

2.1.14 Acceptance Criteria

Em tradução literal, critérios de aceitação. São os requisitos mínimos para uma história ser considerada completa.

Em *Scrum*, um *Product Owner* nunca consideraria o trabalho de uma equipe completo até que ele ou ela tinha consultado os critérios de aceitação da história. Os critérios de aceitação são os requisitos que devem ser atendidos para que uma história possa ser considerada completa. Os critérios de aceitação são extremamente importantes no *Scrum* porque mostram o que um *Product Owner* espera e que o time precisa realizar. Se um requisito está nos critérios de aceitação, então ele precisa estar no produto a ser lançado (SCHWABER e BEEDLE, 2001).

Por que *Scrum* só aceita trabalho que é completo? Primeiro de tudo, um critério de aceitação implícito de qualquer história é que ela deve ser concluída durante o *Sprint* para o qual foi atribuída. Em segundo lugar, é comum para os times descobrirem que a parte final de um trabalho é seu ponto mais desafiador. Quando um *Product Owner* atribui pontos que não deveriam ser atribuídos, isso resulta em inflação de velocidade, o que faz com que a velocidade da equipe deixe de ser uma métrica confiável. Além disso, quando uma equipe ganha o crédito para uma história que deverá terminar no próximo *Sprint*, isso significa que a carga de trabalho da equipe é ainda maior do que o *Sprint Backlog* sugere (SCHWABER e BEEDLE, 2001).

Claramente, é do interesse da equipe ganhar crédito somente quando completamente merecido. Créditos parciais, com certo número de *Story Points*, só servem para minar a capacidade de um *Product Owner* de prever com precisão e adicionar riscos na construção do software (SCHWABER, 2004).

2.1.15 Velocidade

Em *Scrum*, a velocidade é a quantidade de trabalho do *Product Backlog* que uma equipe pode realizar em um *Sprint*. Isso pode ser estimado através da visualização *Sprints* anteriores, assumindo que a composição da equipe e duração *Sprint* são mantidas constantes. Também pode ser determinado *Sprint* por *Sprint*, utilizando o planejamento baseado no compromisso

Uma vez estabelecida, a velocidade pode ser usado para planejar projetos e liberação de previsão e datas de realização do produto.

2.1.16 Burndown Chart

Um *burndown chart* mostra o montante de trabalho restante através do tempo. Uma maneira de visualizar o montante de trabalho restante em qualquer ponto do tempo e o progresso do time em reduzir esse montante. Mostra a tendência do trabalho realizado, com a fonte de dados crus sendo o *Sprint Backlog* e o *Product Backlog* (COHN, 2002).

O trabalho restante é visto no eixo horizontal e os períodos de tempo (dias de um *Sprint* ou *Sprints*) são visto no eixo vertical. A intersecção de uma linha de tendência do trabalho restante e o eixo horizontal mostra o quanto do trabalho estará completo naquele ponto do tempo (COHN, 2002) . Um *burndown chart* nada mais é que um diagrama tradicional de Gantt com os eixos invertidos. Portanto, como em um diagrama de Gantt, serve para ilustrar os avanços das etapas de um projeto. A diferença é que no *burdown chart* não visualizamos cada tarefa de cada membro da equipe, como em Gantt, mas o todo do trabalho realizado pela equipe no espeço de tempo de uma iteração.

O *burndown chart* é a colisão da realidade (trabalho feito e velocidade com que está sendo feito) com o que foi planejado.

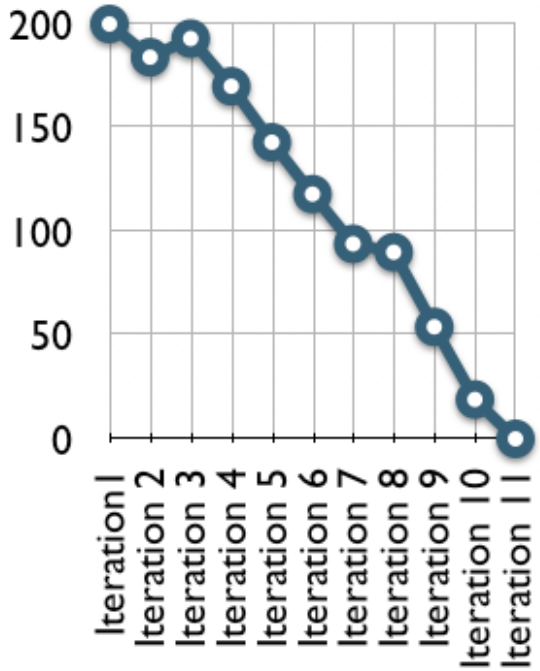

Figura 2.5: Exemplo de *Burndown Chart* (COHN, 2002)

Pode-se criar um artefato correlato, o *burnup chart*, que tem a mesma função, mas mostra linhas "subindo", somente trocando os eixos. A confecção de *burndown* ou *burnup*, por terem o mesmo papel, fica pela preferência e adaptação de cada equipe. O *burnup chart* mantém a estrutura tradicional de eixos dos diagramas de Gantt.

Podem também ser usados dois gráficos, um para o *Sprint* corrente e outro para o projeto como um todo.

2.1.17 Release

Em tradução literal, *release* significa lançamento.

A transição de um incremento do produto potencialmente entregável da equipe de desenvolvimento em uso rotineiro pelos clientes. *Releases* geralmente acontecem quando um ou mais *Sprints* resultaram em um produto com valor suficiente para compensar o custo para implantá-lo.

O produto é liberado por obrigações do cliente ou mercado. O lançamento equilibra funcionalidade, custo e requisitos de qualidade em relação aos compromissos de data (SCHWABER and BEEDLE, 2001).

2.1.18 Review Meeting

No final de cada *Sprint* uma reunião de revisão da mesma é realizada. Durante essa reunião, o *Scrum Team* mostra o que eles realizaram durante o *Sprint*. Normalmente isto toma a forma de demonstração das novas funcionalidades.

A reunião de revisão de *Sprint* é intencionalmente muito informal, normalmente com as regras proibindo o uso de slides e que não permite mais de duas horas de tempo de preparação para a reunião. A reunião de revisão de *Sprint* não deve se tornar uma distração ou desvio significativo para a equipe, mas sim, deve ser um resultado natural do *Sprint* (COHN, 2002).

Os participantes na revisão de *Sprint* tipicamente incluem o *Product Owner*, o *Scrum Team*, o *Scrum Master*, gerência, clientes e desenvolvedores de outros projetos.

Durante a revisão de *Sprint* do projeto é avaliado o objetivo do *Sprint*, determinado durante a *Sprint Planning*. Idealmente, a equipe completou cada item do *Product Backlog* trazidos para o *Sprint*.

2.1.19 Kanban ou Task Board

Kanban não é uma técnica que faz parte de Scrum, mas é normalmente utilizado em conjunto com essa metodologia. O Kanban é uma ferramenta de processo que se baseia em visualizar o fluxo de trabalho, dividindo-o em partes, escrevendo-o em cartões e colocando na parede. Por isso a interação com histórias é tão direta.

Kanban utiliza colunas nomeadas para identificar onde cada item está no fluxo de trabalho (Figura 2.6). O limite de trabalho em progresso (*work in progress*) deve ser associado a limites explícitos (KNIBERG e SKARIN, 2010), ou seja, cada coluna deve indicar quantos itens podem estar em progresso em cada estado do fluxo de trabalho.

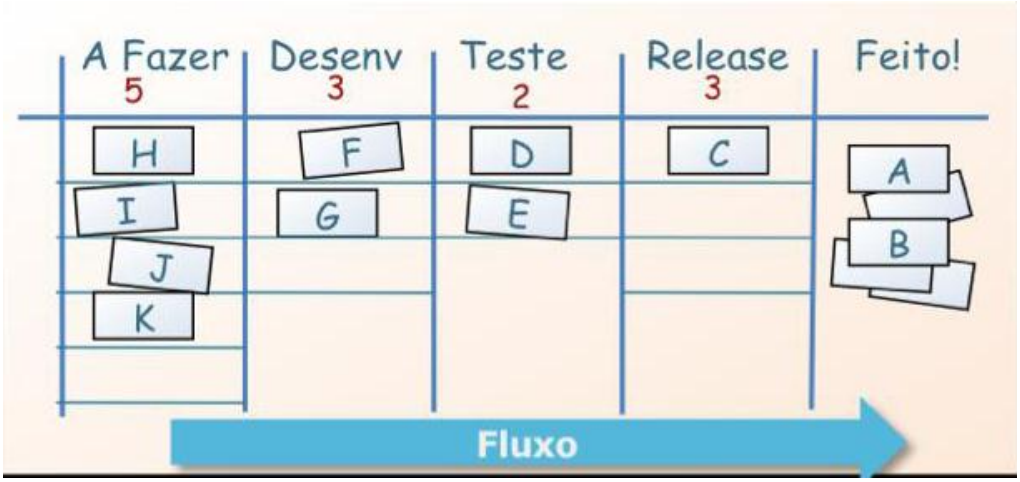

Figura 2.6: Exemplo de *Kanban* (KNIBERG e SKARIN, 2010)

O tempo de execução de cada tarefa deve ser acompanhado. Dessa forma, pode ser otimizado para tornar o processo o mais otimizado e com tempo de execução mais previsível possível. Assim, a interação com *Sprints* e iterações fica muito clara.

Os itens do *Backlog* estão na primeira coluna, nomeada de "A Fazer". No capítulo de relatos, ver-se-á que as outras colunas variam de time para time, dependendo das fases que estão integradas ao Scrum.

2.2 Extreme Programming

De acordo com Don Wells em sua página de referência sobre XP (*Extreme Programming*) (WELLS, 2009), *Extreme Programming* é um dos vários processos ágeis populares; já foi provado ser muito bem sucedido em muitas empresas de todos os tamanhos e indústrias do mundo inteiro. Adiante serão descritos relatos de aplicação bem sucedida de XP, o que colabora com a afirmação do autor.

Extreme Programming enfatiza o trabalho em equipe. Os gerentes, clientes e desenvolvedores são todos parceiros iguais em uma equipe colaborativa. *Extreme Programming* implementa um ambiente simples, mas eficaz, que permite as equipes tornam-se altamente produtivas. A equipe se autoorganiza em torno do problema para resolvê-lo o mais eficientemente possível.

Extreme Programming melhora um projeto de software em quatro formas essenciais: comunicação, simplicidade, *feedback* e respeito. Esses são os pilares sobre os quais a metodologia XP é sustentada (BECK, 1999):

- Comunicação: A maioria dos problemas que ocorrem nos projetos invariavelmente tem sua causa associada ao fato de alguém não ter informado alguma coisa muito importante para umapessoa que precisava saber. Dessa forma, a comunicação é o valor de maior importância no *XP*.

- Simplicidade: A simplicidade não é fácil. Uma das coisas mais difíceis de se fazer é não olhar para as coisas que serão necessárias implementar no dia seguinte, na semana seguinte e no mês seguinte. Deve ser implementado apenas aquilo que é necessário e realmente importa ser construído. Isto significa dizer

que as pessoas só devem se preocupar em solucionar hoje os problemas de hoje. A complexidade custa muito caro e tentar prever o futuro é muito difícil. É necessário aguardar o futuro para ver se está certo ou não.

- *Feedback*: A veracidade dos relatórios do estado atual das atividades é extremamente importante em XP. *Feedback* significa perguntar e aprender com as respostas. A única forma de saber a necessidade do cliente é perguntando a ele. O único modo de saber se um código faz o que ele se destin a fazer é testando-o. Quanto mais cedo se obter o *feedback*, mais tempo se terá disponível para reagir. A metodologia XP fornece um *feedback* rápido e frequente por parte dos seus seguidores.

- Coragem: Depois de se falar nos três valores, comunicação, simplicidade e *feedback*, é hora de se esforçar como nunca antes. Se o trabalho não for desempenhado na sua velocidade mais rápida possível, alguém mais o irá fazer, e ele vai lucrar no lugar. A coragem significa tomar as decisões na hora em que elas precisam ser tomadas. Se uma funcionalidade não está funcionando, ela deve ser consertada. Se algum código não parece estar bem escrito, ele deve ser *refatorado*. Se não será possível entregar tudo o que se havia prometido dentro do prazo acordado, o cliente deve ser informado. A coragem é uma virtude difícil de se aplicar. Ninguém gosta de estar errado ou quebrar um acordo. O único modo de consertar um engano é admitir que houve um engano e consertá-lo.

Programadores de um time que implementa *Extreme Programming* constantemente se comunicam com seus clientes e colegas programadores (BECK, 1999). Eles mantêm seu design simples e limpo, entregam o sistema para os clientes o mais cedo possível e implementam mudanças. Cada pequeno sucesso aprofunda seu respeito para as contribuições únicas de cada membro da equipe. Com essa fundação, programadores Extreme são capazes de responder com coragem a novos requisitos e tecnologia (WELLS, 2009).

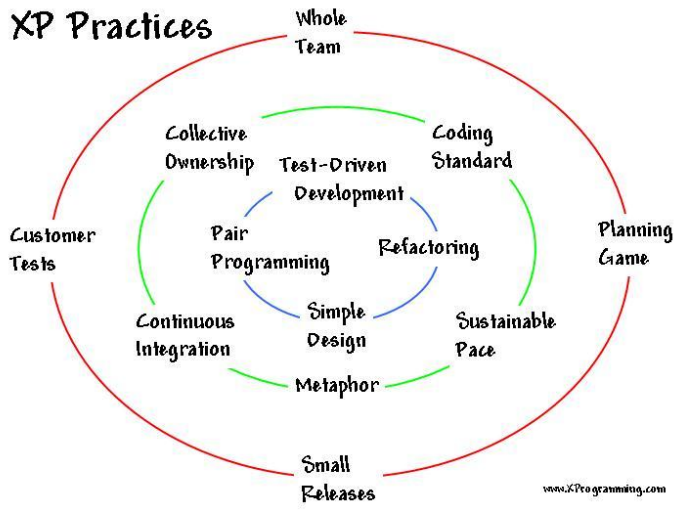

Figura 2.7: Práticas de *Extreme Programming* (JEFFRIES, 1999)

Extreme Programming (XP) é apoiado em práticas que formam a metodologia. A maioria das práticas XP também podem ser adotadas por desenvolvedores individualmente. Devido ao fato da metodologia XP ser muito bem detalhada e explicada, e não existir grandes dificuldades em seguir suas práticas, entretanto, é extremamente difícil e arriscado seguir todas as práticas rigorosamente e ao pé da letra de uma única vez (MARCHESI et al., 2002).

Cada prática pode desempenhar vários papéis. Por exemplo, o teste influencia o projeto da solução e encoraja a execução de pequenos experimentos controlados. Apenas escolher algumas práticas XP ao acaso, sem antes entender a relação existente entre elas, pode levar a resultados desastrosos. Por exemplo, um refinamento de código sem que tenha sido realizada uma rigorosa fase de testes anteriormente poderá resultar na introdução de defeitos tão sutis dentro do código que poderiam levar a extensas sessões de depuração para sua correção.

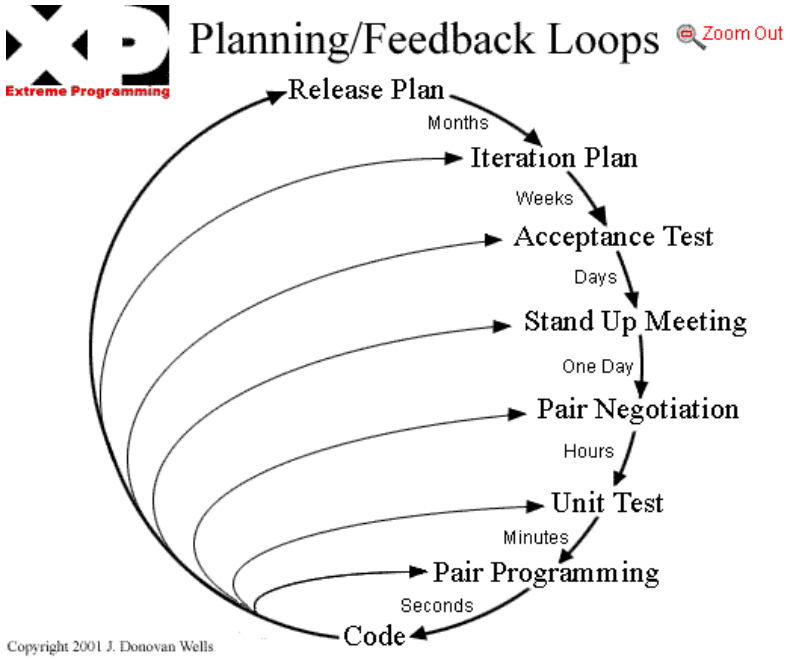

Figura 2.8: *Planning/Feedback Loops* de XP (WELLS, 2009)

A seguir, serão explanadas as práticas de *Extreme Programming* com maior detalhamento.

2.2.1 Planning Game

Prática para determinar rapidamente o alcance do próximo lançamento (*release*) combinando as prioridades do negócio e as estimativas técnicas. Se a realidade ultrapassar a planejamento, o plano deve ser atualizado (BECK, 1999).

As pessoas envolvidas com o negócio fazem decisões sobre escopo, prioridades, composição e datas dos *releases*, sempre apoiadas nas decisões da equipe técnica, que decide sobre estimativas, consequências técnicas em se desenvolver algo, processo e cronograma detalhado. As decisões são baseadas em diálogos, para chegar ao equilíbrio

entre o que é desejado (pelas pessoas de negócios) e o que é possível (pela equipe técnica).

2.2.2 Small Releases

Todo lançamento (*release*) deve ser tão pequeno quanto possível, contendo os requisitos de mais valor para o negócio. A *release* tem que fazer sentido como um todo, ou seja, não se pode implementar metade de um recurso e lançá-lo, somente para fazer o ciclo de lançamento mais ficar curto.

É melhor planejar um mês ou dois do que de seis meses ou um ano de cada vez. Uma empresa lançando software muito grandes para os clientes pode não ser capaz de lançar com tanta frequência, mas de qualquer forma deve ainda reduzir o seu ciclo tanto quanto possível (JEFFRIES, 1999).

2.2.3 Metáfora

São as descrições de um software sem a utilização de termos técnicos, com o intuito de guiar o desenvolvimento do software.

2.2.4 *Design* Simples

O sistema deve ser concebido o simples quanto for possível, em qualquer momento. Complexidade extra é removida assim que for descoberta.

O *design* correto para o software a é o que executa todos os testes, não possui lógica duplicada (inclusive duplicação escondida, como classe paralela e hierarquias), cobre todos os estados importantes para os programadores e tem o menor número possível de classes e métodos (BECK, 1999).

2.2.5 Refatoração

Refatoração é uma prática na qual são feitas pequenas mudanças para o código suportar novos requerimentos ou para manter o design o mais simples possível.

Refatoração é uma técnica disciplinada para reestruturar um corpo de código existente, alterando a sua estrutura interna sem alterar seu comportamento externo. Ela aplica uma série de transformações preservando o comportamento pequena. Cada transformação (ou seja, cada "*refactoring",* refatoração) faz pouco, mas uma sequência de transformações podem produzir uma reestruturação significativa. Já que cada refatoração é pequena, é menos provável a dar errado. O sistema também é mantido em pleno funcionamento após cada refatoração de pequeno porte, reduzindo as chances de que um sistema pode ficar seriamente quebrado durante a reestruturação (FOWLER, 2004).

A refatoração mantém a semântica do código, ou seja, após as mudanças, o código ainda funciona da mesma forma. Por exemplo, se o nome de uma operação de uma classe for mudado, todo o código do seu sistema que invoca aquela operação passará a referenciar o novo nome, e não o antigo (FOWLER, 2004).

A melhor maneira de pensar na refatoração é como uma forma disciplinada de melhorar tanto a qualidade do código como seu design.

2.2.6 Testes

No âmbito de *Extreme Progamming*, qualquer recurso do programa, sem um teste automatizado, simplesmente não existe. Programadores devem escrever *unit tests* (toda a aplicação de teste nas assinaturas de entradas e saídas de um sistema, feito para validar dados via E/S) para que a sua confiança no funcionamento do programa torne-se parte do próprio programa (JEFFRIES, 1999).

O resultado de testes integrados ao desenvolvimento do software é um que programa que se torna mais confiável ao longo do tempo, torna-se mais capaz de aceitar a mudança, não menos. Não há que se escrever um teste para cada método do programa, apenas os métodos que podem quebrar dependendo das entradas e saídas que deles são requeridas.

2.2.7 Pair Programming

Todo código de produção de ser escrito por duas pessoas trabalhando em uma mesma máquina, com um único teclado e único um mouse.

Existem duas funções em cada par. A função do parceiro com o teclado é pensar na melhor maneira de implementar a funcionalidade ou o método. A função do outro parceiro é de pensar mais estrategicamente: esta abordagem vai funcionar? Quais são os outros casos de teste que não se pode trabalhar ainda? Existe alguma maneira de simplificar todo o sistema para que o problema atual desaparecesse?

A formação das duplas é dinâmica. Se duas pessoas *pareiam* pela manhã, à tarde eles poderiam facilmente trabalhar com outras pessoas. Se um dos membros do time tem a responsabilidade de uma tarefa em uma área que lhe é estranha, pode pedir a alguém com experiência nessa área para parear. É uma forma de passar conhecimento para todos no time (BECK, 1999).

2.2.8 Propriedade Coletiva

Qualquer indivíduo da equipe de desenvolvimento tem condições de alterar qualquer parte do sistema a qualquer instante. A idéia por traz desta prática é que não deve existir um único responsável por uma parte de código, é necessário que o time desenvolva um senso de responsabilidade coletiva. A *Pair Programming* auxilia os colegas de dupla a demonstrar seu comprometimento com a qualidade do trabalho realizado em parceria.

2.2.9 Continuous Integration

Em tradução literal, “integração contínua”. Prática de *Extreme Programming*, onde os membros do time devem integrar o seu trabalho frequentemente.

A definição da frequência dependerá de cada time, sendo que diversos fatores devem ser levados em consideração, como tamanho da equipe, grau de distribuição, entre outros. Com *Continuous Integration* segue-se o princípio de *agile* de que quanto mais cedo forem detectados os problemas, mais simples solucioná-los. Devem ser verificadas quebras de *build (*versão "*compilada*" de um software ou parte dele que contém um conjunto de recursos que poderão integrar o produto final) e, de preferência, deve ser feito uso de um *automated build* (*build* automatizado) para a construção.

2.2.10 Semana de 40 horas (*Susteinable Pace*)

Extreme Programming assume que não se deve fazer horas extras constantemente. Caso seja necessário trabalhar mais de 40 horas pela segunda semana consecutiva, existe um problema sério no projeto que deve ser resolvido não com aumento de horas trabalhadas, mas com melhor planejamento, por exemplo.

Essa prática procura ratificar o foco nas pessoas e não em processos e planejamentos. Caso seja necessário, os planos devem ser alterados, ao invés de sobrecarregar as pessoas (BECK, 1999).

2.2.11 On-Site Customer

Um cliente real deve se sentar com a equipe disponível para responder perguntas, resolver conflitos e definir prioridades em pequena escala. O "cliente real" é alguém que vai realmente usar o sistema quando ele está em produção.

A objeção a essa regra é que os usuários reais do sistema em desenvolvimento são valiosa demais para ficarem disponíveis para o time. Os gerentes têm que decidir o que é mais valioso, tendo o software de trabalho mais cedo e melhor ou ter a saída de uma ou duas pessoas. De qualquer forma, o time não consegue fazer perguntar para o cliente 40 horas por semana. O *on-site costumer* terá a desvantagem de estar fisicamente separadas de seus colegas, mas provavelmente ainda poderá ter tempo para fazer seu trabalho normalmente (BECK, 1999).

2.2.12 Coding Standards

O código-fonte é a forma primária de comunicação dentro de um projeto. As requisições por funcionalidades e relatórios vão e vêm, mas o projeto nasce e morre com o software. Enquanto houver alguém realizando a manutenção do software, o projeto se manterá vivo. O *Coding Standards* é um conjunto de convenções que regulamentam a formatação e a estrutura dos códigos. Eles descrevem as boas práticas de programação que cada membro do time adiciona ao longo dos projetos.

As vantagens dos padrões de código é que eles representam o estilo de programação de toda a equipe de desenvolvimento, por esta razão utilizar os padrões de código agilizam a programação economizando tempo se compararmos com o tempo gasto para a compreensão de um código sem endentação e nomes de variáveis sem significado algum para o contexto do programa (BECK, 1999).

Muitas linguagens possuem seus próprios *Coding Standards*, o ideal é começar por esses padrões e gradualmente ir adaptando de acordo com as necessidades do projeto. Acima de tudo, o time de desenvolvimento precisa assumir o compromisso em seguir os padrões para que esta prática realmente tenha o efeito esperado. A prática de *Coding Standards* coopera para as práticas de *Refactoring*, *Pair Programming* e *Collective Code Ownership* (BECK, 1999).

2.2.13 Test-Driven Development

Em tradução literal, *Test-Driven Development* (TDD) significa *desenvolvimento orientado a testes.* TDD é um conceito que ganhou força com a popularização do *Extreme Programming*, mas que pode ser usado como uma prática independente da metodologia.

O objetivo do *Test-Driven Development*, segundo Kent Beck, é um código claro e funcional (BECK, 2002). Muitas forças, no entanto, desviam os desenvolvedores de escrever código claro e, muitas vezes, código funcional. Por isso, o desenvolvimento é feito juntamente de testes automatizados. TDD diz que novo código deve ser escrito somente se um teste falhar e que as duplicações devem ser eliminadas.

As duas regras do *Test-Driven Development* – código quando teste falhar e eliminar duplicações – implicam numa ordem das tarefas de programação:

- *Red* (vermelho): escrever um pequeno teste que a princípio não passa e talvez não compile;
- *Green* (verde): fazer o pequeno teste passar rapidamente, cometendo quantos "pecados" de código forem necessários;
- *Refactor* (refatorar): refatoração do código, eliminando qualquer duplicação criada ao fazer o teste meramente passar.

Test-Driven Development tem também implicações sociais, entre elas: com a densidade de defeitos reduzidas, a *Quality Assurance (*um programa de acompanhamento sistemático e avaliação dos diferentes aspectos de um projeto, serviço ou facilidade para garantir que os padrões de qualidade estão sendo cumpridos) do projeto pode ter um trabalho mais pró-ativo; gerentes podem estimar com precisão suficiente para envolver clientes no processo; engenheiros de software podem trabalhar em colaboração minuto-a-minuto; e há software potencialmente entregável com novas funcionalidades todos os dias (BECK, 2002).

2.2.14 Time Coeso (*Whole Team*)

Todos os contribuintes para um projeto XP sentam-se juntos, e são membros membros de um Time Coeso. O time compartilha os objetivos do projeto e a responsabilidade para atingi-los. Esse time deve incluir um representante comercial, o "Cliente", que estabelece os requisitos, as prioridades, e dirige o projeto. É melhor se o cliente ou um de seus assessores é um usuário final real que conhece o domínio e que é necessário.

O time terá, naturalmente, os programadores. A equipe pode incluir testes, que ajudam o cliente a definir os testes de aceitação do cliente. Os analistas podem servir como auxiliares para o cliente, ajudando a definir os requisitos. Existe geralmente um "treinador", que ajuda a equipe e facilita o processo. Não pode ser um gerente, fornecendo recursos, movimentação de comunicação externa e coordenando as atividades.

Nenhum destes papéis é necessariamente a propriedade exclusiva de um único indivíduo. Todos em uma equipe XP contribuem de qualquer forma que puderem. Os melhores Times não têm especialistas, apenas contribuintes em geral, com habilidades especiais (JEFFRIES, 1999).

2.2.15 Testes de Aceitação (*Customer Tests*)

Como parte da apresentação de cada recurso desejado, o cliente *XP* define um ou mais testes de aceitação automatizados para mostrar que o recurso está funcionando. A equipe constrói estes testes e os usa para provar a si mesma e para o cliente, que o

recurso está sendo aplicado corretamente. Automação é importante, porque na pressão do tempo, testes manuais são ignorados.

Os melhores Times *XP* tratam os seus testes de aceitação da mesma maneira que fazem os testes programador: uma vez que o teste é executado, a equipe mantém a funcionar corretamente depois disso. Isso significa que o sistema só melhora, nunca retrocede (JEFFRIES, 1999).

3 METODOLOGIAS ÁGEIS NO MERCADO

Cada vez mais as metodologias ágeis estão ganhando espaço no mercado. Relatos novos parecem a cada dia, mostrando sucessos e fracassos de projetos, implementações e interpretações diferentes, cada um com seu olhar sob a teoria.

Nesta seção será descrito como as metodologias ágeis estão inseridas no mercado atual, com base em relatos e em observações da forma de trabalho de empresas, complementadas com entrevistas aos gerentes dos projetos que foram avaliados presencialmente. A partir desses dados, serão identificadas as metodologias, práticas, ferramentas e processos mais utilizados, bem como os que são descritos na teoria, mas não aplicados na prática. Será verificada a terminologia utilizada nas empresas e ao quê exatamente cada termo se refere; se os profissionais veem a prática da mesma forma que os autores definiram na teoria ou se há um novo sentido para o que foi definido primeiramente. Em alguns relatos são utilizados termos que são essenciais à compreensão, mas não estão diretamente ligados a metodologias ágeis. Dessa forma, uma descrição um pouco mais detalhada de termos utilizados nesse capítulo encontra-se nos Anexos do presente trabalho.

Assim, nessa seção é descrito o ferramental necessário para a interpretação dos dados que será feita na seção 5, de forma a extrair as práticas mais usadas e servir como guia para iniciantes no estudo dos diversos aspectos das metodologias ágeis.

3.1 Relatos

Serão relatados nove relatos de aplicações de metodologias ágeis, divididos em duas categorias: (a) aqueles descritos em artigos acadêmicos ou papers de convenções, sendo eles *Telessaúde*, *Software House*, *Sistema de Gestão de Ensino*, *Endeeper*, *Instituto Atlântico* e *SAF-ACS*, tendo maior riqueza de detalhes e prendendo-se mais à teoria; e (b) os descritos informalmente em blogs e sites da Web, sendo eles *Orbisat*, *Globo.com* e *Agência AG2*, que não são relatos oficiais das empresas, porém, dão uma visão de como as metodologias são implementadas. Alguns dos *papers* aqui descritos interligam-se com o trabalho relatado na seção 3.2, de observação e pesquisa de campo, sendo ambos complementares e podendo-se tirar conclusões conjuntas pelas diversas fontes de informação.

3.1.1 Telessaúde

O relato descrito por Luna et al. (2008) mostra a aplicação do Telessaúde utilizada pelos núcleos de telemedicina e telessaúde da Universidade Federal Rural de

Pernambuco (UFRPE), que foi desenvolvida utilizando um modelo de Desenvolvimento Distribuído de Software (DDS), utilizando a metodologia ágil *Scrum.*

A primeira motivação para a utilização do *Scrum* nesse caso em específico já se encontra em desarmonia com a teoria. A atividade de desenvolvimento era essencialmente distribuída, com equipes separadas geograficamente. Isso em si já vai contra os princípios ágeis teóricos, pois a indicação é de que todos os integrantes do *time* estejam na mesma sala e sempre presentes, até para um melhor aproveitamento dos artefatos, tais como os cartões de história e *Kanbam*. Essa é uma exigência básica de desenvolvimento com XP, o que não é aplicado nesse relato. No entanto, é notável que muitos projetos utilizem Scrum para atender necessidades de DDS. Essa ideia será desenvolvida no capítulo 5.

Foi utilizada uma ferramenta de apoio à gerência de forma que, mesmo com o DDS, todos os membros do time pudessem ter acompanhamento do projeto e, assim, estar inseridos dentro do contexto da metodologia ágil.

O *paper* cita que o trabalho utilizou boas práticas de *Extreme Programming*, mas que o foco do artigo é descrever a forma de gerência utilizada e, por isso, estaria centrado em Scrum. Havia dias e horários específicos para reuniões, que o artigo não define quais são, tampouco cita o uso de *Daily Meetings*. Até pela característica distribuída, a comunicação se deu por informes e relatórios, o que não é característica das metodologias ágeis (que privilegiam comunicação face a face e meios rápidos de troca de informações). As análises comparativas foram acompanhadas por meio de *burndown charts*. É citado que por meio dele acompanhou-se Velocidade e Execução, embora não seja determinado a que o termo "Execução" se refere no contexto.

Apesar do *paper* apresentar um trabalho com *Scrum*, foi também utilizada uma prática de *Extreme Programming*, a integração contínua (ou *continuous integration*). Para essa tarefa, foi utilizado o *framework* para *build* contínuo *Cruise Control.*

Pela natureza distribuída do trabalho, ocorreram poucos encontros presenciais, o que já impossibilita muitas das características teóricas do *Scrum*. Mesmo assim, foram realizados dois encontros semanais com dias e horários pré-determinados. A comunicação se dava primordialmente por e-mails e Google Docs, que apresenta um modelo colaborativo, próximo do modelo de *wiki*.

O Google Docs foi utilizado também para gerência de requisitos e de atividades. Foi criada uma planilha para esse controle, que é chamada *Ticket Tracking*. Não há citação sobre a forma com que a análise foi desenvolvida, se esses "tickets" são *stories* ou se houve análise prévia.

O trabalho foi desenvolvido em seis *Sprints* (sem citar o tamanho da *Sprint*), onde cada uma delas se preocupou com uma parte específica do trabalho. É citado, no entanto, que o cliente só recebeu o produto final após o sexto *Sprint*, dando o seu aceite, apesar de não terem sido concluídos todos os objetivos e o projeto estar ainda em fase de implementação.

O projeto utilizou *burndown chart* para a medição da velocidade. É citado que foram feitos testes, e que estes eram realizados por membros da fábrica que não faziam parte

do desenvolvimento, mas não está claro se o processo de testes foi integrado ao desenvolvimento ou posterior a esse.

A mesuração do tamanho final do projeto foi em Linhas de Código, uma métrica estabelecida pelo PMBok, não tendo nenhuma ligação com metodologias ágeis. No entanto, há um pararelo dessas Linhas de Código com os *Story Points*, que foram determinados a partir de *Planning Poker* com o time.

3.1.2 Software House

Este relato descreve a aplicação de *Scrum* em uma *Software House* e foi objeto de estudo de alunos da Universidade de Passo Fundo (BRUNHERA e ZANATA, 2010). Essa *Software House*, também localizada em Passo Fundo, atende a segmentos de prestação de serviços, instituições de ensino e pesquisa e cooperativas, com softwares desenvolvidos sob encomenda e soluções em servidores.

O cliente entra em contato com a *Software House*, por visitação, e-mail ou telefone e, com isso, cria-se um artefato de entrada. A partir disso, cria-se uma pasta de documentação de todas as atividades relacionadas ao cliente. Isso pode, a princípio, parecer contra o que dizem os métodos ágeis, mas é importante lembrar que, por ser uma empresa, ela deve manter alguns cadastros burocráticos de seus clientes.

Depois desse primeiro contato, é feita a visita para levantamento de requisitos e elaboração de proposta de trabalho. A empresa trabalha com proposta fechada, com cálculo de orçamento antes de o trabalho começar e com prazos, ou seja, já se tem a primeira diferença quanto à forma de tratamento do *Scrum*: a análise não é integrada com o desenvolvimento e a entrega é feita somente no final do projeto, sem incrementos de produto. O cálculo das horas de trabalho é feito com base na análise, não nos *Story Points* definidos pela equipe, e, a partir dela, é fechado um preço fixo a ser pago pelo projeto.

Após essa análise preliminar que leva ao fechamento de orçamento, existe a análise aprofundada dos requisitos pelo time, que gera um *Product Backlog*. Além disso, são gerados muitos outros documentos, como um Plano de Risco, Modelo Entidade Relacionamento e Cronograma de desenvolvimento. Esses últimos nada têm a ver com o desenvolvimento ágil. Dependendo do que será desenvolvido, o Plano de Risco e o Modelo Entidade Relacionamento podem ajudar, se forem sucintos o suficiente, dentro dos princípios de *Agile Modeling*, no entando, um cronograma de desenvolvimento detalhado logo no início do projeto fere os princípios incrementais de Scrum.

A partir do *Product Backlog*, então, são discutidos quais requisitos farão parte do *Sprint*, também chamado pela equipe de *interação*, talvez numa confusão com a palavra *iteração*, muitas vezes usada (corretamente) como sinônimo do *Sprint*, considerando a natureza iterativa da metodologia.

A equipe conduz Reuniões Diárias (*Daily Meetings*), de duração máxima de 15 minutos, e utiliza *Task Boards*. No entanto, é citado que esses artefatos são feitos para que o *Scrum Master* tenha conhecimento do que está sendo desenvolvido ao longo do projeto, e não para o time todo. De qualquer forma, o *Scrum Master* remove os impedimentos para que os integrantes do time possam trabalhar em suas tarefas assumidas.

O *Sprint* dura de duas a quatro semanas (dependendo do projeto) e, ao final, gera um *Product Increment*, ou Incremento de Produto, completamente funcional, mas que terá sua entrega somente ao final do projeto. O *Task Board* e o *Product Backlog* são atualizados, o Time e *Scrum Master* geram um novo *Sprint Backlog* com a participação opcional dos *Product Owners.*

Em alguns projetos, não há como realizar *Daily Scrum*, sendo que o time opta por três dias da semana para a relização das reuniões. O *Product Backlog* e o *Sprint Backlog* ficam disponíveis pelo Google Docs para que toda a equipe tenha acesso, além da sua forma física na parede da sala do projeto (Figura seguinte).

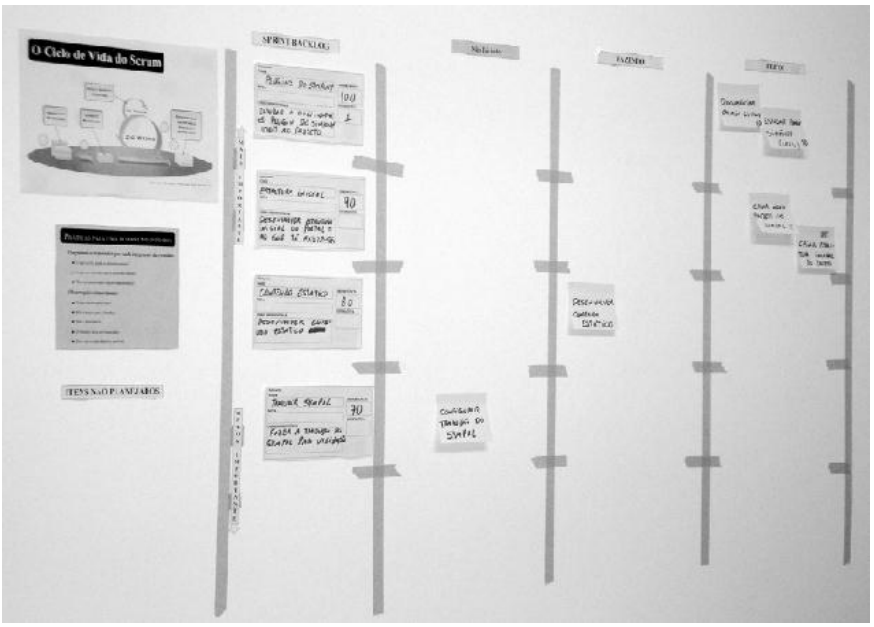

Figura 3.1: *Sprint Backlog* no *Task Board* (BRUNHERA e ZANATA, 2010)

O cliente costuma ter algum envolvimento: são citados casos em que ele participa de reuniões de final de *Sprint*, mas não é a regra, pois em outros casos há envolvimento esporádico, no meio do projeto e ao seu final. O time consistia apenas nos desenvolvedores: as equipes de teste e design não estavam integradas ao *Scrum*, e a análise, como já citado, é feita preliminarmente.

Pode-se ver que esse é um dos trabalhos em que o time utilizou a nomenclatura mais corretamente, apesar de um pequeno deslize de interpretação, possivelmente causado por confusão gramatical e não por falta de conhecimento. No entanto, alguns princípios básicos não são aplicados, como a equipe totalmente envolvida e sem hierarquias (o time reporta-se muito ao *Scrum Master)*, há excesso de documentação e a análise não é desenvolvida ao longo do processo. Não é citado o desenvolvimento de histórias, embora possamos ver pela Figura 3.1 que existem cartões com histórias no *Task Board*. De qualquer forma, há modelagem do projeto, utilizando-se de UML e outras técnicas que não têm combinação usual com *Scrum*.

3.1.3 Sistema de Gestão de Ensino

Este relato é descrito por Savoine et al. (2009).Como o próprio título do trabalho explicita, são utilizadas as metodologias *Extreme Programming* e *Scrum* no projeto estudado.

A decisão pelas metodologias ágeis se deu devido a uma adequação às necessidades dos modelos de negócio atuais. O projeto implementado é um *software* de gestão de ensino, em uma fábrica de *software* onde já eram desenvolvidos outros produtos.

Não há descrição do motivo para eles terem escolhido *Scrum* e *XP* dentre as tantas metodologias ágeis existentes, somente explicita-se que decidiram pela combinação de duas práticas, pois *XP* é voltada para implementação e *Scrum* foca em gerenciamento e planejamento. No entanto, a equipe analisou as práticas das metodologias escolhidas, selecionando algumas para serem utilizadas no projeto, explicitando quais seriam usadas e quais não seriam.

Das práticas de Scrum, a equipe utilizou *Product Backlog*, *Sprints* (porém, com duração de duas semanas), *Sprints Backlog*, *Sprint Retrospective*, *Scrum Master* e *Daily Scrum Meeting.* As nomenclaturas das metodologias utilizadas no projeto são as mesmas utilizadas na teoria.

Em *Extreme Programming*, foram utilizadas as práticas *Código Coletivo*, *Refatoração*, *Simplicidade* (prática essa que está descrita em várias metodologias, sendo inclusive um dos princípios do *Agile Modelling*), *Foco na Codificação, Integração Contínua*, *Programação Pareada* e *Testes.*

Nota-se também que houve uma interpretação livre dos *Testes* do *Extreme Programming.* É descrito no *paper* que "após o término de cada *Sprint* [...] é repassado o código compilado em executáveis para o departamento de testes". Ou seja, a equipe não utiliza a prática de testes unitários que garantem a eficiência do código, tampouco fazem uso de *Test Driven Development* (ver seção 2). Também não há integração da equipe de Qualidade de Software com os desenvolvedores, o que vai contra as práticas tanto de *Extreme Programming* como de *Scrum*.

Percebe-se também que, por decisão de projeto, decidiram não aplicar duas práticas: a *semana de 40 horas* de *Extreme Programming* e uma prática que a equipe denominou "cliente sempre presente", que podemos identificar como sendo o *On-Site Costumer* de *Exteme Programming* ou o *Product Owner* de *Scrum*.

A equipe optou por utilizar uma ferramenta para gerencimento do projeto e do código-fonte, mas não citou o nome da ferramenta. Com ajuda dela e das práticas de *XP*, obteve-se uma boa qualidade de código. As métricas da ferramenta não são as tratadas pela teoria de *Extreme Programming*, tratando de métricas de herança e número de linhas. O *Scrum Master* também a utilizou para gerenciar quais desenvolvedores estavam refatorando o código a cada momento (também não é por padrão uma das preocupações de gerencia do *Scrum Master).*

A ferramenta também foi usada para realizar integração contínua (tal prática necessita de ferramentas para que a integração seja automática), através da opção *Source Control*. É descrito no *paper* que houve um ganho de tempo pela prática. Esse é

um dos pontos de vantagem, mas a teoria destaca que o maior ganho de *Continuos Integration* é a qualidade do software.

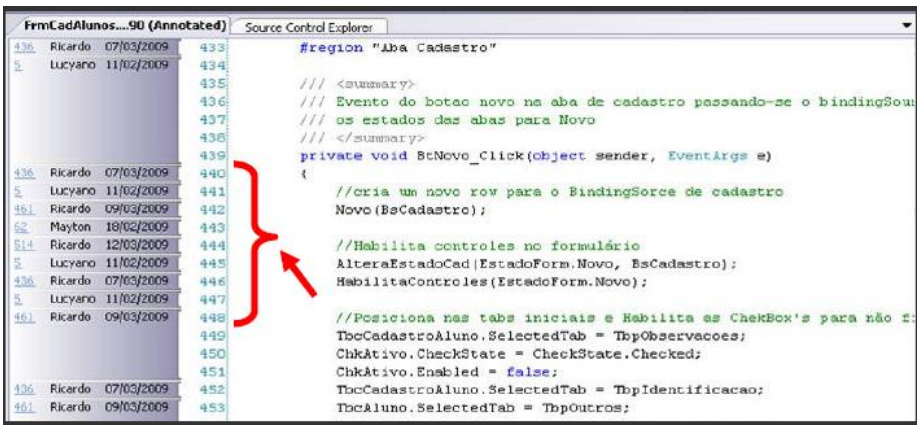

Figura 3.2: Refatoração do código (SAVOINE et al., 2009)

O trabalho é dividido em *Work Itens*, que são desenvolvidos numa iteração. A teoria não engloba esse nome, porém, pela descrição, pode-se constatar que esse é uma renomeação para *Stories*. Não é descrito, porém, um processo de análise, a forma como esses *Work Itens* seriam desenvolvidos.

Também é citado no trabalho que não foi feita análise de riscos, pois isso impactaria na desejada flexibilidade, que foi um dos principais motivos para a adoção de metodologias ágeis pela equipe. As histórias foram acordadas com o cliente e, portanto, previamente definidas, não sendo o processo de análise de negócios integrado aos *Sprints* de desenvolvimento.

3.1.4 Endeeper

Esse relato foi desenvolvido pela empresa Endeeper e pesquisadores do Intituto de Informática da Universidade Federal do Rio Grande do Sul para o I Workshop de Gerenciamento de Projetos de Software (CASTRO et al., 2008). Trata-se da transformação da ferramenta Petroledge, passando-a de projeto de pesquisa para produto comercial. A forma como a Endeeper utiliza metodologias ágeis será mais desenvolvida na análise da pesquisa de campo, no subcapítulo 3.2. O sistema Petroledge auxilia na tarefa de descrição e gerencia análises de descrições petrográficas de rochas-reservatório de petróleo.

O trabalho teve origem em um projeto de pesquisa e precisava incorporar novos requisitos para se tornar um produto comercial. De acordo com o *paper*, havia a necessidade de desenvolver uma cultura de projetos compatível com uma empresa de pequeno porte, e que garantisse o atendimento dos requisitos e do alto nível de qualidade exigido pelas companhias de petróleo.

Assim, foi dedido pela adoção de uma combinação das metodologias ágeis *Scrum* e *Extreme Programming, xP@Scrum* (KANE, 2002), para a implementação de CommomKADS. Esse último tem um ciclo de vida em espiral, compatível com as metodologias ágeis, e propõe uma série de modelos para a elaboração de um sistema de conhecimento. As práticas ágeis permitiram que o desenvolvimento fosse incremental e contribuíram para o aprendizado e aceitação da equipe.

É interessante notar que o projeto pretendia a implementação de uma metodologia de processos, não simplesmente o desenvolvimento de um software (embora a adaptação do software acadêmico para um produto comercial também fosse parte do escopo). Isso demonstra a abrangência das metodologias ágeis, que vão muito além do âmbito de puro desenvolvimento, podendo ser aplicadas em projetos de diversas frentes.

Foram utilizadas reuniões diárias de 20 minutos, desenvolvimento em duplas, *refactoring*, testes em paralelo com o desenvolvimento e disponibilidade total dos gerentes. Além de *refactoring*, que utiliza o nome da teoria, podemos identificar *Daily Scrum* (apesar de com um tempo de 5 minutos maior do que o indicado na metodologia), *Pair Programming*, testes (do âmbito de *XP*) e *On-site Costumer*/*Produtc Owner*.

Diferente do ciclo normal de CommomKADS, foi aplicado um ciclo ágil de três estágios, incorporando o quarto estágio, de riscos, ao longo dos demais. O estágio de revisão identificou os módulos a serem desenvolvidos, definiu casos de uso (notar que não há histórias, *backlog* ou outros artefatos ágeis) e estabeleceu o controle de qualidade. No estágio de planejamento, deve-se elaborar um plano de projeto tradicional, segundo CommomKADS. No entanto, no projeto foram apenas descritas as atividades sem detalhamento, apenas com seus prazos limite e responsáveis.

O estágio de monitoração foi feito continuamente, através das Reuniões Diárias e semanais (a *Sprint* teve duração de uma semana), além de relatórios gerados automaticamente pelo software de controle de versão. Como o estágio de risco foi incorporado e foi citado que sempre que um risco era identificado, era prontamente eliminado, ver-se-á que não houve modelagem. Isso não deixa de estar em concordância com *Scrum*, sendo o papel do *Scrum Master* controlar os riscos e eliminar os impedimentos de trabalho do time.

No projeto, é bem claro que foram utilizados somente alguns conceitos ágeis para a organização da equipe. A nomenclatura não é utilizada, mas pode-se perceber o conceito iterativo das metodologias ágeis na aplicação.

3.1.5 Instituto Atlântico

O *paper* "Integração de *Story Points* e *Use Case Points* em Projetos Baseados em Scrum e CMMI" (MARÇAL et al., 2009) foi escrito por integrantes do Intituto Atlântico para o VII Simpósito Brasileiro de Qualidade de Software. Ele mostra um relato de interligação de metodologias ágeis e modelos de qualidade.

O Instituto Atlântico utiliza o modelo de maturidade CMMI, e para atingir os níveis 4 e 5 do mesmo, faz uso de *Six Sigma*. De acordo com Ken Schwaber, em (SCHWABER, 2004), *Scrum* cobre até parte do nível 3 de CMMI. No Atlântico (forma como a empresa é referida no *paper* e, portanto, será adotada também neste trabalho), a gestão quantitativa de processos é realizada por meio de *baselines* de desempenho que determinam a situação atual da organização, sendo Produtividade e a Densidade de Defeitos as duas principais.

A *baseline* de Produtividade é calculada por uma relação de esforço e tamanho, sendo que o tamanho é medido em *Use Case Points*. Esse é o motivo pelo qual, mesmo

utilizando metodologias ágeis e seus métodos de estimativa (no caso, *Story Points*), deve haver a conversão para *UCP*, para manter as métricas da empresa.

O projeto DMADV (sigla não definida no *paper* que descreve o projeto) foi iniciado com a necessidade de um processo mais ágil e que, ao mesmo tempo, fosse aderente ao CMMI. Dessa forma, foram priorizadas práticas ágeis de análise e gestão. As práticas de análise e projeto adotaram os princípios da Modelagem Ágil (*Agile Modeling*). As práticas de gestão seguem o processo definido pela metodologia *Scrum*.

A proposta de integração de *Use Case Points* com *Story Points* está sendo aplicada por eles em um projeto real de desenvolvimento de um sistema de Gestão de Suprimentos para um cliente da indústria têxtil. O projeto possui requisitos extremamente voláteis, os quais são definidos ao longo do projeto com grande envolvimento do cliente. O *Product Owner* participa ativamente da construção do *Product Backlog* e das reuniões de Planejamento da *Sprint*. Durante as reuniões de Planejamento da *Sprint*, os requisitos priorizados são estimados em *Story Points*.

As estimativas das funcionalidades do sistema são inicialmente realizadas em *Story Points* durante a reunião de *Sprint Planning*. A técnica de *Planning Poker* é usada atrelada a sugestões iniciais construídas pelo time para a estimativa em *Story Points*. Para apoio logístico, foi utilizado o framework de desenvolvimento *jCompany (*www.powerlogic.com.br).

As estimativas em *Story Points* são convertidas para *Use Case Points*, usando-se a ferramenta organizacional para estimativas. Nesta etapa, as UCPs são calculadas derivando-se as complexidades dos casos de uso a partir das *Story Points*.

Tabela 3.1: Conversão de *Story Points* em complexidade de *Use Cases*

Story Points	Complexidade
1, 2 e 3	Simples
5 e 8	Médio
13	Complexo
21 e 34	N-Complexo

Fonte: MARÇAL et al., 2009

Tendo essa análise de conversão, a etapa seguinte foi composta da comparação e avaliação a conversão de *Story Points* em *Use Case Points* e, com isso, a construção de um modelo consistente para gerar número de transações a partir de *StoryPoints*. A complexidade foi estimada a partir da quantidade real de transações do caso de uso, seguindo o procedimento organizacional para contagem de transações.

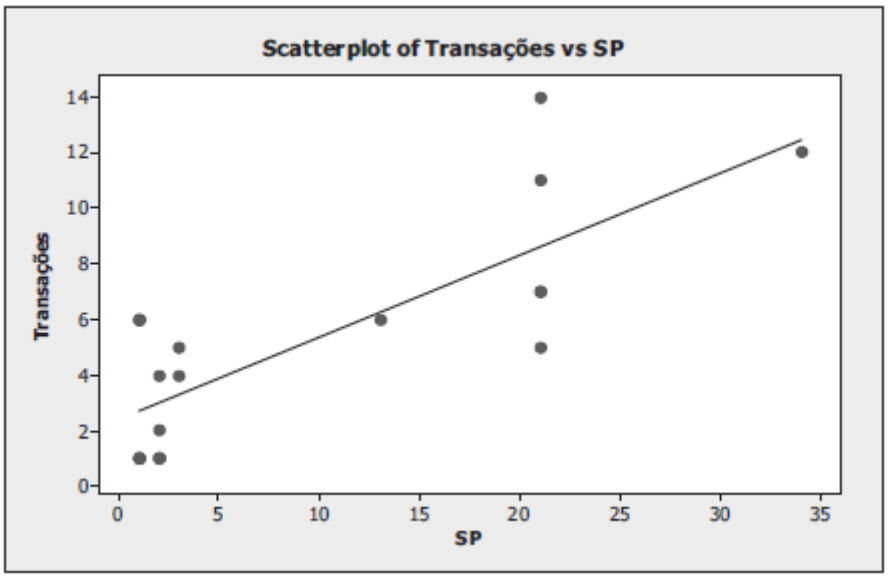

Figura 3.3: Gráfico de transções *versus Story Points* (MARÇAL et al., 2009)

Os seguintes passos foram, então, utilizados por eles ao longo do processo de desenvolvimento: o time realiza as estimativas de cada caso de uso em *Story Points*; o modelo é utilizado para converter os *Story Points* em transações; e os números de transações obtidos alimentam uma ferramenta de estimativas, já utilizada anteriormente pela organização, para o cálculo dos *Use Case Points*. Os *Use Case Points* são utilizados para o planejamento e acompanhamento do projeto, bem como para a obtenção de dados históricos.

Apesar de não ser citado no *paper*, pode-se notar muitas semelhanças nos sistemas de conversão proposto pelo Intituto Atlântico com o sistema descrito por Mike Cohn no artigo "*Estimating with Use Case Points*" (COHN, 2005), onde existem tabelas e fórmulas para a utilização de *Use Cases* e, dessa forma, ainda fazer as estimativas de forma ágil. No entanto, o trabalho do Atlântico vai além quando converte totalmente os *use cases* para histórias e depois faz o caminho de volta, retornado-as a *use cases*.

A empresa utiliza nomenclatura correta e tem muitas características de acordo com o que é descrito na teoria de *Scrum*: desde o *Product Owner* bastante presente até os requisitos voláteis. Não é citado como é feito o desenvolvimento e, pelo que se pode inferir do *paper*, não são utilizados outros artefatos como *Task Board* ou cartões. Não fica claro como os requisitos são levantados, nem a fase de testes, ou quando é dado o *Done*.

Podemos questionar o motivo de tantas conversões, se isso não acaba por não cumprir o principal objetivo das metodologias ágeis, que é aumentar a dinâmica do processo de desenvolvimento, já que acaba engessando e adicionando etapas burocráticas no processo. No entanto, pode ser de extrema validade para equipes que desejam, por diversos motivos, aderir às vantagens das metodologias ágeis e iniciar um processo de migração, mesmo que os processos da empresa ainda exijam um grau de documentação e certa burocracia que vão de encontro ao que é explicitado na teoria das metodologias utilizadas e do próprio Manifesto Ágil.

3.1.6 SAF-ACS

O presente relato apresenta um projeto de aplicação Web gerenciado com a metodologia *Scrum*, desenvolvido por uma equipe de alunos e professores da Universidade Federal Rural de Pernambuco. É um relato acadêmico e relativamente pequeno, mas interessante pela motivação do uso de metodologias ágeis muito adequado aos princípios teóricos: a escolha de *Scrum* se deu pelo tamanho pequeno da equipe e pela natureza dinâmica e altamente mutável dos projetos Web.

O projeto chama-se SAF-ACS (Sistema de Acopanhamento Familiar do Agente Comunitário de Saúde) e tem por objetivo informatizar o processo de cadastro e acompanhamento familiar realizado pelo agente (BARROS et al, 2009). Contou com uma equipe de quatro pessoas (notar que a palavra *time* não é utilizada), cada uma encabeçando uma área: Requisitos, Codificação, Riscos e Testes.

Foram utilizados vários padrões de processo e artefatos de *Scrum*, algumas vezes de acordo com a teoria e outras com nomenclaturas determinadas pela equipe. São eles:

- Pendência: há referência explícita ao nome *product backlog*, sendo essa a tradução para o termo que o time decidiu utilizar. É explicado como sendo uma lista de requisitos priorizados pelo time. Não há citação de uso de Kanbam para organização, no entanto, por meio de figuras ilustrativas, pode-se ver que foi utilizado um software de apoio à gerência para a organização do *backlog*, o Chronos.
- *Sprints*: unidade de trabalho da equipe. Não é citado o tamanho da unidade de trabalho do time em específico.
- Reuniões *Scrum*: conforme o *paper* é uma tradução para *daily meetings*. Na realidade, o termo em inglês usado pela equipe é uma mistura de *daily scrum* com *daily meeting*, mas quer se referir à mesma prática de reuniões curtas, em pé, para conhecimento do andamento do trabalho por toda a equipe.
- Demos: termo criado pela equipe para o incremento de produto potencialmente entregável.

Não há descrição da forma como o trabalho foi tratado, mas se pode inferir por uma figura do trabalho (Figura 3.4), com um esquemático clássico de dinâmica de *Sprints*, similar ao visto no capítulo 2, que foi utilizada a forma teórica dentro de cada *Sprint*, com trabalho de análise, desenvolvimento e teste acontecendo em paralelo.

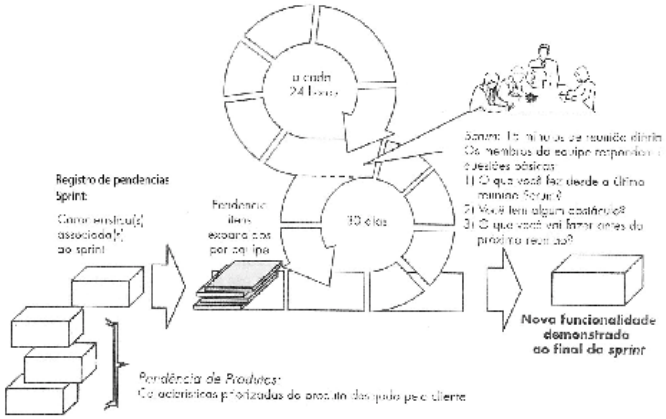

Figura 3.4: Diagrama de fluxo de *Scrum* (PRESSMAN apud BARROS et al., 2009)

A equipe explicita que qualquer metodologia ágil pode ser utilizada para desenvolvimento de aplicativos Web e apóia sua afirmação em Pressman (2007), que diz que um aplicativo Web deve ser entregue em incrementos, com cronogramas curtos e que suas modificações ocorrerão frequentemente.

3.1.7 Orbisat

Este relato é baseado no relato de Leandro Mesquista, em seu blog (MESQUITA, 2010), sendo que o autor prestou consultoria à empresa Orbisat no processo de implementação da metodologia *Scrum*. Esse é um relato informal, ainda que apresente todas as etapas da implementação do projeto.

Segundo Mesquita (ibdem.), o interesse surgiu de um dos departamentos de desenvolvimento da empresa, que tem por característica ser bastante dinâmico, mais especificamente na figura do gerente. Depois de leituras e pesquisas sobre o assunto, contrataram uma assessoria para auxiliar na implementação de *Scrum*.

Foi escolhido um projeto já em andamento, com prazos apertados. Como a adoção da metodologia partiu do gerente, não sendo uma decisão de toda a equipe, em um primeiro momento houve rejeição. Seriam mais coisas a entender, novos conceitos a assimilar, mas com o prazo para entrega do software já definido e pequeno.

Assim, o primeiro trabalho da consultoria foi entender o ambiente, para que a utilização do *Scrum* fosse proveitosa para todos, a empresa, a equipe e o cliente. Só então que foi iniciada a implantação da metodologia, com uma apresentação de introdução, onde foram abordados todos os passos que compõem o *framework Scrum*, juntamente com as práticas de estimativas, utilizando o *Planning Poker*, cerimônias, entregas com valor de negócio para o cliente, criação dos artefatos como o Kanban, *Burndown Chart*, relatórios, entre outros.

O passo seguinte foi a definição do *Product Owner*. O consultor seria o *Scrum Master*, num primeiro momento. Como o trabalho já estava em andamento, o *Product Backlog* já estava organizado, sendo que o foco seria então a organização do primeiro *Sprint*. Pode-se perceber, no entanto, que, exatamente por já haver uma análise prévia, o *Product Backlog* não estava organizado de forma teoricamente descrita no *Scrum*, com

histórias, e sim com casos de uso, o que, apesar de não estar escrito, implica em adaptação.

Houve uma reunião de *Sprint Planning* em que foram definidas características do projeto *Scrum*: tamanho do *Sprint*, *funcionalidade com valor de negócio a ser entregue* (com isso, verificamos que o time definiu um *Sprint Backlog*) e essas funcionalidades foram pontuadas utilizando *Planning Poker*.

Ao final da reunião, foram confeccionados o *Burndown Chart* e o Kanban do Projeto.

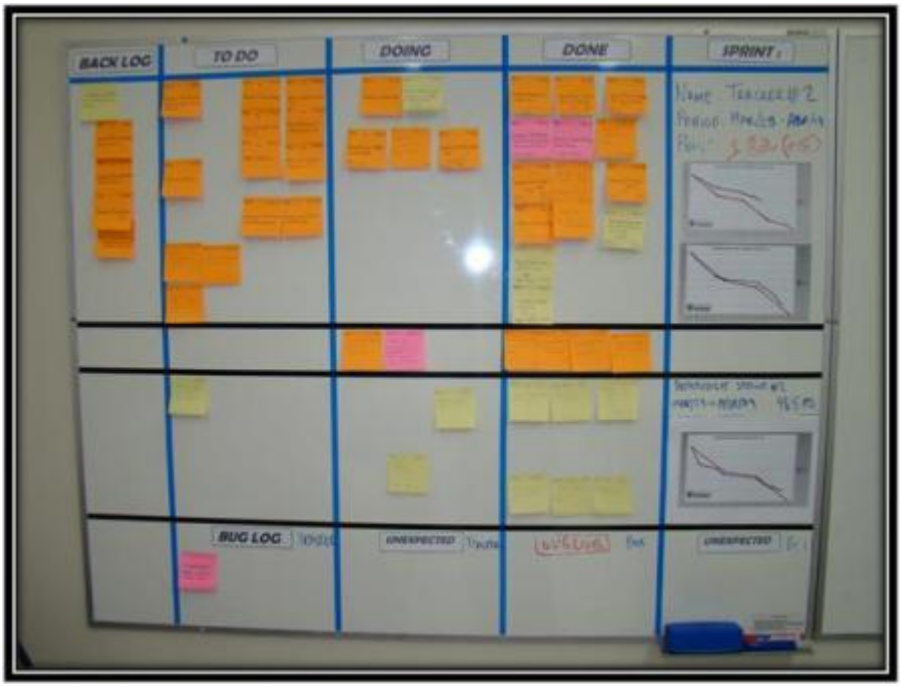

Figura 3.5: *Kanban* da *Orbisat* (MESQUITA, 2010)

A metodologia *Scrum* se adaptou muito bem ao projeto por que o ambiente era *caótico*, ou seja, instável, dinâmico, não linear. Havia a necessidade de *feedback* contínuo e com as atividades explicitadas para o *Product Owner* no Kanban isso se tornava mais claro, além dos problemas serem detectados com mais rapidez.

Na descrição do relato, não há nenhuma citação a boas práticas, bem como de implementação de outra metodologia ágil mais voltada para desenvolvimento, portanto, verfica-se que somente foi empregado *Scrum* para gerência, mas o desenvolvimento e testes continuaram a ser executado da forma tradicional.

3.1.8 Globo.com

O relato da Globo.com foi encontrado no blog pessoal de um dos membros da equipe, Guilherme Chapiewski (2008), sendo assim uma descrição não oficial da forma como metodologias ágeis foram implementadas na empresa.

Alguns desenvolvedores da Globo.com, antes da implementação formal de metodologias ágeis, já utilizavam boas práticas de *Extreme Programming*, tais como

Desenvolvimento Guiado por Testes (*Test-Driven Development*) e programação em pares (*Pair Programming*).

O ambiente de trabalho é altamente mutável, com cliente presente e muita demanda, que chegava sempre em lotes e com prioridade máxima. Os ciclos de planejamento, desenvolvimento e entrega eram irregulares e afetavam o desempenho de toda a equipe.

Dessa forma, em 2007, a empresa decidiu contratar um treinamento com o consultor Boris Gloger para alguns membros da equipe, para começar a mudança de metodologia (GLOGER, 2007). Pode-se verificar no site do consultor que seu foco é *Scrum*, com base em cursos para certificação oficial de *Scrum Master* e facilitadores.

A partir do treinamento, foi decidida a utilização de *Scrum* no time de desenvolvimento. A análise não foi feita utilizando metodologias ágeis, por que o projeto em que foi aplicada a metodologia, Globo Vídeos 4.2, já estava em andamento, utilizando metodologia *Waterfall*, e a fase de análise já se encontrava concluída.

Os profissionais que participaram do treinamento estavam começando a passar os conceitos para sua equipe, mas como não havia experiência prévia da maioria dos profissionais, foi decidido não usar os nomes *Scrum*, XP ou qualquer outro nome de novas metodologias ou práticas. O time foi introduzido à forma de trabalhar, mas sem a nomenclatura.

Foram realizados cinco *Sprints* no projeto Globo Vídeos, durante os quais a implementação da metodologia e o projeto sofreram adaptações, como a forma de criar histórias (*User Stories*) e como integrar a equipe de *Quality Assurance*. O projeto passou para produção, sendo colocado no ar em pouco mais de uma hora (contra 24 horas do projeto anterior, desenvolvido com *Waterfall*) e não houve relatos de *bugs* por mais de uma semana.

Houve outros projetos na Globo.com que foram desenvolvidos com *agile*, inclusive o site do *Big Brother Brasil*, que, segundo o autor do relato, foi desenvolvido utilizando *Scrum* de forma estrita, como está nos livros (COHN, 2004).

Posteriormente, a empresa ofereceu um treinamento de *Scrum* para pessoas de todas as áreas, e metodologias ágeis começaram a ser utilizadas de forma estrita. Houve uma curva de aprendizado, mas após alguns meses já era possível a observação de diferenças entre os times, que se adaptaram de formas diferentes, pois tratam de problemas diferentes.

Foi decidido pela equipe que os *Sprints* teriam duração de apenas duas semanas, e não mais de quatro, que consideraram muito tempo. Por esse motivo, a *Sprint Planning* teve duração de 4 horas, e não 8 horas. As outras reuniões tiveram a mesma duração. As estimativas foram feitas utilizando *Planning Poker* e as equipes são mistas, com desenvolvedores, programação *client-side*, um *tester* e um ROI, que faz o papel de *Product Owner*. Todos os membros da equipe sentem próximos uns dos outros, mas não há citação de que eles dividam espaços comuns, como a mesma mesa de trabalho.

O *Scrum* não fala de práticas de desenvolvimento, por isso foi utilizado XP, como já foi citado nas práticas *pair programming*, TDD e também integração contínua (*continuous integration*), *unit tests*, entre outras. O conceito de "done" é o que mais diverge entre as equipes. Nas equipes que possuem em profissional de *Quality*

Assurance, a homologação é feita dentro do *Sprint*. Um ponto de convergência é que em nenhuma equipe a tarefa de colocar o sistema no ar (nos sistemas online) é parte do *Sprint*, sendo que, quando o *Sprint* termina, é entregue um pacote fechado e pronto para ser colocado em funcionamento, então uma data e hora é agendada para que a subida seja feita acompanhada por um ou dois membros do time.

O consultor Boris Gloger foi chamado mais algumas vezes para acompanhar retrospectivas, dando conselhos e ajudando os times. As retrospectivas têm sido uma base forte para adaptações no processo e forma de trabalhar da Globo.com, pois mostram a evolução do time e a forma como este encontra e resolve problemas.

3.1.9 Agência AG2

Este relato é uma entrevista com Marcelo Bacchieri, responsável pela área de projetos da AG2 Agência de Inteligência Digital S.A., realizada pelo Blog Corporativo da LocaWeb (TORRES, 2008). Por serem apresentadas algumas características importantes de metodologias ágeis, é um material válido a título de comparação com outros relatos empresariais de implementação de agile em ambientes corporativos.

Esse material será complementado com a análise de campo na AG2 Publicis Modem de Porto Alegre, no subcapítulo 3.2.3, onde maiores detalhes de forma de trabalho (e explanações sobre projetos em específico) serão desenvolvidos.

A AG2 é uma agência que apresenta soluções digitais completas para grandes corporações, desenvolvendo sistemas e definindo opções estratégicas e tecnológicas para as mais diversas aplicações. Nesse contexto, a opção por *Scrum* se deu devido principalmente aos prazos apertados e o teor de alguns projetos. Testes começaram a ser feitos, inicialmente em tom investigativo, para conhecimento dos conceitos e posterior alteração do método.

Os times têm em média nove integrantes, um número que está de acordo com o que indica a teoria. Os times sentam lado a lado, na mesma mesa, agrupados por cliente. Os times são mistos, sendo que desenvolvedores, diretores de arte e outros profissionais sentam-se juntos, o que facilita a comunicação e a cultura do cliente é mais facilmente absorvida pelos profissionais.

A AG2 utiliza apenas *Scrum* em seus projetos. Durante a entrevista não foi citada nenhuma outra metodologia ou prática de outras metodologias. A fase inicial foi a de apresentação por meio de *workshops* e a aplicação dos princípios de *Scrum* em um projeto específico. No entanto, alguns projetos ainda utilizam o modelo tradicional de desenvolvimento, em cascata. Segundo Bacchieri, com escopo definido e altíssimo grau de complexidade, a AG2 opta pelo desenvolvimento tradicional, ao passo que quando se trabalha com uma *variável de tempo muito curta* e muitas *indefinições*, a metodologia ágil é a escolha por tender a aumentar as chances de sucesso.

A grande maioria dos clientes da AG2 são extremamente participativos, portanto, são envolvidos do início ao fim do projeto. Reuniões de definições de prioridades para desenvolvimento e aprovações acontecem durante o processo. Com isso, pode-se ver que, nos projetos desenvolvidos em Scrum, há sempre um *Project Owner* presente, e que são realizadas *Sprint Plannings* onde ele de fato desempenha seu papel de definir a prioridade das histórias a serem implementadas.

Como existem profissionais diferentes trabalhando na mesma equipe, nos primeiros *Sprints* tenta-se pragramar as funcionalidades que são independentes de interface, para nos *Sprints* subsequentes as áreas começarem a se aproximar.

A Agência AG2 pretende ainda aumentar a cultura em metodologias ágeis, realizando mais treinamentos e workshops para que todos em seu quadro de funcionários sintam-se confortáveis e confiantes em relação a elas.

3.2 Análise de Campo

A terceira parte de análise deu-se com pesquisa de campo em empresas gaúchas que utilizam metodologias ágeis. Foram visitadas por um dia empresas de diversos perfis para analisar a forma como utilizam as metodologias no seu dia-a-dia de trabalho. Além da observação simples, houve entrevistas com os gerentes ou diretores dos projetos, devidamente gravadas para registro posterior e registro fotográfico, o qual não pode ser utilizado em sua totalidade por questões de confidencialidade de projeto, portanto, algumas fotos não foram incluídas ou tiveram informações de natureza de projeto, irrelevantes para a análise do presente trabalho, borradas para a não identificação das mesmas.

As empresas visitadas foram a Woompa, *startup* porto-alegranse de desenvolvimento *web*; a *Endeeper*, empresa de pequeno porte, que desenvolve *softwares* para análise e gerencimento de informações petrográficas; e a AG2 Publicis Modem, agência de comunicação *webnative*, que atua com soluções web que incluem desenvolvimento de aparato *online*, como *sites*, *blogs* e *webgames*, entre outros.

3.2.1 Woompa

A Woompa é uma *startup* (empresa, geralmente recém-criada, em fase de desenvolvimento e pesquisa de mercados) porto-alegrense, que foca em desenvolvimento de produtos *Web*. Dois de seus projetos iniciais, *Cala Boca, Galvão* e *Polvo Vidente* (*hotsites* para a Copa do Mundo de 2010) utilizaram metodologias ágeis para seu desenvolvimento e em seu blog oficial existem postagens sobre assuntos diversos sobre *agile*, com foco em Scrum.

Atualmente, a Woompa está desenvolvendo um projeto para criação de sites focado no mercado de construção civil, o *Pixel Quadrado*, que se abrevia como *Px²*. Esse projeto aplicou metodologias ágeis em seus seis primeiros meses. Atualmente, por uma redução da equipe, que passou a ter somente dois integrantes, os métodos ágeis não estão mais sendo utilizados. No entanto, ainda são seguidos os princípios do Manifesto Ágil, tanto no decorrer do projeto como filosofia da empresa.

Flávio Steffens de Castro, criador da Woompa, teve seu primeiro contato com *Scrum* quando trabalhava em um grupo de pesquisa na Pontifície Universidade Católica do Rio Grande do Sul (PUC-RS), em 2006. Estava atuando como Gerente de Projetos e leu um artigo falando de métodos ágeis e começou seus estudos para aplicação de *Scrum*. Steffens é hoje um *CSM (Certified Scrum Master)*, embora afirme que a certificação não trouxe diferenças significativas em seu dia-a-dia na Woompa e em empresas onde atuou, e desenvolvou uma dinâmica para ensino de *Scrum*, chamada *Dinâmica da Fábrica de Aviões*.

Nos projetos em que *Scrum* estava sendo aplicado, diversas práticas da metodologia foram utilizadas. A análise se dava previamente e incrementalmente, pois os requisitos acabavam mudando com o tempo. Essa análise dava origem a histórias (com respectivas tarefas), organizadas em um *Product Backlog*. Para controle de velocidade, utilizava-se um *Burndown Chart*.

Cada *Sprint* era desenvolvido em 15 dias. Antes de cada *Sprint* era conduzida uma reunião de *Sprint Planning*, onde se definia o *Sprint Backlog* da próxima iteração. Diariamente, havia *Daily Meetings* para *status* do projeto.

A Figura 3.6 mostra o *Kanban* com as histórias em desenvolvimento, o *Burndown Chart* e o *Sprint Backlog* do *Sprint* corrente.

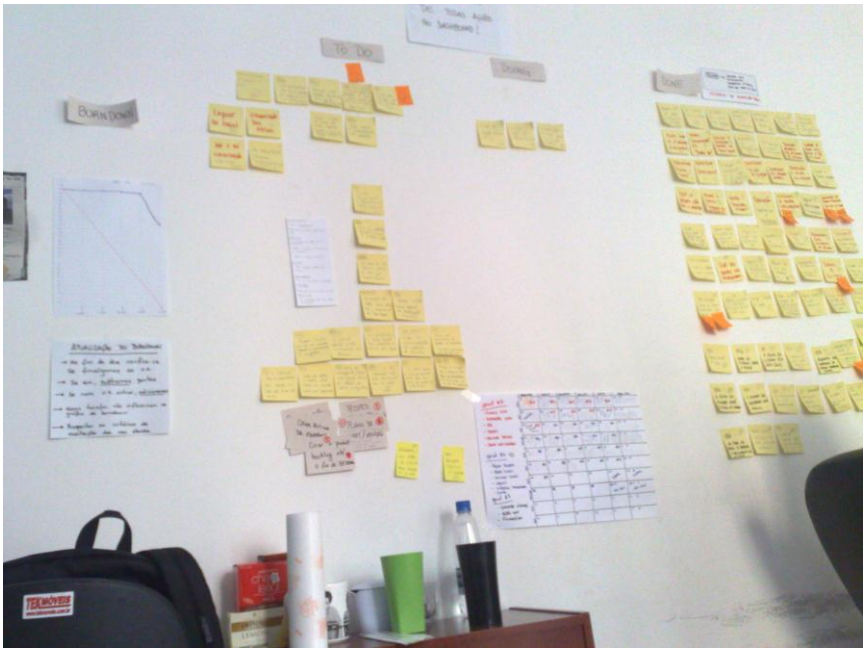

Figura 3.6: Kanban, Burndown Chart e Product Backlog

Algumas práticas de *Extreme Programming* também são utilizadas informalmente, como o *time coeso*, os *testes de aceitação*, *propriedade coletiva* e *emana de 40 horas*. Apesar do foco da empresa ser *Scrum*, essas práticas de *XP* têm muito dos princípios ágeis.

Não eram utilizados *incrementos de produto potencialmente entregáveis* por que os projetos não envolviam entregas para clientes (os *hotsites* foram projetos pequenos e o *Px²* é um produto da própria Woompa, que ainda não foi lançado).

Ao final de cada *Sprint*, eram realizadas reuniões de retrospectiva, muitas vezes em ambientes informais, fora da área de trabalho. Nas retrospectivas eram reunidos os pontos fortes, fracos e a serem mantidos da *Sprint* que passou, descritos em *post-its* e agrupados (Figura 3.7), de forma a servirem de referência para os *Sprints* seguintes.

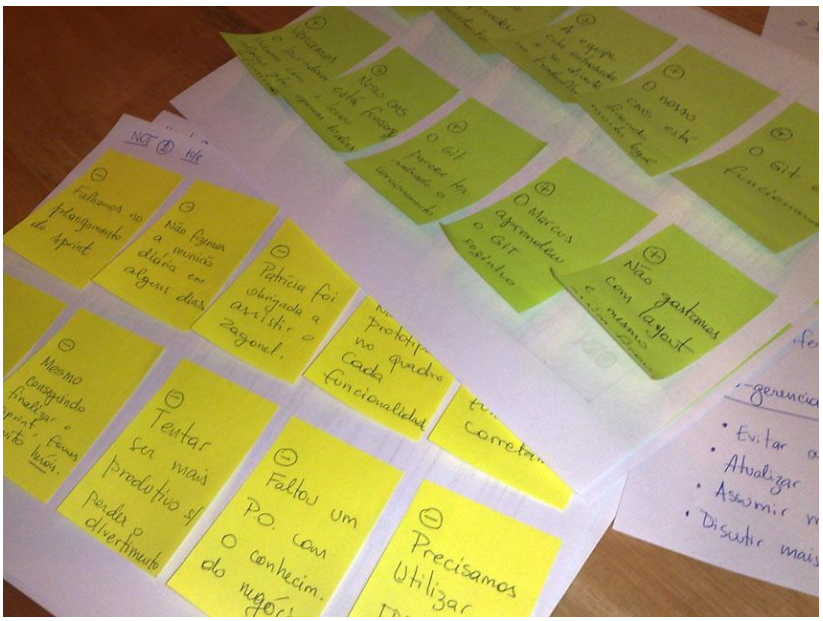

Figura 3.7: artefatos da Retrospectiva de *Sprint*

Como os dois primeiros projetos desenvolvidos eram pequenos e autorais, não houve a necessidade de um *Product Owner*, sendo que as decisões de projeto eram da própria equipe, muitas vezes rascunhadas em um quadro branco (Figura 3.8) e discutidas por todos. Nesses projetos não houve criação de histórias formalmente, por serem pequenos, mas houve rascunho de definições no quadro, criando um tipo de documentação ágil e colaborativa, de acordo com o Manifesto Ágil. No *Px²*, mesmo com a criação de histórias, o quadro branco continuava a ser utilizado com ferramenta de documentação ágil e troca de idéias para o projeto.

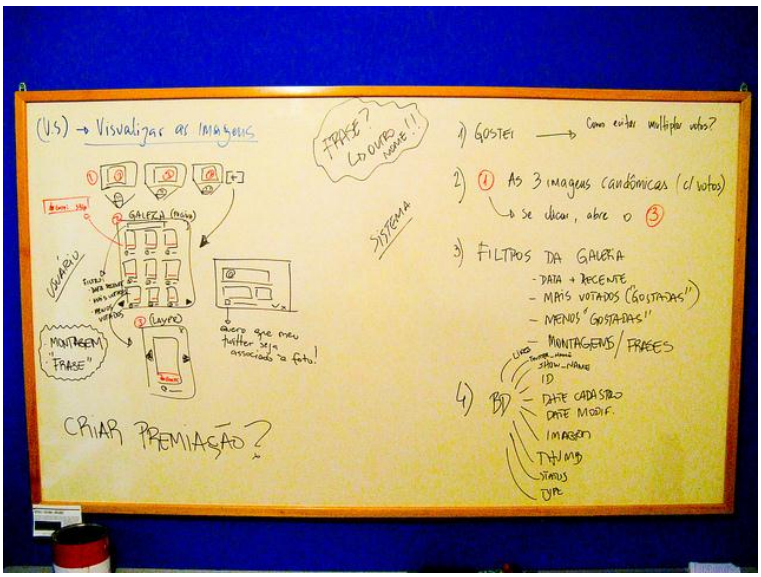

Figura 3.8: Quadro Branco Colaborativo

Além do quadro branco, outros tipos de documentação ágil foram utilizados. O *Google Docs* foi utilizado, com acesso de toda a equipe, como numa *wiki*, e foi

desenvolvido um *pseudo-UML* para algumas partes do sistema do *Px²*. Os softwares de apoio ao trabalho foram o a já citada suíte de escritório *online Google Docs*, o sistema de versionamento Git (http://git-scm.com/) e o disco virtual *Dropbox* (http://www.dropbox.com/) para armazenamento de código e outras informações relevantes.

No projeto *Px²*, por ser um produto a ser lançado pela Woompa e não um projeto encomendado por um cliente, pode-se considerar que não existe um *Product Owner*. Porém, por ser um produto idealizado por Steffens, pode-se considerar que ele acumule o papel de *Product Owner*, além de *Scrum Master*. De qualquer forma, não é uma prática em acordo com a teoria, pois a situação da empresa e do projeto são diferentes das abordadas na blibliografia básica de *Scrum.*

Mesmo quando *Scrum* estava sendo aplicado, o time não focava em utilizar nomenclaturas específicas. Desde que as práticas e princípios sejam aplicados e entendidos, não há um esforço para uso de acordo com a teoria. No entanto, talvez pelo fato do *Scrum Master* ter muitos anos de estudo e ser certificado, é notável que o uso das nomenclaturas de *Scrum* seja natural no ambiente de trabalho.

O *Scrum* parou de ser aplicado na Woompa quando o time foi reduzido a somente duas pessoas, de forma que não havia mais sentido para a empresa ter uma metodologia formal. Os prazos para o lançamento do *Px²* começaram a se esgotar com a diminiução da equipe, de modo que o custo-benefício de utilizar metodologias ágeis não estava mais sendo vantajoso para o time. Pretendem voltar a utilizar *Scrum* assim que a equipe aumentar.

Os planos futuros da empresa envolvem uma expansão, mas focando em uma equipe de no máximo 15 pessoas, multidisciplinar. O foco será na troca de ideias e passagem de conhecimento e, para isso, a equipe deve estar toda na mesma sala, o que segue os princípios tanto de *Scrum* como de *XP*, esse último tendo inclusive o princípio do *Whole Team* especificando que a equipe esteja geograficamente reunida.

3.2.2 Endeeper

A Endeeper é uma empresa porto-alegrense, provedora de software e serviços para gestão do conhecimento e integração de dados, atuando principalmente no setor de petróleo e gás. Desenvolve uma gama completa de software para avaliação de reservatórios com base em características petrológicas das rochas. Desde 2007, a Endeeper encontra-se no catálogo de fornecedores aprovados pela Petrobras.

A Endeeper teve seu *paper* analisado no capítulo 3.1.2 desse trabalho. Nesse capítulo será descrito e analisado o relato de observação presencial da empresa, que se deu por meio de uma visita onde pode ser observado o ambiente de trabalho e uma reunião de *Sprint Planning* de um produto em desenvolvimento. Por motivos de confidencialidade, nome do projeto e características do produto a ser lançado não serão citadas, bem como tecnologias que não tenham impacto nas metodologias de trabalho utilizadas pelo time.

A empresa utiliza *Scrum* combinada com algumas práticas de *XP*. Carlos Santin, *Project Manager* da Endeeper e *Scrum Master* do projeto, explica que os conhecimentos do time em metodologias ágeis foram sendo adquiridos com o tempo e

prática, sendo que não foi feito nenhum curso formal para adquirir conhecimento. Dessa forma, as metodologias são usadas de forma bastante livre e adaptada à realidade da empresa.

A análise é feita preliminarmente e é utilizado um software livre de gerenciamento de projetos para o planejamento de cronograma e valor do projeto. No entanto, apesar de existir um cronograma, a análise não para durante o desenvolvimento. Ela é incremental e a cada *Sprint* são adicionadas novas funcionalidades, que são modeladas em forma de histórias.

A Endeeper possui uma parceria com o Instituto de Geologia da UFRGS, que é o *Product Owner* do projeto. Eles estão relativamente disponíveis, participando de algumas reuniões para validação e aplicando testes de usuário (prática de *Extreme Programming*). No entanto, não há um representante que permaneça junto com a equipe durante todo o período de trabalho.

O time é pequeno, possui somente três desenvolvedores (incluindo Santin, que é desenvolvedor e *Scrum Master* do projeto) e uma representante da Geologia que faz algumas atividades de análise, mas que não senta no mesmo ambiente e não é dedicada exclusivamente ao projeto. O time de desenvolvimento realiza *Daily Meetings* para o acompanhamento do *status* de cada membro e impedimentos.

Nomenclaturas tratadas na teoria não são uma prioridade para o time, que utiliza traduções livres para referir-se a artefatos de *Scrum*. Por exemplo, o *Task Board* é chamado simplesmente de Quadro. Ele está organizado pelas histórias em desenvolvimento, com suas respectivas tarefas, e são essas tarefas que mudam de *status* no período de desenvolvimento. Para o controle de velocidade, utiliza-se um *Burndown Chart*, referido pelo time pela nomenclatura correta, que mostra as tarefas realizadas pelo time no *Sprint*.

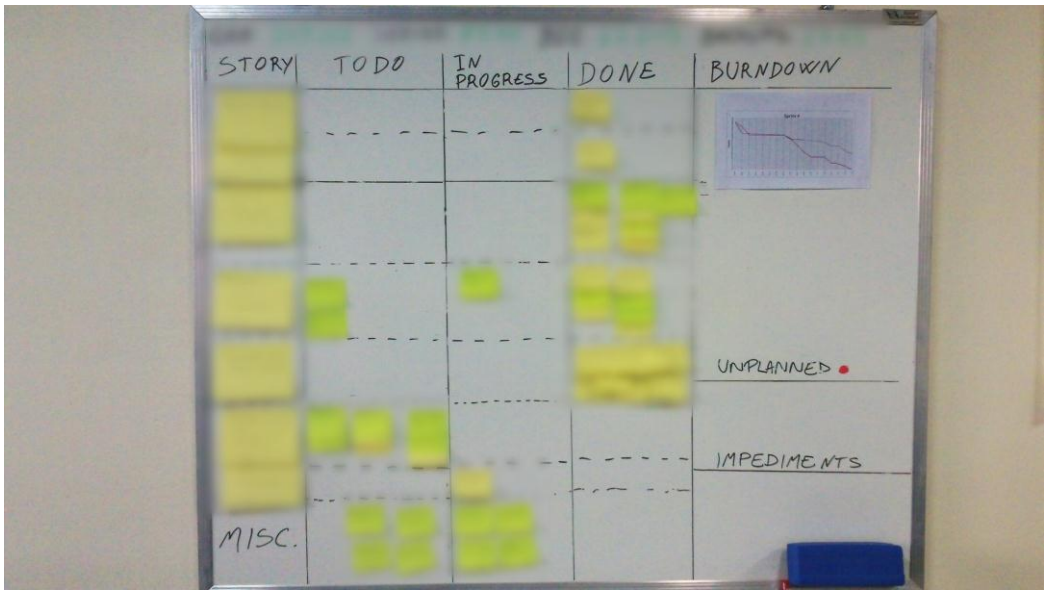

Figura 3.9: Quadro (*Task Board*) e *Burndown Chart* da Endeeper

Testes unitários começaram a ser aplicados a pouco pelo time. Além disso, atualmente, há uma empresa contratada para realização de testes de aceitação, testes de usuário, carga e performance. Ou seja, os testes não são integrados ao ciclo do *Scrum.* O *done* da tarefa e da história é dado quando o desenvolvimento foi concluído.

Anteriormente, mais práticas de *Extreme Programming* eram aplicadas. Todas as sextas-feiras, os desenvolvedores *pareavam* para fazer a refatoração do código (notar que a prática não era usada no desenvolvimento). *Acceptance criteria*, de *Scrum*, também era uma prática utilizada a partir de relatórios de teste, com *checkpoints*, que eram aplicados após um novo desenvolvimento e antes da liberação do produto final. Com a falta de tempo, essas práticas deixaram de ser aplicadas no projeto em andamento, porém a empresa tem o objetivo de retomá-las assim que novos integrantes forem adicionados ao time.

Os códigos são feitos da forma mais genérica possível, para possibilitar o reaproveitamento para outros projetos. Portanto, identifica-se que a simplicidade de *XP* não é aplicada. Mas o time aplica *Coding Standarts*, Refatoração, inclusive utilizando o livro básico de Martin Fowler como base de estudo (FOWLER, 2004), Semana de 40 horas.

Alguns documentos são desenvolvidos. Além do já citado documento preliminar de análise, também são utilizados alguns diagramas para modelagem do sistema e manuais de utilização, que são tratados como uma tarefa a ser desenvolvida.

Não há responsáveis por determinados códigos do sistema, o que significa que é aplicada a prática de *Propriedade Coletiva do Código* de XP. Existem alguns membros do time que são mais familiares a algumas áreas do código que outros, mas o time tenta fazer com que todos tenham conhecimentos (mesmo que superficiais) em todas as áreas.

Na pesquisa de campo pode-se acompanhar uma reunião de *Sprint Planning*, que não tem uma duração fixa de tempo, em desacordo com o que diz a teoria. Essa reunião mistura um pouco das dinâmicas de uma Retrospectiva, que não é feita pelo time separadamente, pois inicia com uma conversa sobre o que houve na *Sprint* anterior, positiva e negativamente. Utilizando o programa de gerenciamento, as tarefas são editadas e adicionadas durante a reunião.

As novas histórias, com suas respectivas tarefas, são criadas e inseridas no *Product Backlog* (que o time chama de lista de requisitos), tendo seu âmbito e complexidade discutidos durante o *Sprint Planning*. As estimativas são feitas em horas, baseadas na experiência dos desenvolvedores. As histórias muitas vezes são grandes, do tamanho de *Epics*, e tem duração de várias *Sprints*. Os desenvolvedores se comprometem a terminar as tarefas, e não necessariamente a história.

As tarefas não concluídas voltam para a lista de requisitos, assim como os *bugs* encontrados, que são transformados em tarefas. Não necessariamente precisam entrar na *Sprint* seguinte. É discutido o tempo de aprendizado de novas tecnologias necessárias ao projeto, sendo que esse tempo entra na estimativa da tarefa, que tem seus tempos estimados sempre arredondados para cima. As estimativas sempre se referem a tarefas, nunca a histórias.

O *Sprints* não tem duração fixa, mudando sua duração de um para o outro, mas mantendo-se entre 2 e 4 semanas, com média de 20 dias, o que está em desacordo com a teoria. Cada desenvolvedor diz sua estimativa e algumas tarefas são a ele atribuídas, formando assim o *Sprint Backlog* (o time não utiliza essa nomenclatura). Com isso, somam-se as estimativas de horas de cada tarefa com as quais os desenvolvedores se comprometeram e desse cálculo obtêm-se a duração da *Sprint*.

Na *Sprint Planning* acompanhada, o *Sprint* terá duração de 15 dias. As histórias e tarefas adicionadas são atualizadas no documento de gerenciamento, escritas em *post-its* e colocadas no Quadro, com *status* de *To-Do*. Com isso, o *Burndown* para a *Sprint* é gerado e as atividades de desenvolvimento têm início.

3.2.3 AG2 Publicis Modem

A AG2 Publicis Modem é uma agência de comunicação *webnative* do Grupo Publicis. No Brasil, já possuem mais de uma década de atuação, com cerca de 200 colaboradores em São Paulo e no Rio Grande do Sul. Atua com comunicação multicanal a partir de plataformas e ambientes de interação para potencializar as estratégias de marketing e vendas das organizações.

A AG2 teve sua aplicação de *Scrum* analisada no capítulo 3.1.9, para as equipes de Marcelo Bacchieri. Neste capítulo analisar-se-á a aplicação de *Scrum* para equipes maiores do que as tratadas por Bacchieri em sua entrevista, a partir de uma visita para observação na empresa, na sede de Porto Alegre.

O projeto observado neste trabalho tem um tempo de desenvolvimento total de 2 anos, aproximadamente 60 mil horas de trabalho e uma equipe de 45 pessoas. Dessa forma, a equipe está distribuída por função e também geograficamente em quatro times. Em Porto Alegre está o time de Arquitetura de Informação (AI), em Pelotas estão os times de Codificação e *Design* e em São Paulo está o time de Redação.

Alexandra Ibaldo, Gerente de Projetos (GP) da AG2, *Scrum Master* e *Product Owner* do projeto, explica que a intenção inicial era que cada time tivesse seu próprio *Scrum Master* e que sua função fosse apenas de Gerente. No entanto, por ser a única gerente e pelo nível da equipe, que é formada essencialmente por profissionais-júnior, houve a opção de que ela acumulasse outros papéis para um melhor funcionamento do projeto. Em breve, a equipe contará com outro Gerente de Projeto, atuando em São Paulo, que terá papel de *Product Owner*, e Alexandra Ibaldo terá seu trabalho com foco maior em gerência.

São utilizadas práticas de *Scrum* para o desenvolvimento do trabalho. *Extreme Programming* não é aplicável, pois os times não são focados somente em desenvolvimento (apenas o time de codificação). Algumas práticas até podem ser identificadas, como os Testes de Aceitação (que são feitos com o cliente) ou a Propriedade Coletiva, mas elas não estão sendo aplicadas com a intenção de seguir práticas de *XP* e, portanto, não serão analisadas.

Para a análise preliminar é utilizado o *Microsoft Project*, juntamente com práticas de PMI para plano de projeto, análise de riscos, entre outros. Isso é necessário pela complexidade do projeto e pelo cliente, que exige essa documentação inicial.

Não são utilizadas histórias. Foram utilizadas somente no processo de arquitetura, mas não são utilizadas no dia-a-dia dos times. Não há *backlogs* e os requerimentos são passados para os times pela demanda. Existem alguns documentos no *Google Docs*, como planilhas de controle, que podem ser acessados e editados por todos para informações do projeto.

As estimativas são feitas em horas e a velocidade é medida pela experiência. Se o time costuma entregar duas tarefas em uma semana de trabalho e entregam em menos tempo, a velocidade aumentou. Não há *burndown* ou *burnup charts* e nenhuma definição mais formal de velocidade. O volume de trabalho é adaptado da análise inicial feita no *Microsoft Project* para que não haja atrasos para o cliente. Existem metas semanais, mas não são rígidas, levando em consideração a curva de aprendizado para a tarefa a ser executada.

Existe um *Kanban* para integração das equipes e outro utilizado somente pela equipe de Arquitetura de Informação. Nesse *Kanban* são mostrados os impedimentos e tarefas (descrições simples do que deve ser realizado) que estão atreladas a cada membro.

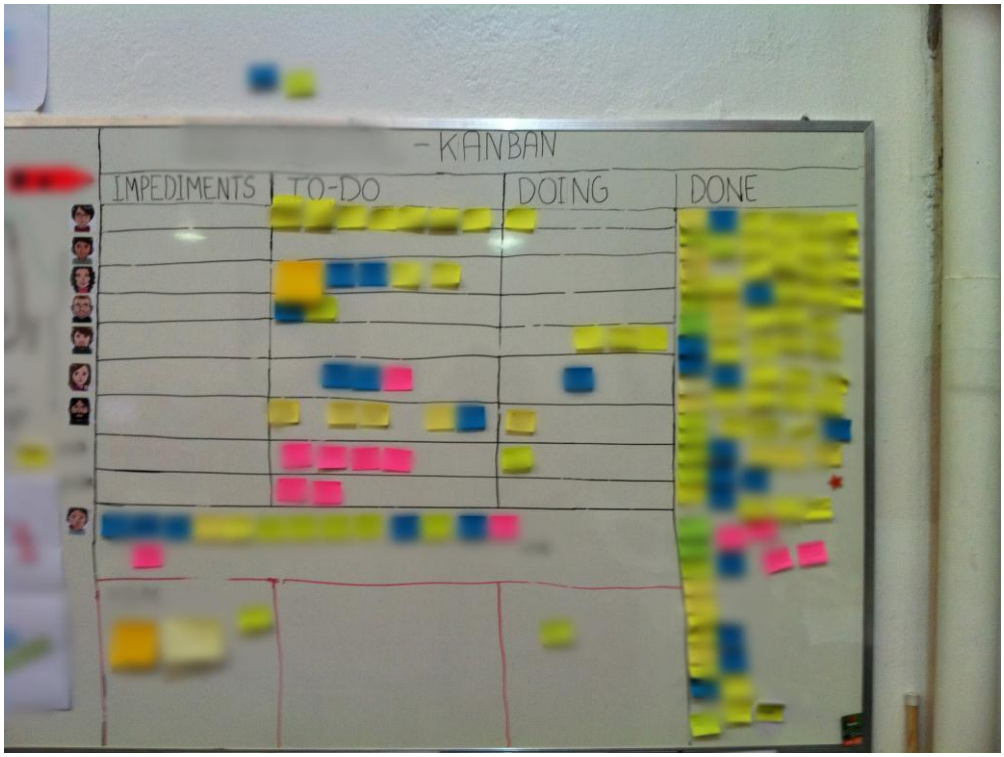

Figura 3.10: *Kanban* do Time de Arquitetura de Informação

Cada time se auto-organiza e realiza suas *Daily Meetings*. Até pela questão geográfica, nem sempre a *Scrum Master* está presente, sendo que os impedimentos são passados posteriormente para ela.

Os produtos potencialmente entregáveis vão sendo construídos pelas equipes e sendo passados para validação do cliente, sem entrar em produção. Há também entregas internas de “padrões” de um time para o outro.

Os *Sprints* têm duração de uma semana. Todas as sextas-feiras é realizada uma reunião de retrospectiva, *status* e problemas, onde todos os times participam, por vídeoconferência. Nessa reunião, são vistos os impedimentos, inclusive dos times remotos.

Algumas vezes, por problemas do projeto, a reunião acaba por focar mais em anúncios e resoluções. No entanto, ao final da reunião, há um tempo reservado para a Retrospectiva em si, onde os membros do time explicitam os pontos positivos e negativos da *Sprint* passada. Essa reunião não usa artefatos (quadro, *post-its*) e não obedece um cronograma rígido, sendo uma conversa entre os times e a *Scrum Master*.

O processo de *Quality Assurance* é feito concomitantemente com os *Sprints*, tanto por profissionais dos times (em especial de Arquitetura de Informação e *Design*), quanto por um profissional de fora dos times, do cliente. Esse profissional executa testes de aceitação e repassa os resultados ao time, que reintegra as modificações a serem feitas nas próximas *Sprints.*

O conceito de *Product Owner* é aplicado da melhor forma encontrada pela equipe, com sua Gerente de Projeto indo ao cliente (em São Paulo) para reuniões periódicas e levando para os times o *feedback* sobre o projeto. Não é a forma ideal designada pela metodologia, mas não está em discordância, pois quando não há no cliente alguém que possa designar o valor de negócio do projeto, essa função passa a ser do *Scrum Master*.

O cliente é participativo, mas nem sempre está disponível, sendo que algumas vezes a comunicação é difícil. No entanto, entre a equipe a comunicação é encorajada, sendo dito que não se deve deixar de realizar as reuniões (diárias ou semanais) para o cumprimento de prazos, o que está em concordância com os princípios ágeis.

O ambiente de trabalho da equipe de Arquitetura de Informação é bastante de acordo com as metodologias ágeis: todos sentam juntos, o que facilita a troca de informações e a proximidade do time. Mas pela característica dos times de separação por função e geográfica, algumas vezes a comunicação entre times é prejudicada.

Figura 3.11: ambiente de trabalho da equipe de AI

O time utiliza as nomenclaturas com muito apuro. Todos os membros, mesmo os de áreas não técnicas, sabem os significados dos termos usados na teoria de *Scrum* e referem-se aos artefatos e reuniões utilizando-os, sem criar nenhuma nomenclatura específica do time.

O relato da AG2 é interessante por que integra *Scrum* de forma distribuída com equipes com focos diferentes e, dessa forma, permite que elas se comuniquem com maior clareza. O fato de a equipe ser grande estaria em desacordo com a teoria, mas tendo sido quebrada em quatro equipes menores, está em concordância com o que é dito em *Scrum.* A metodologia é aplicada com bastante liberdade, pois não se trata de uma fábrica de software, mas dentro do contexto possível, é aplicada com bastante acuidade.

3.3 Considerações

Os relatos analisados apresentam particularidades que refletem a forma de trabalho de cada empresa e de sua equipe. No caso dos relatos descritos em *papers* e artigos, refletem também a visão dos autores sobre a experiência vivida em um ambiente de metodologias ágeis, assim como os relatos de observação acabam por trazer a visão da autora sobre a forma e os conceitos das metodologias ali aplicadas.

Pode-se notar que as aplicações são variadas e que, de uma forma ou de outra, os relatos apresentam motivações para o uso de metodologias e princípios ágeis. O ponto contrário, de situações onde metodologias ágeis não são indicadas, será discutido no próximo capítulo. A sumarização e conclusões sobre tão diferentes jeitos de aplicar as mesmas metodologias e príncípios será analisada no capítulo 5.

4 O OLHAR INVERSO: QUANDO AGILE NÃO É SOLUÇÃO

As metodologias ágeis são bastante amplas e adaptáveis a diversos cenários. Inclusive, pode-se observar pelos relatos analisados no presente trabalho que nem sempre elas são utilizadas da maneira originalmente descrita pelos autores, sendo customizadas pelas equipes de acordo com a realidade e demanda de cada projeto.

No entanto, as metodologias ágeis são somente uma opção dentre as tantas formas de se desenvolver software (somente nesse trabalho várias outras já foram citadas, dentre elas CMMI, cascata, espiral). E nem sempre são a solução adequada para as características do projeto, empresa ou etapa de desenvolvimento.

O livro básico de *Extreme Programming* (BECK, 1999) dedica um capítulo a situações onde a metodologia não é indicada. A primeira e principal barreira apontada no livro é a cultura, em especial a cultura empresarial. Se a empresa tem a cultura de tratar dos projetos linearmente, traçando um objetivo e apontando para ele desde o início com o trajeto todo traçado, XP criará um conflito, pois o trajeto para chegar ao objetivo é traçado e reestruturado continuamente. A cultura também se manifesta na exigência de análise preliminar do problema, acompanhada de uma documentação extensiva. Os clientes ou gerentes terão de se acostumar com diálogos no lugar de documentos, o que requer engajamento contínuo, algo muitas vezes difícil para gerentes de diversos projetos sobrecarregados com compromissos. Empresas tradicionais (o autor cita bancos como exemplo) podem exigir documentação simplesmente por hábito ou parte da cultura organizacional, sendo um desafio para a equipe mostrar que os códigos e testes já devem ser inteligíveis e autoexplicativos.

Extreme Programming exige que a equipe trabalhe em sua velocidade máxima, portanto, exigência de horas extras é contraprodutivo para ambientes XP, pois deixa os desenvolvedores cansados e, com isso, menos produtivos em suas horas regulares de trabalho.

O tamanho da equipe é outro fator a ser considerado. XP funciona bem com equipes pequenas, especialmente de até 10 pessoas. Com 20 pessoas, a equipe já passa a ser difícil de gerenciar dentro do âmbito de Extreme Programming. Com isso, projetos muito grandes não são ideais para as práticas de XP, embora a quantidade de funcionalidade a ser produzida e o número de pessoas produzindo-as não seja uma relação linear e deve ser analisada para cada projeto. Em *Scrum*, tem-se uma relação ainda mais exata: o número ideal de pessoas em um time é sete, podendo variar em dois para mais ou para menos. No entanto, encontra-se no mercado casos de equipes maiores

utilizando metodologias ágeis. Nesses casos, para melhor gerência, deve-se "quebrar" o time em subprojetos que possam ser tratados individualmente.

Uma das situações difíceis de utilizar *Extreme Programming* é quando a curva de custo do projeto seja inerentemente exponencial, ou seja, quando o projeto será utilizado por muitos anos, recebendo atualizações periódicas ou baseado em diversos sitemas legados. Isso impede que o código seja mantido limpo e simples, pois há uma tendência a não mexer no código existente e, com isso, fazer o código atual ser extremamente complicado. Isso também faz com que seja desaconselhado (embora não impossível) utilizar XP por desenvolvedores terceirizados (*outsourcing).* Ao fim do projeto, pode-se ter uma pilha de código que o cliente não sabe como manter e poderá, posteriormente, ser passada para outra equipe. A terceirização também dificulta que o cliente esteja presente no projeto, o que vai contra não somente as práticas de XP, mas de outras metodologias ágeis, como Scrum.

Longo tempo para obter *feedbacks* é um impeditivo apontado para a aplicação de *Extreme Programming*, mas claramente vai contra a filosofia de metodologias ágeis como um todo. No contexto *Scrum*, é extremamente complicado avançar diariamente com um sistema que precisa de muitas horas para compilar ou *linkar*. Isso já alimina artefatos importantes da técnica, como as *Daily Meetings*, visto que a equipe poderia estar constantemente bloqueada por grandes períodos de tempo.

Basicamente, essas características podem ser aplicadas a quaisquer projetos e são extensíveis a grande maioria das metodologias ágeis, pois tratam de situações onde a mudança é impossível ou muito morosa, de forma que não é sustentável aplicar um método que precise de grande velocidade e exija mudanças relativamente grande nesse tipo de situação ou ambiente.

Uma situação impeditiva para as técnicas de XP é que elas devem ser aplicadas com toda a equipe presente no mesmo ambiente. Dois andares de um mesmo prédio já impossibilitam a aplicação da técnica – embora essa regra não se aplique a *Scrum* estritamente, como visto nos relatos tratados nesse trabalho.

O relato do Instituto Atlântico trata de um caso em que a empresa, por exigências organizacionais, precisava manter um histórico de *Use Cases*, com seus respectivos *Use Case Points* e complexidades. O *paper* explana todas as etapas para a integração entre os *Story Points* para a utilização com Scrum e os *Use Case Points*. Houve sucesso para a empresa, porém, deve-se ser levado em conta o tempo necessário da equipe como um todo para as estimativas nos dois métodos e, posteriormente, para manter a documentação necessária. Numa situação como essa, deve ser pesado se o incremento de tempo de análise (algo que já vai contra a filosofia das metodologias ágeis, que considera análise parte do trabalho incrementalmente, junto com desenvolvimento e teste) vale a pena para uma mudança que se dará somente no projeto, não na cultura organizacional como um todo.

Como citado na própria análise do *paper*, pode ser um passo inicial para mudança de método como um todo, e para projetos em específico o custo-benefício de adotar a metolodologia ágil, mesmo que combinada com mais processos e documentos exigidos pela organização, pode ser de valia frente às características que se apresentam. De qualquer forma, não se deve esquecer que, por não mudar a cultura da empresa quanto à

necessidade de documentação, histórico e análise, ela sempre será exigida, de forma que esse incremento de tempo e trabalho deve ser considerado para todos os projetos em que metodologias ágeis forem utilizadas. Dessa forma, há ser pesado o custo-benefício da metodologia ágil frente a outros aspectos do projeto para escolher a melhor forma de desenvolvê-lo.

O tamanho da equipe conta também quando existem equipes muito pequenas. No relato de observação da Endeeper, percebeu-se que práticas de *Scrum* e *Extreme Programming* funcionam para times com três integrantes, considerando ainda que existe uma quarta integrante que atua de modo não fixo no time. Na Woompa, quando o time contava com três membros, o *Scrum* era aplicado, mas quando o time foi reduzido a uma dupla, sendo que os membros acumulavam diversos papéis, foi verificado que não havia mais sentido em algumas práticas da metodologia e acabava-se perdendo mais tempo sendo formal com o processo. O custo-benefício da aplicação de *Scrum* conforme a teoria já não estava sendo vantajoso e, portanto, o time decidiu por abandonar *Scrum* e focar somente em princípios ágeis, sem a utilização de nenhuma metodologia em específico.

5 ANÁLISE E RESULTADOS

Cada empresa desenvolve, ao longo do tempo, seu método de trabalho. Seja qual for a metodologia aplicada, ela será adaptada à realidade da equipe que a está aplicando. Em um mundo de possibilidades tão amplo e livre como o de metodologias ágeis, é esperado que as interpretações sobre a teoria sejam as mais diversas. No entanto, existem alguns pontos em comum que são aplicados com maior frequência, e outros que estão deixando de ser usados pela maioria das equipes, formando assim um cenário que uniformiza o que é chamado de metodologias ágeis.

Pode-se dizer que, atualmente, no contexto do mercado de trabalho brasileiro, usar metodologias ágeis é praticamente um sinônimo de usar *Scrum*. Todos os relatos analisados utilizavam *Scrum* como metodologia principal, e esse foi o motivo dela ser analisada com maior aprofundamento no presente trabalho. Em certos casos, *Scrum* é combinado com *Extreme Programming*, sendo que algumas vezes isso é feito formalmente, com a equipe dizendo que utiliza a metodologia e estudando as práticas para aplicá-las (como no relato *Sistema de Gestão de Ensino*), e outras informalmente, quando a equipe aplica algumas práticas isoladas de XP, sem se preocupar com seguir a metodologia em si (como no relato de observação da Endeeper, por exemplo).

É notável que as metodologias ágeis, mesmo *Scrum*, que é uma metodologia de gerenciamento, sejam aplicadas primordialmente por equipes de desenvolvimento. Há exceções, como no caso da AG2 (tanto o relato de observação como no descrito na entrevista para o *blog* da Locaweb), porém, em sua maioria, os processos de análise e testes não são integrados nas *Sprints*. Com isso, o conceito de "*done*" da história ou tarefa, principalmente quando é utilizado o *Task Board*, acaba por ser quando o desenvolvimento acaba e a história ou tarefa é passada para a equipe de testes (ou para o incremento de produto potencialmente entregável, quando não há uma etapa de testes).

Outra consequência de *Scrum* ser aplicado principalmente por programadores é que, em alguns casos, como nos relatos *Software House* e *Globo.com*, a análise de requisitos seja feita preliminarmente, sem seguir a natureza incremental das metodologias ágeis. Especialmente em projetos que já estavam em andamento, como o do relato *Globo.com* ou quando há exigências burocráticas da empresa, como no *Instituto Atlântico*, os times acabam por utilizar *Use Cases* como seus requisitos, ou convertê-los para histórias e posteriormente realizar a integração.

Mesmo com o uso de *Use Cases* em alguns casos, histórias são largamente utilizadas por equipes que trabalham com metodologias ágeis. Com exceção da AG2, que não trabalha somente com desenvolvimento e, por isso, considera tarefas e não histórias, e

do *paper* da Endeeper, onde o âmbito do projeto era a implementação de CommomKADS e não o desenvolvimento de *software*, todos os outros relatos usam histórias ou algum mecanismo muito próximo, tratado por outro nome (*Ticket Tracking* para o *Telessaúde* e *Work Itens* no *Sistema de Gestão de Ensino*).

As histórias costumam ter estimativas associadas. Nem sempre são utilizados *Story Points*, as estimativas podem ser em horas ou simplesmente pela experiência da equipe. A experiência também pode ser utilizada como medida de velocidade, mas essa métrica é mais comumente acompanhada pelo gráfico de *Burndown* (citado por *Telessaúde*, *Orbisat*, *Endeeper* e *Woompa*).

Apesar de não estar em total harmonia com a teoria, *Scrum* distribuído é bastante utilizado e tem resultados satisfatórios (por exemplo, o *Telessaúde* foi concluído com sucesso e seus integrantes encontraram formas de se comunicar, mesmo com a distância). Artefatos físicos, como *Kanbam* e *Burndown chart* migram para versões *online* (ou são utilizados pelos times distribuídos individualmente, como no caso da AG2) e deve-se ter atenção redobrada para a comunicação não sofrer muito com a natureza distribuída do projeto. Nenhum dos projetos que utiliza *Scrum* distribuído o combina com *Extreme Programming*, pois XP diz explicitamente que todos os membros do time devem sentar na mesma sala, não sendo somente uma indicação, até por que algumas práticas, como *Pair Programming*, são mais difíceis de ser aplicadas de maneira distribuida (não presencial), demandando ferramentas precisas para comunicação e coordenação de horários e atividades mais complexa do que todos os profissionais encontram-se no mesmo ambiente.

Também não é comum realizar reuniões de final de *Sprint*, *Review Meeting*, apesar de "Retrospectiva" ser utilizada em alguns casos. Por exemplo, no relato da Woompa há fotos dos artefatos utilizados na Retrospectiva, e essa reunião foi acompanhada presencialmente no estudo da AG2. Já as *Review Meetings* não foram realizadas em nenhum dos relatos, embora tenham sido citadas "reuniões com o cliente" (por exemplo, no *Telessaúde*). O que pode ser inferido é que as reuniões de *Review* e, na maioria dos casos, Retrospectiva, são feitas conjuntamente com a *Sprint Planning* da próxima *Sprint*. Por exemplo, na *Sprint Planning* acompanhada na visita à Endeeper, foram analisadas as funcionalidades desenvolvidas no *Sprint* passado (característica da *Review Meeting*) e acontecimentos bons e ruins (característica da Retrospectiva) antes de entrar no planejamento da próxima *Sprint*, o que é parte da *Sprint Planning* em si.

Percebeu-se ainda que as nomenclaturas corretas nem sempre são utilizadas. Muitas vezes o time cria nomenclaturas próprias para se referir a artefatos ou reuniões, em especial no âmbito de *Scrum*. Incremento de produto potencialmente entregável sendo chamada de *Demo*; Funcionalidades com valor a ser entregue para os itens do *Sprint Backlog* e *Task Board* chamado de Quadro são alguns exemplos de nomenclaturas diferenciadas vistos nos relatos estudados.

Em relação aos *Acceptance criteria*, ou critérios de aceitação, eles foram citados somente no relato de observação da Endeeper, e pararam de ser utilizados devido à falta de tempo. Entretanto, eles confundem-se com os Testes de Aceitação (*Customer Testes*) de XP, que são utilizados com uma frequência um pouco maior (i.e., foram citados por três empresas que utilizam *Extreme Programming*). Essa fase de testes do usuário não é

integrada com o ciclo iterativo das metodologias e não há citação sobre o planejamento que seria necessário para os *acceptance criteria*, mas é um momento em que é avaliado se os pontos necessários para o usuário estão presentes no incremento de *software* desenvolvido.

O conceito de *Product Owner* é utilizado de uma forma geralmente diferente do que diz a teoria. O papel existe em praticamente todos os projetos, mas, no entanto, a teoria indica que o *Product Owner* teria um maior envolvimento, priorizando o *backlog* e estando mais presente fisicamente no ambiente de desenvolvimento. A realidade é que o *Product Owner* nem sempre está disponível e o time precisa priorizar o *backlog* de acordo com as sua experiência e pode haver dúvidas que não são sanadas devido à distância do *Product Owner*. Isso pode ser sanado com o *Scrum Master* assumindo algumas responsabilidades do *Product Owner* e sendo um facilitador da comunicação do cliente com o time, como no relato presencial da AG2. Quando o produto é interno da empresa, e não de um cliente externo, o papel de *Product Owner* é mantido com maior acuidade, pela facilidade de te-lo sempre disponível na equipe.

A maior diferença da teoria de *Scrum* para a realidade de mercado são os *releases*. As empresas normalmente não entregam versões intermediárias do projeto, ou seja, não transformam os incrementos de produto potencialmente entregáveis em um *software* funcional antes da finalização do projeto. Os *softwares* só são entregues quando a totalidade das histórias ou tarefas do *Product Backlog* é finalizada. O que pode acontecer é aprimoramentos posteriores, mas não há a previsão de lançamentos de versões intermediárias para o cliente começar a utilizar o produto antes de sua finalização completa.

Ferramentas de gerência são utilizadas com frequência para auxiliar no processo de monitoramento do projeto, principalmente por parte do *Scrum Master*. A documentação da equipe costuma ser mais ágil, com vários relatos citando o uso do *Google Docs* como ferramenta de documentação e comunicação (*Telessaúde, Sistema de Gestão de Ensino, Woompa, Endeeper, AG2*). Alguns projetos de grande complexidade e para clientes com estrutura mais tradicional exigem uma documentação mais extensa. Nesses casos, há maior análise preliminar, mas pode-se perceber que, mesmo sem a citação de *Agile Modeling*, as empresas tendem a aplicar a técnica, desenvolvendo somente os documentos necessários para o gerenciamento efetivo do projeto ou para suprir as necessidades do cliente.

Os conceitos mais utilizados em *Scrum* são os que caracterizam a metodologia. Os *Sprints* são usados em todos os relatos, mas nem todos os projetos utilizam a duração indicada na teoria, de um mês. Foram vistos relatos com *Sprints* de duas semanas (por exemplo, o *Sistema de Gestão de Ensino*), de uma semana (como no relato presencial da AG2) e até de duração variável dentro de um mesmo projeto, no relato presencial da Endeeper.

O *Scrum Master*, em geral, é o Gerente de Projetos da equipe. Algumas vezes foram contratados "treinadores" para ensinar a metodologia à equipe, como nos relatos da *Orbisat* e *Globo.com*, onde o papel de *Scrum Master* ficava com o treinador. Porém, essa realidade era só nos primeiros projetos da equipe que, depois de dominar a metodologia, assumia sozinha a gerência.

O *Product Backlog* sempre conta com uma análise preliminar e, na maioria das vezes, aumenta incrementalmente ao longo do projeto. As equipes implementam *Daily Scrum* com frequência, mesmo se o *Scrum Master* não pode estar presente. A reunião mais utilizada, além da *Daily Scrum*, é a *Sprint Planning*. Nela são definidos os tópicos do próximo *Sprint Backlog* que, usualmente, são atualizados no *Kanbam* ou *Task Board* do time.

Quando se trata de *Extreme Programming*, é ainda mais difícil encontrar pontos de concordância entre as equipes. Na maioria das vezes, XP não é encarado como uma metodologia, mas sim como um conjunto de práticas, o que não deixa de ser correto e é inclusive citado no livro básico da metodologia (BECK, 1999). No entanto, pode-se notar que, em geral, não há um estudo prévio da totalidade das práticas, sendo escolhidas apenas algumas para um uso informal. Em alguns casos, como na *Globo.com* e no *Telessaúde*, chega a ser citado que os desenvolvedores utilizaram "boas práticas de XP", mas que o âmbito do trabalho para implantação de metodologias ágeis era focado em *Scrum*.

Algumas práticas de XP confundem-se com conceitos de *Scrum*. Além dos já citados Testes de Aceitação, percebe-se que *On-Site Costumer* confunde-se com *Product Owner* (sendo que o cliente não está presente como dito na teoria das duas metodologias), o *Planning Game* confunde-se com as estimativas e os *Small Releases* confundem-se com os incrementos de produto potencialmente entregáveis e *Releases* de *Scrum*, sendo que não é comum que aconteçam lançamentos pequenos do produto para o cliente, mesmo que as funcionalidades já estejam prontas.

Uma prática que pode ser notada nas duas visitas presenciais de empresas que utilizam *Extreme Programming* (Woompa e Endeeper) é a Semana de 40 horas (*Susteinable Pace*). Os times organizam sua carga de trabalho para que ela, normalmente, não ultrapasse as 40 horas semanais e não sejam exigidas horas extras. Há exceções, o relato do *Sistema de Gestão de Ensino* diz explicitamente que essa foi uma prática que a equipe decidiu por não utilizar.

A prática de Metáfora de *Extreme Programming* não foi citada em nenhum relato analisado. *Design Simples* foi citado no relato presencial da Endeeper, para evidenciar que não é utilizado. Testes unitários, *Coding Standards* e *Test Driven Development* foram citados somente uma vez em toda a análise de relatos. Time Coeso e Propriedade Coletiva não chegaram a ser citados, mas foram identificados na Woompa. Não significa que essas práticas não sejam utilizadas por equipes de desenvolvimento de *software*, elas somente não estão sendo necessariamente associadas com a metodologia de *Exteme Programming*.

As três práticas mais utilizadas e associadas com *Extreme Programming* são Refatoração, que inclusive é usada com base na teoria (FOWLER, 2004), *Pair Programming* e, especialmente, *Continuous Integration*. Essa última utiliza ferramentas para sua aplicação, que são as mais diversas, dependento da decisão do time. Testes de Aceitação são bastante utilizados, mas nem sempre são associados com a metodologia, nem integrados com o ciclo de desenvolvimento.

6 CONCLUSÃO

Este trabalho apresentou um estudo sobre a realidade atual do mercado de trabalho brasileiro frente a metodologias ágeis, centrando em *Scrum* e *Extreme Programming*. A teoria foi estudada e relatos de aplicações práticas foram analisados, determinando assim a forma como essas metodologias e princípios são aplicados no dia-a-dia de empresas no Brasil e extraindo as práticas mais utilizadas, que podem servir como guia para profissionais que estão iniciando seus estudos na área.

As metodologias ágeis têm a proposta de deixar o processo de desenvolvimento de *software* mais rápido e simples. A sua popularidade tem muito a ver com a velocidade do mercado atual, onde respostas rápidas são cada vez mais exigidas. A variedade de interpretações sobre a teoria das metodologias tem também a ver com a velocidade, onde a curva de aprendizado de novos conhecimentos deve ser rápida e adequada à realidade em que a empresa está inserida.

Dessa forma, não existe forma certa ou errada de aplicar a metodologia escolhida. Não se pode descaracterizar o que está sendo aplicado, mas a realidade mostra que as práticas centrais das metodologias são aplicadas, o que sofre adaptação são as práticas de complementação. As empresas devem ter estudos constantes para encontrar novas soluções que tragam ganhos para a qualidade do *software* por elas desenvolvido, para a velocidade da entrega e também para a motivação de seus funcionários.

Os princípios ágeis são amplos e podem ser considerados como uma "filosofia" da empresa, sem precisar se ater a uma metodologia em específico. Atualmente, falar de métodos ágeis no Brasil é quase como dizer que a sua equipe aplica *Scrum*, apesar do Manifesto Ágil ter sido corroborado por representantes de sete metodologias diferentes. Isso se considerarmos que grandes empresas começam agora uma mudança que já era vista há mais tempo dentro das comunidades de *software*. Nelas, observa-se discussões sobre outras metodologias, inclusive algumas mais novas que o Manifesto Ágil, como *Lean*, que daqui a algum tempo podem levar a uma nova mudança de cenário.

As dificuldades encontradas devem-se a poucos estudos aprofundados sobre metodologias ágeis no Brasil, o que leva a pouco material publicado em *papers*, simpósios e convenções para estudo de caso. Encontra-se muito material informal, relatos em *sites* pessoais, *blogs* e artigos opinativos, mas poucos estudos teoricamente embasados. Por outro lado, as opiniões e interpretações abundam na *web*, e a bibliografia sobre o assunto é bastante vasta, especialmente títulos em língua inglesa, pois não existem muitas publicações nacionais sobre o assunto e somente alguns autores como Beck e Ambler possuem edições traduzidas. Isso demanda cuidado e análise para

discernir os materiais, o que dizem os autores que efetivamente criaram a metodologia, o que é análise, o que é relato e o que é opinião pessoal.

Os trabalhos futuros incluem experimentos práticos de aplicações e combinações de metodologias ágeis ou tradicionais, nos âmbitos gerenciais, de desenvolvimento e teste, para o estudo de interações entre as mesmas, as mudanças na performance das equipes e no ambiente de trabalho. Com isso, determinar uma combinação de metodologias que traga maior ganho para determinados nichos de empresas de desenvolvimento, focando em sua especialidade e tamanho.

A computação ainda é uma área relativamente nova e extremamente mutável. As metodologias devem se adaptar à realidade de mercado, e não o contrário. Nenhuma empresa deve forçar situações para se adaptar a uma metodologia se isso traz mais dificuldades do que benefícios. Isso é parte da evolução da computação, pois, se não houvesse o ímpeto de adaptação e criação inerente à Ciência da Computação, talvez não se tivesse chegado ao nível tecnológico atual, de forma que a mutação constante das técnicas e ferramentas com as quais os *softwares* são criados é parte essencial para a evolução da tecnologia.

REFERÊNCIAS

AMBLER, S. **Agile Modeling: Effective Practices for eXtreme Programming and the Unified Process.** 1ª edição. [S.l.]: Wiley, 2004.

ANDERSON, C. **Free: Grátis – O Futuro dos Preços**. 1ª edição. [S.l]: Ed. Campus, 2009.

BARROS et al**. SCRUM: Uma metodologia ágil para projetos WEB com pequenas equipes**. In: IX Jornada de Ensino, Pesquisa e Extensão (JEPEX 2009). IX Jornada de Ensino, Pesquisa e Extensão , Universidade Federal Rural de Pernambuco, Serra Talhada, 2009.

BECK, K. et al. Manifesto for Agile Software Development. **Agile Manifesto**, Snowbird, 2001. Disponível em: http://agilemanifesto.org. Acesso em: jun. 2010.

BECK, K. **Extreme Programming explained: Embrace Change**. Edição US. Boston: Addison-Wesley, 2000.

BECK, K. **Test-Driven Development: By Example**. 1ª edição. Boston: Addison-Wesley, 2002.

BRUNHERA, D.; ZANATTA, A. ***Scrum*: Uma aplicação em uma *software house***. 2010. 18 f. Projeto de Diplomação (Bacharelado em Ciência da Computação) – Instituto de Ciências Exatas e Geociências, UPF, Passo Fundo.

CASTRO, E.S.E. ; VICTORETI, F. ; FIORINI, S. R. ; ABEL, M. ; PRICE, R. T. . **Um Caso de Integração de Gerenciamento Ágil de Projetos à Metodologia CommonKADS**. In: I Workshop de Gerenciamento de Projetos de Software (WGPS), 2008, Florianópolis. I Workshop de Gerenciamento de Projetos de Software, 2008. v. 1.

CHAPIESKI, G. Como estamos indo com a adoção de Scrum na Globo.com. **Guilherme Chapieski - Blog sobre desenvolvimento de software e tecnologia**, [S.l.], 2008. Disponível em: <http://gc.blog.br/2008/05/27/como-estamos-indo-com-a-adocao-de-scrum-na-globocom/>. Acesso em: maio 2011.

COHN, M. Estimating With Use Case Points. **Methods & Tools**, [S.l.], 2005. Disponível em : <http://www.methodsandtools.com/archive/archive.php?id=25>. Acesso em: abr. 2011.

COHN, M. **User Stories Applied: For Agile Software Development.** Michigan: Addison Wesley Professional, 2004.

COHN, Mike. Introduction to Scrum - An Agile Process. **Mountain Goat,** Lafayette, 2002. Disponível em: <http://www.mountaingoatsoftware.com/topics/scrum>. Acesso em: ago. 2010.

DUNCAN, S. et al. Scrum Is An Inovattive Approach to Make Things Done. **Scrum Alliance,** Indianapolis, 2005. Disponível em: <http://www.scrumalliance.org/learn_about_scrum>. Acesso em: jun. 2010.

FOWLER et al. **Refatoração: Aperfeiçoando o Projeto de Código Existente.** 1ª edição. [S.l]: Bookman, 2004.

FRIED, J.; HANSON, D. **Rework**. [S.l.]: Crown Business, 2004.

GLOGER, B. The Zen of Scrum. **Bor!sGloger**, Baden-Baden, 2007. Disponível em: <http://www.glogerconsulting.de> . Acesso em: mar. 2011

GYGI et al. **Six Sigma for Dummies.** [S.l.]: For Dummies, 2005.

JEFFRIES, J. The trouble with Tasks. **XProgramming.com – An Agile Software Developmente Resource**, [S.l.], 2010. Disponível em: <http://xprogramming.com/beyond/the-trouble-with-tasks/> . Acesso em: maio 2011.

JEFFRIES, R. What is Extreme Programming. **XProgramming.com – An Agile Software Developmente Resource**, [S.l.], 1999. Disponível em: <http://xprogramming.com>. Acesso em: fev. 2011.

KNIBERG, H; SKARIN, M. Kanban e Scrum - obtendo o melhor de ambos. **InfoQ**, [S.l.], set. 2010. Disponível em: <http://www.infoq.com/br/minibooks/kanban-scrum-minibook>. Acesso em: mar. 2011.

LUNA et al. **Desenvolvimento Distribuído de uma Aplicação de Telessaúde com a Metodologia Ágil SCRUM.** In: IX Congresso Brasileiro de Informática em Saúde (CEBIS 2008). IX Congresso Brasileiro de Informática em Saúde: Campos do Jordão, 2008.

MARÇAL et al. Integração de Story Points e Use Case Points em Projetos Baseados em SCRUM e CMMI. In: VIII Simpósio Brasileiro de Qualidade de Software (SBQS 2009), 2009, Ouro Preto. **Anais do VI Simpósio Brasileiro de Qualidade de Software (SBQS 2009)**, 2009. p. 394 - 401.

MARCHESI, M., SUCCI, G., WELLS, D. e WILLIANS, L. **Extreme Programming Perspectives**. Michigan: Addison Wesley Professional, 2002.

MESQUITA, L. Implementando Scrum – mais um case de sucesso. **Leandro Mesquita – Um Blog sobre Desenvolvimento Ágil XP**, [S.l], abr. 2010 Disponível em: <http://leandromesq.wordpress.com/2010/04/19/mais-um-case-de-sucesso/>. Acesso em: jul. 2010.

PRESSMAN, R. S. **Engenharia de Software**. 6ª edição. [S.l.]: Ed.Mc Graw Hill, 2007.

SAVOINE, Márcia, MARTINS, Lucyano, ROCHA, Mayton, SANTOS, Cirlene dos. Análise de Gerenciamento de Projeto de Software Utilizando Metodologia Ágil XP e Scrum: Um Estudo de Caso Prático. XI Encontro de Estudantes de Informática do

Tocantins, 2009, Palmas. In: **Anais do XI Encontro de Estudantes de Informática do Tocantins**. Palmas: Centro Universitário Luterano de Palmas, 2009. p. 93-102.

SCHREIBER, G.; AKKERMANS, H.; ANJEWIERDEN, A.; HOOG, R.; SHADBOLT, N.; DE VELDE, W. V.; and WIELINGA, B. **Knowledge Engnineering and Management: the CommonKADS Methodology**. Cambridge: MIT Press. 2002.

SCHWABER, K. ; SUTHERLAND, J. **Scrum Guide**, [S.l.], 2009. Disponível em: <http://www.scrum.org/ >. Acesso em: jun. 2010.

SCHWABER, K. **Agile Project Management With Scrum**. 1ª edição. [S.l]: Microsoft Press, 2004.

SCHWABER, K.; BEEDLE, M. **Agile Software Development with Scrum**. 1ª edição. [S.l.]: Prentice Hall, 2001.

SIMS, C. Burn Stories Not Tasks. **InfoQ**, [S.l.]: jan. 2009. Disponível em: <http://www.infoq.com/news/2009/01/Burn-Stories-Not-Tasks>. Acesso em: maio 2011.

TAKEUSHI, H.; NONAKA, I. **The New New Product Development Game**. [S.l.], Harvard Business Review, 1986.

TORRES, L. AG2 também usa metodologias ágeis de desenvolvimento. **Blog da Locaweb**, [S.l.], dez. 2008. Disponível em: <http://blog.locaweb.com.br/archives/3459/ag2-tambem-usa-metodologias-ageis-de-desenvolvimento/>. Acesso em: ago. 2010.

WELLS, D. Extreme Programming: A Gentle Introduction. **Extreme Programming**, [S.l.], 2009. Disponível em: <http://www.extremeprogramming.org/>. Acesso em: out. 2010.

ANEXO A: GUIA PARA INICIANTES

Para inserção no mercado, é de interesse que o profissional esteja atualizado e domine o ferramental mais utilizado. Estudos aprofundados são essenciais para o entedimento completo de determinado assunto, no entanto, nem sempre há tempo hábil para análises teóricas mais aprofundadas, de forma que é de interesse saber os pontos principais do tópico para que a curva de aprendizado seja mais rápida.

Com a análise dos relatos, identificação de seus pontos em comum, o que é largamente utilizado e o que não é, pode-se extrair um grupo de características e práticas que servem como guia para profissionais iniciando seus estudos na área de metodologias ágeis.

Sendo assim, as práticas mais utilizadas pelo mercado podem ser utilizadas como um Guia para Iniciantes, para iniciar os estudos pelas práticas de maior valor de mercado da metodologia.

Em *Scrum*, é importante estudar o que forma a estrutura da metodologia. A natureza iterativa, que é uma das principais mudanças das formas tradicionais de desenvolvimento para as ágeis, e os papéis principais de projetos, mas que são reformulados por *Scrum* são os conceitos mais utilizados pelo mercado. É muito importante o profissional ter conhecimento sobre as reuniões e pelo artefato mais utilizado para começar em um ambiente de projeto *Scrum* com maior facilidade.

A Tabela 1 traz as práticas de *Scrum* mais utilizadas no mercado atual, e motivações mercadológicas para que os estudos sejam iniciados por elas.

Tabela 1: Práticas de *Scrum* mais utilizadas no mercado

Prática *Scrum*	**Motivação**
Sprint	A natureza iterativa de *Scrum* é o cerne da metodologia, colocada por todas as empresas que a utilizam. A duração da *Sprint* muda conforme o projeto, mas a motivação de desenvolver em ciclos permanece a mesma.
Scrum Master	O papel é normalmente desempenhado pelo Gerente de Projeto, atuando como facilitador. É um dos papéis mais diferenciados do time, portanto é de

Prática *Scrum*	**Motivação**
	interesse um estudo sobre suas atividades.
Product Owner	Papel desempenhado pelo cliente, ou seja, está sempre presente em qualquer projeto. No entanto, é importante ter ciência de que a teoria prega uma realidade difícil de ser alcançada, de te-lo sempre presente. É de interesse o estudo dessa prática.
Daily Scrum	É a prática mais utilizada para comunicação nas empresas, mesmo em projetos distribuídos.
Sprint Planning	Depois da *Daily Scrum*, é a reunião mais utilizada para planejamento e comunicação. Essa reunião por vezes se mistura com a Retrospectiva e *Sprint Review*, portanto, é de interesse estuda-las.
Kanbam ou *Task Board*	O quadro que permite acompanhamento do que está sendo desenvolvido é utilizado por boa parte das empresas analisadas, em sua versão física ou online. Nele, utiliza-se o conceito de *Sprint Backlog*, com as histórias ou tarefas com as quais o time se comprometeu para a *Sprint* atual.

No âmbito de *Extreme Programming*, poucas práticas foram aplicadas em mais de um relato. As práticas são complexas e a maioria tem bibliografia própria, além de poderem ser usadas individualmente. Ao aplicar XP, é importante que as equipes se foquem nos princípios, que acabam por levar naturalmente às práticas.

Alguns pontos em comum foram identificados e podem ajudar profissionais iniciantes a decidir quais a práticas se dedicar primeiro. A Tabela 2 sintetiza essas práticas e motivações pra seu estudo.

Tabela 2: Práticas de *XP* mais utilizadas no mercado

Prática de *Extreme Programming*	**Motivação**
Refatoração	Prática que simplifica o código, é utilizada no mercado tanto como parte de XP, quanto como prática individual.
Pair Programming	Utilizada em diversos âmbitos, *pair programming* pode ser usado para *refatorar* o código, desenvolvê-lo ou passar o conhecimento entre a equipe, facilitando outra prática de XP, a

	Propriedade Coletiva do Código.
Continuous Integration	Prática mais citada nos relatos. É de grande valor de mercado pois ajuda a identificar os problemas do código mais rapidamente e, como isso, diminuir os custos para a correção.

Esses tópicos são somente um guia inicial, para facilitar e motivar a aprendizagem da metodologia com conceitos mais utilizados na prática. É importante estudar todos os conceitos da metodologia que a equipe utiliza para um conhecimento mais amplo e completo e, assim, a decisão de quais se adequeam à empresa, equipe e profissionais, e que trará ganhos de qualidade e produtividade.

Para que o estudo dos conceitos e sua subsequente aplicação sejam feitos de acordo com a teoria, são sugeridos os seguintes livros e *websites* para a consulta:

- Manifesto for Agile Software Development: http://agilemanifesto.org
- Ken Schwaber: Agile Project Management With Scrum
- Ken Schwaber e Mike Beedle: Agile Software Development with Scrum
- Kent Beck: Extreme Programming explained: Embrace Change
- Scrum Alliance: http://www.scrumalliance.org
- Don Wells: http://www.extremeprogramming.org/

ANEXO B: CONCEITOS IMPORTANTES

Os projetos de Tecnologia de Informação são plurais, utilizando combinações de conceitos, artefatos e métodos que, a primeira vista, parecem não poder conviver conjuntamente. Isso pode acontecer por diversos motivos: retro-compatibilidade, cultura empresarial, talvez somente uma equipe com conhecimentos diversos que resultou em uma distribuição não usual de técnicas.

Dessa forma, o presente Anexo apresenta alguns conceitos que não são ligados intimamente com nenhuma metodologia ágil, mas são de importância para o entendimento pleno dos relatos descritos adiante nesse trabalho.

1 ROI

Sigla do inglês *Return of Investiment*, em tradução literal, retorno do investimento. Basicamente, é o foco, do *Product Owner*, de fazer o projeto levar para a sua empresa o retorno (financeiro, de praticidade, etc.) ao investimento feito (SCHWABER, 2004).

2 Ferramentas Case

A sigla CASE significa "Computer-Aided Software Engineering". Em português: "Engenharia de Software Auxiliada por Computador". Segundo Pressman (2007), "a oficina do engenheiro de software é o ambiente integrado de suporte a projetos e o conjunto de ferramentas que preenchem a oficina é o CASE". As ferramentas CASE proporcionam ao engenheiro de software a capacidade de automatizar as atividades manuais e melhorar as informações de engenharia.

3 CMM

O CMM (*Capability Maturity Model)* descreve os principais elementos de um processo de desenvolvimento de software, os estágios de maturidade por que passam as organizações enquanto evoluem no seu ciclo de desenvolvimento de software, através de avaliação contínua, identificação de problemas e ações corretivas, dentro de uma estratégia de melhoria dos processos. Este caminho de melhoria é definido por cinco níveis de maturidade: Inicial, Repetível, Definido, Gerenciado e Otimizado.

O CMM fornece às organizações orientação sobre como ganhar controle do processo de desenvolvimento de software e como evoluir para uma cultura de excelência na gestão de software. O objetivo principal nas transições através desses níveis de maturidade é a realização de um processo controlado e mensurado que tem

como fundamento a melhoria contínua. A cada nível de maturidade corresponde um conjunto de práticas de software e de gestão específicas, denominadas áreas-chave do processo (KPAs - *Key Process Areas*). Estas devem ser implantadas para que a organização possa atingir o nível de maturidade desejado.

4 CommonKADS

O CommonKADS, modelo de engenharia e gestão do conhecimento desenvolvido por Schreiber et al. (2002), concebe o conhecimento em relação ao propósito e ao contexto, focado na ação, esperando benefícios de um sistema de conhecimento no aumento da rapidez e a melhoria da qualidade na tomada de decisão.

O CommonKADS parte do pressuposto de que o conhecimento pode ser modelado num sistema, visando à melhoria da qualidade, da produtividade e a agilidade na tomada de decisão. Segundo seus idealizadores: "A engenharia do conhecimento permite focar as oportunidades e gargalos a respeito de como as organizações desenvolvem, distribuem e aplicam seus recursos de conhecimento, de modo a fornecer as ferramentas para a gestão do conhecimento corporativo" (SCHREIBER et al., 2002, p.7).

5 PMBok

O *Project Management Body of Knowledge*, também conhecido como PMBok, é um conjunto de práticas em gerência de projetos publicado pelo *Project Management Institute* (PMI) e constitui a base do conhecimento em gerência de projetos do PMI. Estas práticas são compiladas na forma de um guia, chamado de Guia do Conjunto de Conhecimentos em Gerenciamento de Projetos, ou Guia PMBok.

O Guia PMBOK é o guia que identifica um subconjunto do conjunto de conhecimentos em gerenciamento de projetos, que é amplamente reconhecido como boa prática, sendo em razão disso, utilizado como base pelo *Project Management Institute* (PMI). Uma boa prática não significa que o conhecimento e as práticas devem ser aplicadas uniformemente a todos os projetos, sem considerar se são ou não apropriados.

O Guia PMBOK também fornece e promove um vocabulário comum para se discutir, escrever e aplicar o gerenciamento de projetos possibilitando o intercâmbio eficiente de informações entre os profissionais de gerência de projetos.

6 Use Case Points

A UCP é uma técnica de estimativa de tamanho de projeto de software orientado a objetos, criada por Karner (1993). As estimativas com UCP são realizadas a partir da contagem do número de transações dos casos de uso. A quantidade de transações determina a complexidade de casos de uso, que, juntamente com a complexidade dos atores e os fatores técnicos e ambientais do projeto, determina o tamanho do produto.

7 Six Sigma

Six Sigma é uma metodologia para minimização de erros e maximização de valor. Cada erro que uma organização ou pessoa comete tem um custo – um consumidor

perdido, queda na produtividade, etc. De fato, erros custam às empresas em torno de 20% a 30% de seus orçamentos (GYGI et al., 2005).

Six Sigma costumava ser uma metodologia pra melhoria da qualidade, mas hoje é uma abordagem geral de minimização de erros e maximização de valor: quantos produtos podem ser produzidos, quantos serviços podem ser entregues, quantas transações podem ser completadas no menor tempo e com o custo mais baixo possível?

A ação do Six Sigma ocorre em duas frentes: gerencial e técnica. Em nível gerencial, uma iniciativa Six Sigma inclui muitas unidades, pessoas, tecnologias, projetos, horários, e detalhes a serem gerenciados e coordenados (GYGI et al., 2005).