UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ – UTFPR
DIRETORIA DE PESQUISA E PÓS-GRADUAÇÃO
ESPECIALIZAÇÃO EM ENGENHARIA DE SOFTWARE

ALESSANDRO DA VEIGA

UMA ABORDAGEM SIMPLIFICADA DO PENSAMENTO *LEAN* EM METODOLOGIAS ÁGEIS

MONOGRAFIA DE ESPECIALIZAÇÃO

MEDIANEIRA
2012

ALESSANDRO DA VEIGA

UMA ABORDAGEM SIMPLIFICADA DO PENSAMENTO *LEAN* EM METODOLOGIAS ÁGEIS

Monografia apresentada como requisito parcial à obtenção do título de Especialista na Pós Graduação em Engenharia de Software, da Universidade Tecnológica Federal do Paraná – UTFPR – Câmpus Medianeira.

Orientador: Prof Juliano Rodrigo Lamb.

MEDIANEIRA

2012

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Diretoria de Graduação e Educação Profissional
Especialização em Engenharia de Software

TERMO DE APROVAÇÃO

Uma abordagem simplificada do pensamento *Lean* em metodologias ágeis

Por

Alessandro da Veiga

Esta monografia foi apresentada às 14:40 h do dia 16 de março de 2012 como requisito parcial para a obtenção do título de Especialista no curso de Especialização em Engenharia de Software, da Universidade Tecnológica Federal do Paraná, Campus Medianeira. O acadêmico foi arguido pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

_____
Prof. M.Sc Juliano Rodrigo Lamb
UTFPR – Câmpus Medianeira
(orientador)

_____
Prof. M.Sc Alan Gavioli
UTFPR – Câmpus Medianeira

_____
Prof M.Sc. Alessandra B. Garbelotti Hoffmann
UTFPR – Câmpus Medianeira

RESUMO

VEIGA, Alessandro Da. Uma abordagem simplificada do pensamento Lean em metodologias ágeis. 2012. 51 p. Monografia (Especialização em Engenharia de Software). Universidade Tecnológica Federal do Paraná, Medianeira, 2012.

Este trabalho apresenta um estudo sobre as principais características que as metodologias de desenvolvimento ágil possuem em comum que se enquadram nos pensamentos *Lean*, para prever questões dinâmicas em nível de constatação e execução de projetos de desenvolvimento de *software* que não agregam valor sobre demanda durante a concretização das tarefas, demonstrado através de um estudo experimental fictício.

Palavras-chave: Desenvolvimento de software, Entrega sem desperdício, Características ágeis.

ABSTRACT

VEIGA, Alessandro da. A simplified approach of Lean thinking in agile methodologies. 2012. 51 p. Monografia (Especialização em Engenharia de Software). Universidade Tecnológica Federal do Paraná, Medianeira, 2012.

This project presents a study on the main characteristics that the methodologies of fast development have in common. These are included in Lean premises to predict dynamic questions related to verification and execution of software development project, which don’t have any added value over demand during the completion of tasks, as demonstrated through a case study.

Keywords: Software development, Delivery without waste, Dynamic features.

ÍNDICE DE FIGURAS

ÍNDICE DE QUADROS

SUMÁRIO

1 INTRODUÇÃO

A melhor maneira de obter um software de qualidade é reduzir o número de ações não utilizadas, que proporcionam produtos que não agregam muitas funcionalidades, influenciem sobre os projetos que buscam inovação e tecnologia. Obter um estudo claro da engenharia do *software* e dos *frameworks* é um bom inicio para quem quer chegar a algo simples e ao mesmo tempo sofisticado aos olhos de quem vê o projeto funcionando.

A velocidade com que ocorreu o desenvolvimento tecnológico nos últimos 50 anos é certamente impressionante. Em poucos anos talvez as tecnologias de informação que hoje são padrões da indústria, podem desaparecer com a mesma velocidade com que surgiram. Elas darão lugar a outras, com a mesma visão, porém com diferenciais gerados a partir de uma evolução rápida, algumas vezes se tornando novamente padrões da indústria. Outras, porém, desaparecem com a mesma velocidade que vieram. Esta mesma evolução pode ser também notada nos métodos utilizados para desenvolver software, mas possivelmente de uma forma não tão rápida. (CRESCÊNCIO, 2011)

Soluções vendidas em larga escala, todas com o mesmo nível de componentes e funcionalidades não são o que as empresas que visam modernização buscam de um produto, que tem a função de prever, ou pelo menos demonstrar de forma segura, se os objetivos estão sendo alcançados, por parte de quem a está utilizando. Uma empresa ao obter uma solução desse gênero está se igualando em nível de serviços executados a outras, que muitas vezes não possuem estrutura para suportar tamanho conjunto de necessidades requisitadas pelo cliente, deixando assim um cliente insatisfeito.

De forma ostensiva, porém não tão instantânea, os métodos ágeis surgem com o pressuposto de deferir que o desenvolvimento de software serve como melhoramento e obtenção de lucros. Esse recurso serve para o mundo atual como uma maneira de extravasar os métodos que buscavam maior previsibilidade para o desenvolvimento de software, procurando entender de antemão e geralmente fixar qual seria o escopo, o custo e o prazo de desenvolvimento. (CRESCÊNCIO, 2011)

1.1 OBJETIVO GERAL

Apresentar as características associadas ao desenvolvimento ágil de software que contribuem no processo de desenvolvimento de *software*, mediante o pensamento *Lean*.

1.2 OBJETIVOS ESPECÍFICOS

- ⇨ Apresentar os dois *frameworks* que visam agilidade no processo de software mais utilizados, que são *Kanban* e *Scrum*, e que estão enquadradas nos princípios do pensamento *Lean* (enxuto) para obtenção de um bom projeto;
- ⇨ Apresentar as experiências e resultados esperados de cada um dos métodos, adquiridas através de gráficos ou figuras, que identificam os desvios encontrados na produção de um software, que demonstrem percas e ganhos significativos e que devem ser utilizados dentro do pensamento *Lean*;
- ⇨ Demonstrar as características herdadas e comuns para cada *framework*, que são apenas prescrições de artefatos encontrados, que podem ou não serem seguidas, dependendo muito da situação e do objetivo;
- ⇨ Identificar as melhorias adquiridas por uma equipe que adota o pensamento *Lean*, dentro dos artefatos escolhidos para utilização de um *framework* que se adequa às metodologias ágeis.

1.3 JUSTIFICATIVA

Analisando o cenário de projetos da área de Tecnologia da Informação, percebe-se que, mesmo com os esforços e investimentos realizados, as empresas

têm falhado sistematicamente na entrega de seus projetos de desenvolvimento de sistemas. Pesquisas apontam diferentes causas para esse insucesso, entre elas, a falta de domínio de métodos e técnicas e/ou a adoção de práticas errôneas de gerenciamento de projetos. Em face dessa situação, verifica-se a existência de uma lacuna entre a necessidade dessas empresas e os resultados práticos de desempenho alcançados. (GRANDO, 2010)

Os princípios do *Lean* são muito bem conhecidos por várias empresas que buscam limitar o desperdício de recursos, porém as organizações não estão aplicando a mesma uniformemente, porque o pensamento *Lean* requer uma mudança na cultura e hábitos organizacionais de uma empresa, que estão além de sua capacidade. Por outro lado às empresas que tem entendido e adotado a essência do pensamento *Lean*, perceberam significativas melhoras no seu desenvolvimento sustentável. (POPPENDIECK e POPPENDIECK, 2003)

Não obstante a inovação é gradativa e necessária, cabe a cada empresa definir o que melhor se adapta ao seu modelo de trabalho e só assim colher os frutos desejados. Quem trabalha em um sistema que as situações movem a equipe, tem um bom resultado, e melhor identifica os problemas que possam impossibilitar o projeto de ser entregue na data prevista ou até demonstrar ao cliente a raiz da demora em determinada situação.

Se o princípio mais importante do *Lean* é gerar um fluxo contínuo de valor, precisa-se aprender a identificar o que é desperdício. A mente humana não foi treinada para identificar aspectos negativos com facilidade. Há uma tendência natural do homem a olhar para aquilo que gosta e identificar mais claramente as coisas boas do que as ruins. Por este motivo, precisa-se exercitar a percepção sobre o que é desperdício. Aqui temos uma boa definição do que é desperdício segundo a perspectiva *Lean*: “Desperdício é tudo aquilo que esgota recursos de tempo, dinheiro e espaço sem adicionar valor ao cliente”. (CRESCÊNCIO, 2011)

Para isso estudar novas maneiras de se enquadrar no mercado produtivo de software de qualidade é uma das melhores formas de se garantir no páreo e ter uma visão avante no sistema. Tecnologia é algo primordial, e talvez obrigatório dependendo do que se pretende atingir.

Processos ágeis proporcionam total visibilidade, controle e rastreabilidade de tudo o que ocorre durante o ciclo de desenvolvimento. De fato, os métodos ágeis propiciam uma oportunidade diária para análise de riscos e tomada de decisão de

modo a corrigir o mínimo desvio indevido de curso. Todas as ocorrências são disponibilizadas através dos componentes de gestão como *dashboard* e *burndown charts* para todos os membros do projeto. Além disso, a comunicação direta entre equipes gera maior colaboração, visibilidade e controle do projeto. O próprio processo de ciclos curtos proporciona maior aprendizado e *feedback* concreto sobre o exato andamento do projeto, gerando maior segurança e consequente aumento de autoestima para todos os envolvidos. (CRESCÊNCIO, 2011)

Os métodos ágeis atualmente são o que melhor se adaptam as novas tecnologias, e demonstra-los é a melhor forma para garantir vazão e entendimento na linha produtiva. Gráficos que demonstram o andamento dos trabalhos da equipe ou simples abstrações dos mesmos em imagens trazem isso para a realidade, para que qualquer pessoa desenquadrada do procedimento se adeque e consiga se expressar e até trazer melhorias, que muitas vezes quem está no meio muitas vezes não identifica.

1.4 ESTRUTURA DO TRABALHO

No Capítulo 2 deste trabalho serão definidos melhor os princípios dos métodos ágeis que mais são encontrados no mercado. Após será obtida suas raízes e porque não as suas comparações e características em comum.

No Capítulo 3 será demonstrado um caso de uso fictício que será utilizado para apresentar as características dos artefatos de tecnologias ágeis dentro do pensamento *Lean*.

Por fim, no Capítulo 4, as características apresentadas de forma descritiva a atender os conceitos do pensamento *Lean* dentro do caso de uso proposto.

2 FUNDAMENTAÇÃO TEÓRICA

Na ultima década, o movimento para diminuir o peso e aumentar a agilidade tem sido a mudança mais significativa que afetou as empresas de *software* desde o advento do modelo em cascata na década de 1970. Originando uma variedade de pensamentos e práticas líderes e comprovadas no mundo real, em experiências bem sucedidas, os métodos ágeis tem se provado excelentes para entregar benefícios para as medidas "*big four*": produtividade, qualidade, moral e tempo de mercado. (LEFFINGWELL, 2010)

Nota-se que um projeto bem sucedido não depende exclusivamente de métodos seguidos de forma veemente. Aplicar lições aprendidas no dia-a-dia não só ajudam na qualidade do *software* como também ajudam na autoestima da equipe. Seguir uma receita de bolo quando aplicada as novas tecnologias não traz agilidade e qualidade que o cliente deseja. Os métodos ágeis são uma prova disso, não só devido ao crescimento diário de adeptos como também pela inovação que traz a uma equipe que o aplica em níveis de segurança e conhecimento agregado.

O resultado esperado para qualquer solução é sempre um grande número de funcionalidades visíveis e aplicáveis ao uso diário. Não adianta um produto enorme e pesado que aplique pouco ou quase nada a realidade necessária ao cliente. Para o cliente não adianta centenas de páginas de documentação para revisar e aplicar, e difíceis de lembrar, o que interessa mesmo são os mecanismos intuitivos.

2.1 DESENVOLVER COM TECNOLIGIA

Por muito tempo a indústria se ancorou em métodos que buscavam maior previsibilidade para o desenvolvimento de *software*, procurando entender de antemão e geralmente fixar qual seria o escopo, o custo e o prazo de desenvolvimento. Ao longo dos anos, contudo, percebeu-se que a tão esperada previsibilidade não se provou verdadeira, especialmente para projetos nos quais se busca o desenvolvimento científico de tecnologias que ainda não existem. Contudo, mesmo para projetos em que a tecnologia é de certa forma trivial, a complexidade

dos negócios envolvidos torna também o processo imprevisível. É muito comum o mercado mudar suas necessidades, ou mesmo o próprio cliente não saber o que quer até que comece a ver partes disso funcionando. Quando o conhecimento sobre a tecnologia e sobre os negócios é baixo, a gestão dos projetos torna-se muito complexa, algumas vezes gerando caos e anarquia. Esses conceitos podem ser visualizados FIGURA 1. (CRESCÊNCIO, 2011)

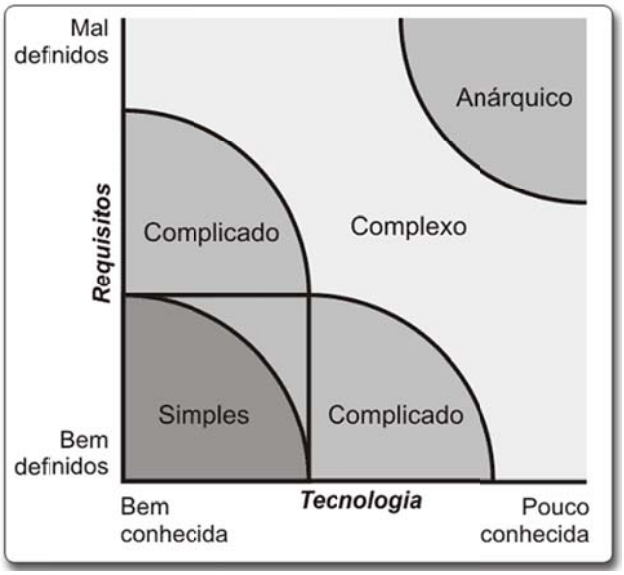

FIGURA 1 - Interseção do conhecimento entre tecnologia e negócio.
FONTE: (CRESCÊNCIO, 2011)

O valor de mercado esta na satisfação do cliente e não na complexidade do produto entregue. Para o cliente quanto mais intuitivo o produto se mostra, menos esforço deve partir de sua parte, consequentemente tem-se uma maior qualidade agregada a um numero maior de utilidades que realmente serão utilizadas pelo usuário. Outra grande importância ao cliente é saber o que realmente esta acontecendo na linha de produção de seu produto, ou seja, acompanhar caso a caso as funcionalidades a serem entregues e os problemas e discussões que surgiram no caminho. Quando o cliente se propõe a revisar os problemas não são encontrados conflitos entre as partes, tendo assim justificativas hábeis e propostas bem compreendidas.

O tempo de mercado ainda prova ser um dos fatores chaves, ao lado de qualidade e custo de mercado, para empreendimentos prósperos embutidos em um mercado dinâmico alto com complexidade crescente dos produtos e mercados

globais com curtos ciclos de vida dos produtos. O curto tempo para comercializar é um dos elementos fundamentais. Ele provê a chance de ganhar quotas de mercado mais elevadas, e assim aumentar a receita total. Ao mesmo tempo em que permite um avanço mais rápido na curva de aprendizagem que pode dar a empresa uma significativa vantagem de custo comparada com os competidores deixados para trás. Em uma perspectiva de longo prazo ela pode fornecer a empresa maior nível de complexidade aos negócios. Isso permite responder a demanda do cliente mais rápido do que a concorrência pode oferecer, e assim oferecer aos clientes produtos que eles querem quando eles querem. (BAUCH, 2004)

Quanto menos tempo o produto leva para ficar pronto, menos o cliente perde com soluções alternativas para validar os conceitos e produtos necessários em sua produção. E quanto mais o cliente interagir com desenvolvimento dos requisitos aplicáveis mais consistência se da à solução final e menos controvérsias se descriminam entre recursos não disponíveis ou que não são totalmente aplicáveis. Para o cliente que não está de acordo com todas as discussões de previsibilidade e incoerência entre um produto bem sucedido e a solução assim proposta pelo mesmo, vão sendo abertas as portas, e trazendo assim um ambiente de simpatia entre ambas as partes.

2.2 MANTER A AGILIDADE

O desenvolvimento de *software* Ágil requer inovação e responsabilidade, baseada na geração e compartilhamento de conhecimento entre a equipe de desenvolvimento e o cliente. Desenvolvedores que se propõe a metodologias de entregas rápidas aproveitam os pontos fortes dos clientes, usuários, e desenvolvedores, buscando somente o necessário para o equilíbrio entre qualidade e agilidade. (POPPENDIECK e POPPENDIECK, 2003)

Qualidade e agilidade são preceitos muito requisitados em uma relação completa de custo benefício produto e desenvolvimento. Para determinar alguns mecanismos de comum acordo entre vários experientes na área de *software*, que bem conhecem os recursos que podem ou não ser aplicados durante o desenvolvimento de um *software*, se reuniram a fim de minimizar suas perdas e

porque não a de outros adeptos que se iniciavam na arte do bom atendimento e fornecimento de ótimos recursos aos produtos de sua linha de produção entregue a seus clientes.

Esses conhecedores de práticas antigas mal ou bem sucedidas se propuseram a sugerir um novo pensamento entre os engenheiros, analistas, desenvolvedores e usuário, a fim de diminuir as perdas e aumentar produtividade, e talvez porque não melhorar os lucros. Esse grupo se propôs em um encontro formal a unir forças e prescrever, ou simplesmente sugerir novas maneiras de empregar o conhecimento de uma equipe e fim de melhorar o recurso final a espera e integração do cliente.

De 11 a 13 de fevereiro de 2001, em *The Lodge* no hotel *Snowbird* de esqui nas montanhas *Wasatch* de *Utah*, dezessete pessoas se reuniram para tentar encontrar um fundamento em comum e definir o Manifesto Ágil, que tem como objetivo definir melhores maneiras para o desenvolvimento em entregas rápidas. Representantes de *Extreme Programming*, *Scrum*, *DSDM*, *Adaptive Software Development*, *Crystal*, *Feature-Driven Development*, *Pragmatic Programming*, e outros simpatizantes da necessidade de uma alternativa a documentação dirigida, e os processos de desenvolvimento de *software* pesados. (HIGHSMITH, 2001)

Dentre a pauta a ser tratada tiveram a petulância de sugerir um nome à inovação que surgiria para melhora do *software* mundial. Essa equipe definiu preceitos que se bem aplicados trazem uma melhora significativa à produção e utilização de recursos agregados ao software.

Nomeavam a si mesmos de "*The Agile Alliance*", este grupo de pensadores independentes sobre desenvolvimento de *software*, e às vezes concorrentes entre si, concordaram com a Manifesto para Desenvolvimento Ágil de *Software*. (HIGHSMITH, 2001)

O primordial dessa reunião era definir os princípios por muitos empregado com sucesso e bem valorizado como produto de inovação. Tem-se assim um pensamento simples que aplicado gera recursos não tão simples a inovação.

> Descobrir a melhor maneira de desenvolver *software*, fazendo-o de maneira que uns possam ajudar os outros a obter o melhor. Através deste trabalho, será possível valorizar:
>
> - Indivíduos e interações mais que processos e *frameworks*;
> - *Software* em funcionamento mais que documentação abrangente;
> - Colaboração com o cliente mais que negociação de contratos;

> - Responder a mudanças mais que seguir um plano.
>
> Ou seja, mesmo havendo valor nos itens à direita, valorizar mais os itens à esquerda. (CUNNINGHAM, 2001)

Dos antigos preceitos muitas vezes não totalmente empregados surgiram modelos e *frameworks*[1] que de muito auxiliam a nova era de desenvolvedores e usuários. Esses novos *frameworks* sintetizam simplicidade, mudança e portabilidade, fazendo com que não se perdesse a raiz da causa de um *software* de qualidade e um cliente bem satisfeito.

Metodologias tradicionais fixam-se em planejamento e controle. A execução de uma etapa deve garantir que seu *output* seja o *input* perfeito para a etapa seguinte. As metodologias ágeis aplicam-se mais em inspeções e adaptações, mantendo-se prontas para mudanças de direção ocasionais. Para isso, vale-se de pequenos ciclos de desenvolvimento incremental, que produzem *feedback* contínuo. (GONÇALVES, 2011)

Entre os principais *frameworks* aplicados e bem sucedidos esta *Scrum* e *Kanban*, que através da melhor escolha de seus artefatos obtêm-se boas práticas para os novos recursos ágeis pensados e idealizados a fim de um conceito de grande valia as partes envolvidas.

2.3 *FRAMEWORK LEAN*

Com relação ao desenvolvimento de produtos, o pensamento *Lean* é baseado em um extenso conjunto de princípios econômicos e matemáticos comprovados que descrevem o fluxo sobre as informações do produto dentro da empresa, mas que aplicam igualmente ao fornecedor e ao cliente elementos de grande valor a cadeia de negócios. Como tal, ele é mais amplo e profundo do que os alguns métodos específicos de desenvolvimento ágil tem descrito até agora. (LEFFINGWELL, 2010)

Dentre os principais princípios do pensamento enxuto, definidos por POPPENDIECK e POPPENDIECK (2003):

- Eliminar o desperdício;

[1] Segundo (FAYAD e SCHMIDT, 1997) um *framework* é um conjunto de classes que colaboram para realizar uma responsabilidade para um domínio de um subsistema da aplicação. (Framework, 2008)

- Ampliar o aprendizado;
- Decidir o mais tarde possível;
- Entregar o mais rápido possível;
- Capacitar à equipe;
- Construir com qualidade;
- Aperfeiçoar o todo.

2.3.1 Eliminando o desperdício

Desperdício parece ser um termo claro, porém OHNO (1997) deu um novo significado a palavra. Qualquer coisa que não gera valor para o cliente é desperdício. A parte que esta aguardando para ser usada é desperdício. Fazer alguma coisa que não será usado imediatamente é desperdício. Impulso é desperdício. Todas as tarefas extras são desperdício. E, é claro, os defeitos são desperdícios. (POPPENDIECK e POPPENDIECK, 2003)

A eliminação do desperdício está especificamente direcionada para reduzir custos pela redução da força de trabalho e dos estoques tornando clara a disponibilidade extra de instalações e de equipamentos, possibilitando diminuir gradualmente o desperdício secundário. (OHNO, 1997)

Ter um processo *just in time* significa reduzir desperdício fazendo somente o que é necessário, na quantidade necessária, no local necessário e quando é necessário. Em uma linha de produção, o fluxo *just in time* permite diminuir estoques e aumentar o giro de produtos. Associado a uma técnica de produção conhecida por sistema puxado, o *just in time* possibilita também minimizar as perdas com produção excessiva e consequentemente aumentar a produtividade da linha de produção. O *just in time* também pode ser aplicado em *software* de diversas maneiras. (CRESCÊNCIO, 2011)

2.3.2 Ampliar o aprendizado

Desenvolvimento de *software* é um exercício de descoberta, enquanto produção é um exercício de redução das variações, e por essa razão, que uma introdução *Lean* para as práticas de resultados no desenvolvimento são diferentes de práticas *Lean* na produção. Desenvolvimento é como criar uma receita, enquanto produção é como fazer o prato. Receitas são projetadas por experientes chefes que desenvolveram um instinto para que trabalho e os ingredientes disponíveis possam ser adaptados à ocasião. Entretanto mesmo grandes chefes produzem inúmeras variações de novos pratos como eles interagem com receitas saborosas e são fáceis de serem feitas. Chefes não esperam ter a receita perfeita na primeira tentativa, eles esperam produzir diversas variações de um tema como parte do processo de aprendizagem. Desenvolvimento de *software* é mais bem concebido como um processo de aprendizagem, como um desafio para a equipe de desenvolvimento, tendo assim grandes resultados que se tornam mais complexos do que uma simples receita. A melhor abordagem para a melhoria de um ambiente de desenvolvimento de *software* é ampliar o aprendizado. (POPPENDIECK e POPPENDIECK, 2003)

2.3.3 Decida o mais tarde possível

No desenvolvimento de *software Lean* são atrasadas todas as tomadas de decisão o maior tempo possível, porque é fácil mudar uma decisão que não foi feita. Desenvolvimento de *software Lean* enfatiza desenvolvimento robusto e tolerante a mudanças, que aceita todas as inevitáveis mudanças e estruturas do sistema para que possa ser facilmente adaptado aos mais prováveis tipos de mudança. (POPPENDIECK e POPPENDIECK, 2003)

Um sistema puxado de produção determina a carga de trabalho de acordo com o consumo do cliente. Se não houver consumo não haverá produção, eliminando completamente o desperdício com a produção excessiva. Diferentemente de um sistema empurrado, onde há produção independentemente da demanda, o sistema puxado gerencia toda a cadeia produtiva – desde a extração da matéria

prima até a transformação em um produto acabado. Para auxiliar neste processo, Taichi Ohno concebeu um *framework* chamado *Kanban*, que permite um gerenciamento visual e colaborativo da produção puxada. O *Kanban* tornou-se também um *framework* muito importante para gerenciar o desenvolvimento de sistemas complexos. Veremos mais adiante como aplicá-lo a software. (CRESCÊNCIO, 2011)

2.3.4 Entregar o mais rápido possível

Quando se fala em entregar o mais rápido possível, não significa que se deve correr rapidamente arriscando o estado real. Apenas é recomendado uma boa prática de produção. Entregas rápidas são práticas operacionais que promovem uma sólida vantagem competitiva. Clientes gostam de entregas rápidas tanto que uma vez que uma empresa em um negócio aprende como entregar rapidamente, seus concorrentes são forçados a seguir o exemplo. (POPPENDIECK e POPPENDIECK, 2003)

Uma equipe tem duas grandes razões para entregar rápido: para que o cliente não mude de ideia enquanto constrói-se e para que o concorrente não entregue antes. Em geral, as mudanças no mercado e a velocidade com que o concorrente consegue entregar irão determinar o quão rápido se precisa ser na hora da entrega. Além disto, entregar rápido e de forma incremental permite *feedback* e aprendizado sobre o que a equipe está fazendo. As experimentações do produto no mercado, ainda que inacabado, proporcionarão um desenvolvimento muito mais assertivo. (CRESCÊNCIO, 2011)

2.3.5 Capacitar à equipe

Excelente capacidade de execução encontra-se em obter os detalhes certos, e ninguém entende melhor os detalhes do que as pessoas que realmente fazem o trabalho. Envolver os desenvolvedores nos detalhes das decisões técnicas é

fundamental para alcançar a excelência. As pessoas na linha de frente combinam o conhecimento dos detalhes minuciosos com o poder de muitas mentes. Quando equipada com a competência necessária e guiada por um líder, a equipe irá fazer as melhores decisões técnicas e as melhores decisões do processo do que qualquer um pode fazer por eles. Porque a decisão é feita tarde e a execução é rápida, isso não é possível a uma autoridade central definir as atividades dos trabalhadores. Assim, as práticas *Lean* usam técnicas puxadas para planejar o trabalho e conter mecanismos para sinalizar aos trabalhadores o que deixou de ser feito. No desenvolvimento de *software Lean*, que é um acordo para entregar cada vez mais versões refinadas de *software* trabalhado em intervalos regulares. Alguns sinais ocorridos são visíveis em gráficos, reuniões diárias, e testes abrangentes. (POPPENDIECK e POPPENDIECK, 2003)

O principal objetivo do trabalho conjunto e integrado de uma equipe *Lean* é identificar com assertividade as necessidades de negócio e promover a entrega incremental de valor efetivo. É fundamental uma equipe saber identificar o que é valor e qual a porção mínima de *software* necessária para entregar tal valor. Por isso, cita-se propositadamente o termo entrega de valor, em vez de entrega de *software*. (CRESCÊNCIO, 2011)

2.3.6 Construa com qualidade (embutida)

É percebido que um sistema começa a ter integridade quando o usuário pensa, “Sim! Era exatamente isso que eu queria. Alguém leu minha mente!” O termo “Fatia de mercado” é uma medida aproximada de perceber a integridade dos produtos, porque mede a percepção do cliente ao longo do tempo. Integridade conceitual significa que aos conceitos centrais do sistema trabalham juntos como um todo, coeso, e é um fator critico na criação da integridade percebida. *Software* precisa manter um nível de integridade adicional – ele deve manter a sua utilidade ao longo do tempo. *Software* geralmente é esperado para evoluir graciosamente e se adaptar ao futuro. *Software* com integridade possui uma arquitetura coerente, vários pontos em usabilidade e adaptação a um propósito, e é de fácil manutenção, adaptável e extensível. O estudo científico mostrou que a integridade vem da sábia

liderança, conhecimentos especializados, comunicação eficaz, e disciplina saudável; processos, procedimentos e medidas não são substitutos adequados. (POPPENDIECK e POPPENDIECK, 2003)

2.3.7 Aperfeiçoe o todo

Um sistema consiste de partes independentes que interagem juntas em um único propósito. Um sistema não é apenas a soma das partes – é o produto de suas interações. A melhor parte não necessariamente faz o sistema, a capacidade do sistema de alcançar seu objetivo depende de quão bem suas partes trabalham juntas, não o que representam individualmente. (POPPENDIECK e POPPENDIECK, 2003)

No conceito “A Pirâmide *Lean*”, termo empregado por CRESCÊNCIO (2011) para definir como o pensamento *Lean* engloba os princípios e valores dos métodos de desenvolvimento, as pessoas precisam enxergar e compreender o todo. Mais do que isto, precisam contribuir para melhoria da organização como um todo. Este princípio aplica-se também na parte técnica. Se você tem especialistas em certas partes do código e só eles conseguem mexer lá, certamente as demais pessoas da equipe não estão vendo o todo.

2.4 *SCRUM*

Para demonstrar um dos *frameworks* mais utilizados muitas vezes individualmente ou consorciado a outros, tendo em vista à mudança e adaptação a estrutura de trabalho, deu-se o *framework Scrum*[2] uma visão privilegiada da equipe, por assim dizer uma boa escolha para iniciação ou permanência continua com o emprego simples e reduzido.

[2] Ken Schwaber utiliza o termo *framework* para informar que *Scrum* é mais que uma metodologia.

O *Scrum* é focado nas práticas de gerenciamento e organização, enquanto o XP (*eXtreme Programming*) dá mais atenção às tarefas de programação mesmo. Nesse caso obtemos o porquê delas trabalharem bem juntas – elas abrangem áreas diferentes e uma complementa a outra. (KNIBERG, 2007)

A aplicação dos métodos ágeis traz uma gama muito grande de artifícios que podem ser utilizados, porém não obriga nenhuma equipe a segui-los veementemente, apenas os sugere como um mecanismo de melhoramento e integração. *Scrum* define vários papéis que podem ser aplicados assim como tarefas que podem ou não ser utilizadas. O importante é manter uma equipe engajada, a fim de manter o foco do projeto.

Scrum é um *framework* para desenvolver e manter produtos complexos. Esta definição consiste em papeis, eventos, artefatos do *Scrum* e as regras que mantém tudo integrado. Ken Schwaber e Jeff Sutherland desenvolveram o *Scrum*, e escreveram o Guia do *Scrum*. (SUTHERLAND e SCHWABER, 2011)

Scrum é um *framework* baseada em simplicidade e adaptabilidade. Um fundamento dessa filosofia é a manutenção da base desse conceito: equipes gerenciáveis. Várias literaturas apontam equipes de no máximo nove pessoas, multidisciplinar e motivada. Mas é inevitável a existência de projetos onde essa força de trabalho tem de ser aumentada, devido ao tamanho do sistema sendo desenvolvido. (GONÇALVES, 2011)

Para o bom funcionamento de *Scrum* as tarefas executadas devem seguir determinado cronograma, independente de serem executadas completamente ou não. Com isso são obtidos os *sprints* previamente definidos, onde são executados pela equipe, em determinado fluxo de tempo para cada conjunto de funcionalidades escolhidas para aquele *sprint*.

As funcionalidades (*user stories*) que devem ser desenvolvidas compõem uma lista denominada product backlog. Esta lista não precisa estar completa antes do começo do desenvolvimento, ela cresce e se modifica ao longo do processo. Antes do início do *sprint*, o *product backlog* é avaliado para gerar o *selected backlog*, lista das funcionalidades que serão desenvolvidas naquele *sprint*. As funcionalidades do *selected backlog* são então quebradas em tarefas gerando o *sprint backlog*. (GONÇALVES, 2011)

Tudo que acontece sob *Scrum* é realizado em um período fechado de tempo. Essa característica é denominada *time boxing*. As reuniões e outros eventos têm

todos os tempos definidos e fechados. Essa premissa é fundamental para quem deseja entregar *software* no menor tempo possível. A FIGURA 2 apresenta a mecânica de um *sprint* completo. (GONÇALVES, 2011)

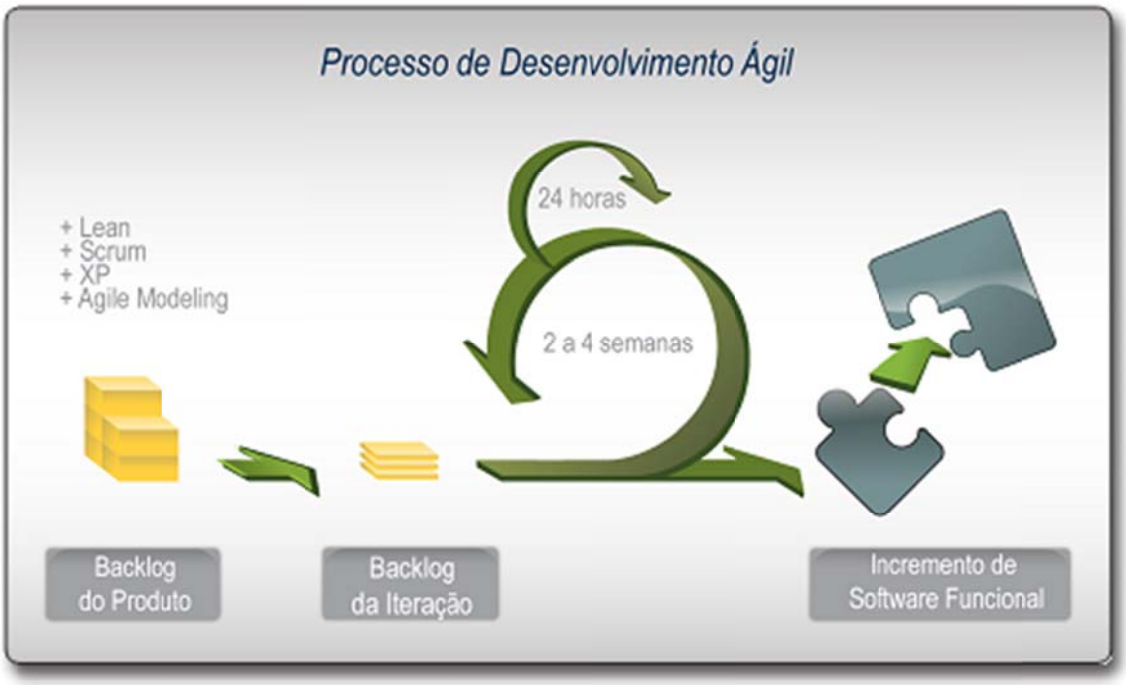

FIGURA 2 – Processo de desenvolvimento ágil.
FONTE: (CRESCÊNCIO, 2011)

2.4.1 Entendendo *Scrum*

Na visão *Scrum* são prescritos papéis e ações com sugestão para uma equipe iniciante identificar e tentar emprega-los de forma completa ou simplificada. Essa base ajuda a prever o que de melhor pode ser empregado nas funcionalidades a serem geradas pela equipe ao *software* final. Tanto os papéis como as métricas são optativas, as mesmas apenas visam a melhor aderência de um projeto ágil.

Scrum prescreve 3 papéis: *Product Owner* (cria a visão do produto e prioridades), *Team* (implementa o produto) e *Scrum Master* (remove impedimentos e fornece liderança de processo). (KNIBERG e SKARIN, 2009)

Em *Scrum*, as estimativas são feitas utilizando métricas aderentes à filosofia de trabalho do *Scrum*: agilidade. Para tanto, o método se vale de uma técnica

intitulada *history points*, uma contagem de pontos que define um tamanho para cada estória de maneira relativa às outras estórias de mesmo projeto. É, portanto, uma medida do tamanho geral de uma atividade, não absoluta. (GONÇALVES, 2011)

O *product backlog* é o coração do *Scrum*. É aqui que tudo começa. O *product backlog* é basicamente uma lista de requisitos, estórias, Coisas que o cliente deseja, descritas utilizando a terminologia do cliente. (KNIBERG, 2007)

Em *Scrum*, o *sprint backlog* mostra quais são as tarefas a serem executadas durante a iteração corrente (= "*sprint*" na linguagem *Scrum*). Geralmente ele é representado usando cartões na parede ou quadro branco, conhecido também por quadro do *Scrum* ou quadro de tarefas. (KNIBERG e SKARIN, 2009)

Para a equipe ou até mesmo para as pessoas externas ao meio é exibido, caso a equipe permita, seu quadro de tarefas que supõe a evolução do projeto e das tarefas a serem executadas, caso contrario são exibidos apenas os resultados de toda a evolução as partes envolvidas. Em alguns casos são obtidos fluxos externos as tarefas executadas que devem ser tratados, e depois de sanados devem ser revistos para que situações imprevistas não ocorram mais, que podem gerar atrasos indesejados. Com a manutenção desses artefatos, pode ser mantido um gráfico que ilustra o processo de evolução da equipe. Na FIGURA 3 verifica-se como podem ser mantidas essas interações realizadas pela equipe juntamente com os itens não planejados que podem atrasar os projetos se não tratados corretamente.

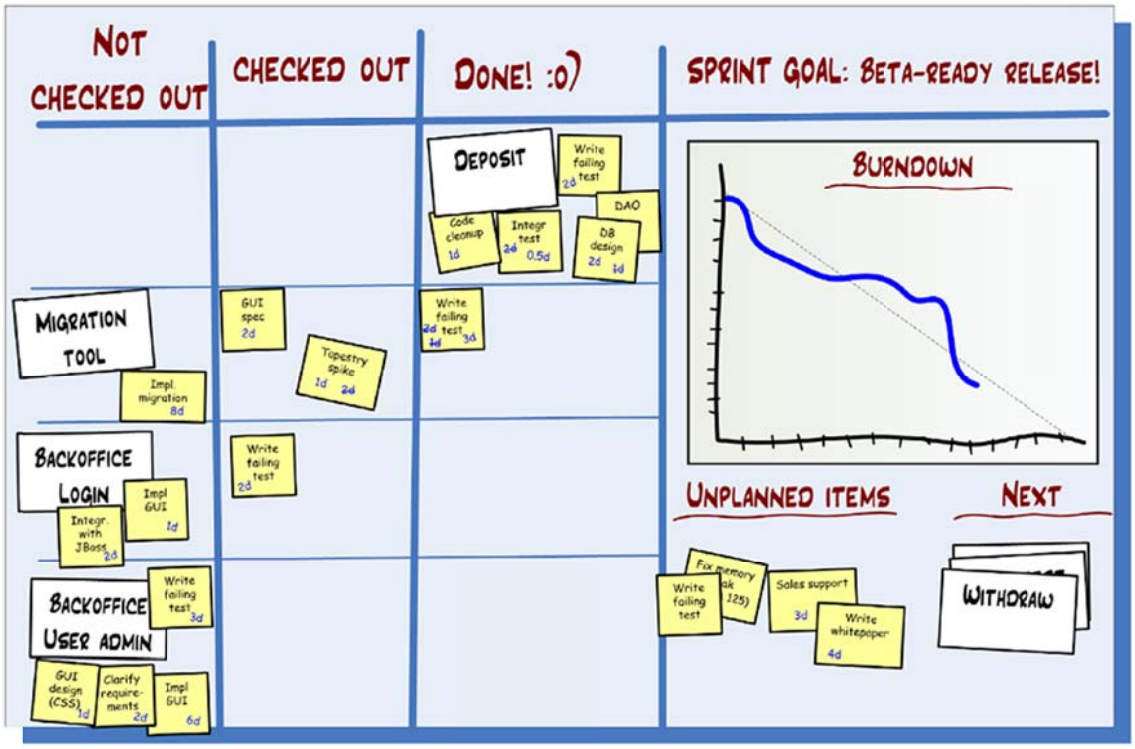

FIGURA 3 - Quadro de tarefas.
FONTE: (KNIBERG, 2007)

Possuir algum tipo de cronograma para a reunião de planejamento do *sprint* reduzirá o risco de ultrapassar o espaço de tempo previsto. (KNIBERG, 2007)

Scrum é baseado em iterações de tempo fixo. A equipe pode escolher a duração da iteração, mas a ideia geral é manter a mesma duração de iteração por um período de tempo, desta forma estabelecendo uma cadência. (KNIBERG e SKARIN, 2009)

Quando o *sprint* acaba, o quadro é limpo – todos os itens são removidos. O *sprint* novo é iniciado e depois da reunião de planejamento do *sprint*, tem-se um novo quadro do *Scrum*, com novos itens na coluna mais à esquerda. (KNIBERG e SKARIN, 2009)

Devido às perspectivas não serem obrigatórias à equipe pode optar por não seguir as métricas de estórias definidas, para isso apenas dividem as tarefas em processos menores que vão sendo executados e revisados a cada fluxo de tempo.

Algumas equipes optam por fazer estimativas e medir a velocidade, outras equipes escolhem pular as estimativas, mas tentam quebrar cada item em itens menores de aproximadamente do mesmo tamanho – desta forma eles podem medir a velocidade simplesmente em termos de quantos itens foram concluídos por

unidade de tempo (por exemplo, requisitos ou funcionalidades por semana). (KNIBERG e SKARIN, 2009)

Para obter um controle sobre os processos e fluxos algumas equipes optam por demonstra-los em quadros visíveis a todos, e manter um controle sobre o fluxo de trabalho total comparado com o que já foi executado através de gráficos.

Alguns dos artefatos do *Scrum* permitem visualizar melhor o fluxo de trabalho, são eles:

- Dashboard;
- Gráfico de *Burndown*.

2.4.2 *Dashboard*

Um quadro de *Scrum* pertence a exatamente uma equipe *Scrum*. Uma equipe *Scrum* é multifuncional, ela contém todas as habilidades necessárias para completar todos os itens contidos na iteração. Um quadro de *Scrum* é geralmente visível a qualquer pessoa que esteja interessada, mas somente aqueles que fazem parte da equipe *Scrum* podem editá-lo - é o seu *framework* para gerenciar o seu compromisso para esta iteração. (KNIBERG e SKARIN, 2009)

O painel de controle do projeto é exclusivamente alterado pela equipe que o mantém, ou seja, toda e qualquer ação executada deve gerar uma ação sobre o quadro. Assim como toda tarefa indesejada deve ser adicionada pelo *Scrum Master*, para que o mesmo possa pensar e juntamente com a equipe realizar ações imprescindíveis para a prescrição do *sprint*. Na FIGURA 4 é possível verificar as tarefas não executadas que estão no aguardo para iniciação, e as que foram iniciadas, estando elas prontas ou não.

Em *Scrum*, o *sprint backlog* é apenas uma parte de algo maior - a parte que mostra o que a equipe está fazendo durante o *sprint* atual. A outra parte é o *product backlog* - a lista de coisas que o *Product Owner* quer que sejam feitas nos *sprints* futuros. (KNIBERG e SKARIN, 2009)

FIGURA 4 - *Sprint backlog* e *product backlog*.
FONTE: (KNIBERG e SKARIN, 2009)

2.4.3 Gráfico de *Burndown*

Um gráfico de *Burndown* mostra a quantidade de trabalho restante ao longo do tempo. O gráfico de *Burndown* é uma excelente forma de visualização da correlação entre a quantidade de trabalho restante em qualquer ponto no tempo e o andamento da equipe de projeto(s) em reduzir esse trabalho. A intersecção de uma linha de tendência para o trabalho restante e o eixo horizontal indica a conclusão mais provável de trabalho naquele momento. Um gráfico de *Burndown* refletindo isso é mostrado na FIGURA 5. Isso permite ao projeto, adicionar e remover funcionalidades da versão para conseguir uma data mais aceitável ou prorrogar a data para incluir mais funcionalidade. O gráfico de *Burndown* é a colisão da realidade (trabalho feito e quão rápido ele está sendo feito), com o que está planejado ou esperado. (SCHWABER, 2004)

Em um gráfico mantido corretamente são tratados vários problemas, e podem ser calculadas varias ações de contorno, imprescindíveis ao bom funcionamento da dinâmica do grupo.

No exemplo da FIGURA 5 é demonstrado como é mantido o gráfico de *burndown*, assim no primeiro dia do *sprint*, 1 de agosto, a equipe estimou que havia aproximadamente 70 pontos por estória de trabalho a serem feitos. Isso foi na verdade a velocidade estimada de todo o *sprint*. Em 16 de agosto a equipe estimou aproximadamente 15 pontos por estória de trabalho a serem feitos. A linha de tendência tracejada mostra que eles estão aproximadamente dentro do prazo, isto é,

neste ritmo eles completarão tudo até o final do *sprint*. Algumas equipes também usam gráficos de *burndown* de release, que segue o mesmo formato, mas em nível de release - ele normalmente mostra quantos pontos de história restam no *product backlog* depois de cada *sprint*. A elaboração e manutenção deste gráfico é de responsabilidade da equipe. (KNIBERG, 2007)

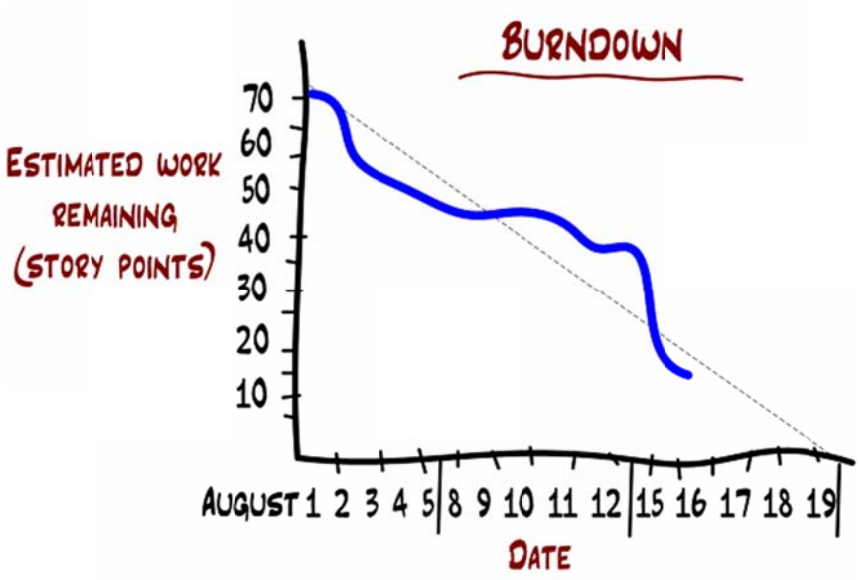

FIGURA 5 - Gráfico de *burndown*.
FONTE: (KNIBERG, 2007)

A unidade do eixo Y é a mesma unidade utilizada nas tarefas de *sprint*. Normalmente horas ou dias (se a equipe divide itens de *backlog* em tarefas) ou pontos de história (se a equipe não os divide). Entretanto, há muitas variações disso. (KNIBERG e SKARIN, 2009)

Tudo o que for gerado pela equipe deve constar no gráfico. Nos finais de cada *sprint* ele pode ser levado e apresentado ao cliente como justificativa de sucesso ou de não entrega de itens por questões não calculadas tanto pela equipe quanto pelas iterações com o cliente. Com tudo calculado fica fácil de disponibilizar resumos, ou de prever inconsistências.

Na Reunião de Planejamento do *sprint* a Equipe de *Scrum* identifica e estima as tarefas específicas que devem ser concluídas para o *sprint* ser bem sucedido. O total de todas as estimativas do *sprint backlog* de trabalho restante a ser preenchido é o atraso acumulado. Quando as tarefas do *sprint* forem concluídas, o *Scrum Master* recalcula o trabalho restante a ser feito e o sprint *backlog* diminui, e o gráfico

diminui ao longo do tempo. O *sprint backlog* cumulativo é zero no final do *sprint*, e o *sprint* é bem sucedido. (SUTHERLAND e SCHWABER, 2007)

Conforme é atualizado o gráfico pode-se verificar que tarefas indicadas pela equipe com certo grau de dificuldade apresentam restrições que podem gerar atrasos indesejados ao projeto. É necessário que esses desvios não resultem em um acúmulo de tarefas que pode incapacitar o término do *sprint*.

2.4.4 Tratar Desvios Rapidamente

O *scrum master* é responsável por garantir que a equipe haja quando surgirem sinais de alarme. (KNIBERG, 2007)

Durante a manutenção diária das tarefas o *scrum master* atualiza o gráfico com as iterações, é possível que durante esse processo seja verificado desvios na linha da evolução da equipe. Quando os desvios sugerem que muitas tarefas não estão sendo finalizadas conforme o tempo estipulado, provavelmente algumas estórias não tenham ganhado o devido valor, que gerou um tempo extra para a conclusão das mesmas. O desvio aparente é apresentado na FIGURA 6.

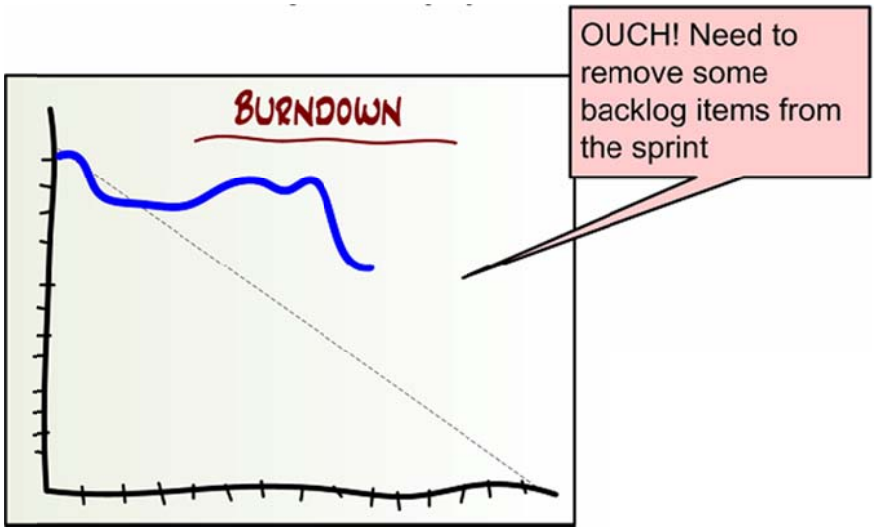

FIGURA 6 - Desvio indicando que as tarefas devem ser reduzidas.
FONTE: (KNIBERG, 2007)

As tarefas também podem ser superestimadas o que gera outro desvio perceptível, que também deve ser tratado, como é o caso da FIGURA 7.

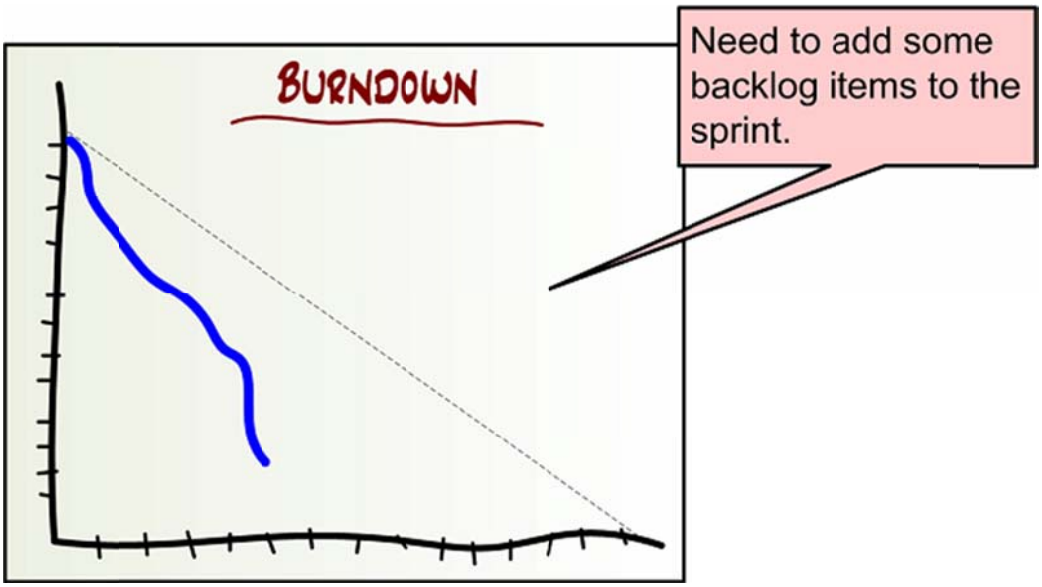

FIGURA 7 - Desvio indicando que deve ser aumentado o número de tarefas.
FONTE: (KNIBERG, 2007)

Além das tarefas superestimadas podem surgir tarefas indesejadas ou não planejadas que limitam a equipe na conclusão precisa dos itens contidos nas estórias. Esse desvio pode ser observado no quadro de tarefas do *sprint*, onde os itens são adicionados como extras ao lado do gráfico, como observado na FIGURA 8.

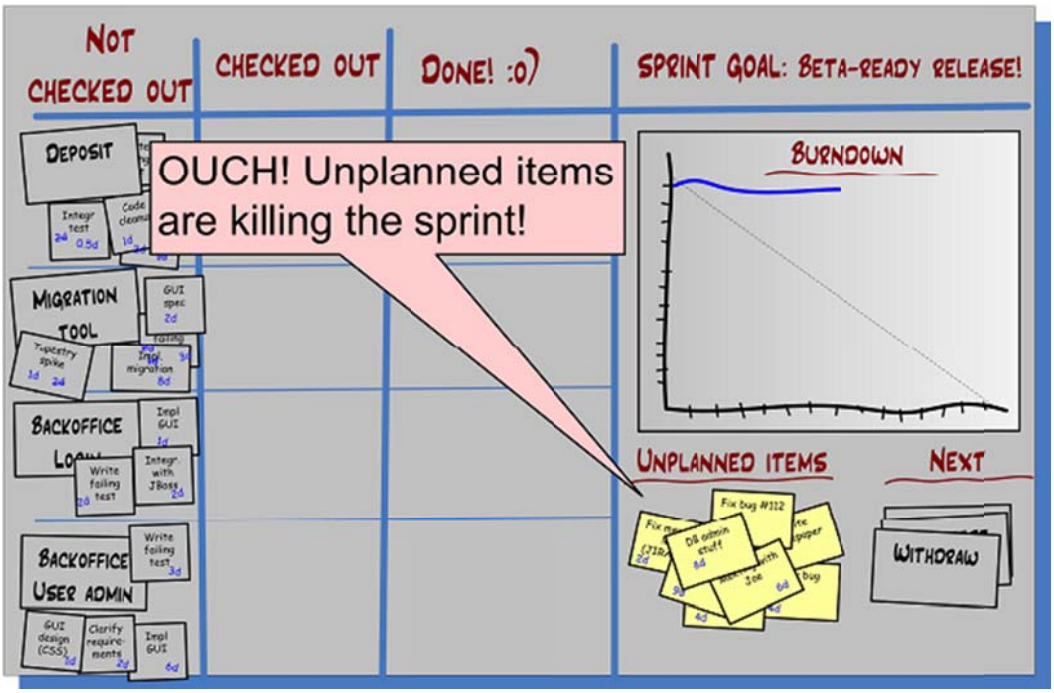

FIGURA 8 - Desvio para aumento indesejado das tarefas não planejadas.
FONTE: (KNIBERG, 2007)

Com os devidos tratamentos uma equipe consciente consegue sobrepor as barreiras facilmente dando continuidade ao projeto, entrega consciente e finalização do *sprint*.

2.5 *FRAMEWORK KANBAN*

Scrum e *Kanban* são *frameworks* que, em certa medida, ajudam a trabalhar de maneira mais eficaz, dizendo a equipe o que fazer. Java também é uma ferramenta, ela fornece uma maneira mais simples de programar. (KNIBERG e SKARIN, 2009)

O *Kanban* é um *framework* para conseguir o *just-in-time*. Para que esse *framework* funcione relativamente bem, os processos de produção devem ser administrados de forma a fluírem tanto quanto possível. Esta é realmente a condição básica. Outras condições importantes são nivelar a produção tanto quanto possível, e trabalhar sempre de acordo com métodos padronizados de trabalho.

Kanban é utilizado como *framework* por várias equipes que possuem um conhecimento grande sobre o modelo ágil de desenvolvimento, que buscam assim apenas as poucas restrições oferecidas pela mesma, e agregam várias outras atividades relacionadas com outros *frameworks* com intuito de melhorar o desempenho do software. Como o *framework Kanban* é simplificado ele possibilita a adição de recursos, ele geralmente é bem visto para que uma equipe engajada consiga alcançar os objetivos desejados.

De forma simplificada, o *Kanban* é um processo de produção puxado que mapeia as etapas de desenvolvimento. Para cada etapa identificada, ele estabelece limites para a quantidade de trabalho sendo realizada simultaneamente. Os limites superiores auxiliam a minimizar a multitarefa, neste caso nocivo à produtividade da equipe. Limites inferiores vão auxiliar a garantir que sempre haja demanda suficiente para que o trabalho não pare. (CRESCÊNCIO, 2011)

2.5.1 Entendendo *Kanban*

O sistema *Kanban* é praticado sob regras rígidas e progride pela supervisão minuciosa e constante dessas regras como num problema sem fim. As regras definidas por OHNO (1997) para utilização são as seguintes:

- O processo subsequente apanha o número de itens indicados pelo *Kanban* no processo precedente;
- O processo inicial produz itens na quantidade e sequencia indicadas pelo *Kanban*;
- Nenhum item é produzido ou transportado sem um *Kanban*;
- Serve para afixar um Kanban as mercadorias;
- Produtos defeituosos não são enviados para o processo seguinte. O resultado é mercadorias 100% livres de defeitos;
- Reduzir o numero de *Kanbans* aumenta sua sensibilidade aos problemas. (OHNO, 1997)

Para a primeira regra do *Kanban*, define-se que em um projeto em desenvolvimento a duração das iterações é estipulada pela equipe, que também pode planejar melhor o processo de entregas, e qual será a periodicidade dessas atividades (“*release* toda segunda”), ou por demanda (“*release* sempre que tivermos algo útil a entregar”). (KNIBERG e SKARIN, 2009)

O objetivo de eliminar desperdício também é enfatizado pelo *Kanban*. Sua utilização mostra imediatamente o que é desperdício, permitindo um estudo criativo e propostas de melhorias. Na planta de produção, o *Kanban* é uma força poderosa para reduzir mão-de-obra e estoques, eliminar produtos defeituosos, e impedir a recorrência de panes. (OHNO, 1997)

Para fazer a segunda regra do *Kanban* funcionar (fazer com que o processo introdutivo produza apenas a quantidade retirada pelo processo subsequente, a força de trabalho e o equipamento em cada processo de produção deve estar preparado, em todos os aspectos, para produzir as quantidades necessárias no momento necessário. (OHNO, 1997)

Sob a sua primeira e segunda regra, o *Kanban* serve como um pedido de retirada, um pedido de transporte ou entrega, e como uma ordem de fabricação. A regra três do *Kanban* proíbe que se retire qualquer material ou que se produza qualquer mercadoria sem um *Kanban*. A regra quatro requer que um *Kanban* seja afixado às mercadorias. A de número cinco exige produtos 100% livres de defeitos (ou seja, não envie peças defeituosas para o processo subsequente). A regra seis pede a redução do número de *Kanbans*. Quando essas regras são fielmente praticadas, o papel do *Kanban* se expande. (OHNO, 1997)

Introduzir o *Kanban* sem efetivamente praticar essas regras, não trará nem o controle esperado do *Kanban* nem a redução dos custos. Assim, uma introdução parcial do *Kanban* traz uma centena de malefícios, mas nem um ganho sequer. Qualquer um que reconheça a efetividade do *Kanban* como um *framework* de gestão da produção para reduzir custos deve estar determinado a observar as regras e a superar todos os obstáculos. (OHNO, 1997)

Algumas equipes optam por fazer estimativas e medir a velocidade assim como no *Scrum*. Outras equipes escolhem pular as estimativas, mas tentam quebrar cada item em itens menores de aproximadamente do mesmo tamanho - desta forma eles podem medir a velocidade simplesmente em termos de quantos itens foram concluídos por unidade de tempo (por exemplo, requisitos ou funcionalidades por semana). (KNIBERG e SKARIN, 2009)

Como o *Kanban* não prescreve artefatos específicos para tratamento do fluxo de trabalho, muitas equipes optam por utilizar alguns artefatos do *Scrum*, são eles:

- Dashboard;
- Gráfico de *Burndown.*

2.5.2 Dashboard

Tanto *Scrum* quanto *Kanban* limitam as atividades em andamento, mas de formas diferentes. Equipes *Scrum* geralmente medem a velocidade - quantos itens (ou unidades de medidas correspondentes como *history points*) serão feitos por iteração. Desta forma, em *Scrum* as atividades em andamento são limitadas por unidade de tempo. Em *Kanban* as atividades em andamento são limitadas pelo fluxo de trabalho. (KNIBERG e SKARIN, 2009)

Limitar as tarefas em andamento é de extrema importância para que a equipe consiga prescrever ações de correção caso algum item não planejado apareça durante a execução do projeto. Como *Kanban* não prescreve o gráfico para controle de desvios encontrados na execução das tarefas, manter um controle das atividades é necessário quando se tem itens não prescritos.

Uma vez que a equipe definiu devidamente os limites das atividades em andamento, pode-se começar a medi-los e prever o tempo de execução do ciclo, isto é, o tempo médio que cada item leva para cumprir todo o ciclo através do quadro. (KNIBERG e SKARIN, 2009)

2.5.3 Gráficos de *Burndown*

Em *Kanban*, gráficos de *burndown* não são definidos. Na verdade, não é prescrito nenhum gráfico em especial. Mas a equipe está, logicamente, autorizada a usar qualquer tipo de gráfico que quiser (incluindo *burndowns*). (KNIBERG e SKARIN, 2009)

Como *Kanban* é um *framework* que permite várias escolhas perante a equipe que o emprega, os gráficos muito utilizados pelas equipes que optam por *Scrum*, não são prescritos, porém inúmeras equipes optam por adicioná-lo as iterações devido aos mesmos trazerem bons resultados.

Kanban deixa muito aberto quanto à escolha dos artefatos. As únicas restrições são: Visualize Seu Fluxo de Trabalho e Limite Suas Atividades em

Andamento. Apenas a alguns centímetros de Faça Qualquer Coisa, mas ainda assim surpreendentemente poderoso. (KNIBERG e SKARIN, 2009)

3 PROCEDIMENTOS METODOLÓGICOS DA PESQUISA

Será demonstrado através de um estudo experimental, onde as características encontradas nos frameworks *Scrum* e *Kanban* podem ser usadas para redução das perdas em um projeto em execução simples, promovendo assim os ganhos esperados para a aplicação do pensamento *Lean*.

O estudo experimental fictício proposto por DIEDRICH (2011) vem demonstrar como aplicando os recursos necessários e uma boa gestão pode-se ter um produto de qualidade. Este estudo experimental tem como base a confecção de pedidos com gerenciamento de faturas e manutenção dos produtos utilizados para isso, e disponibilizá-los por meio de um catálogo aos clientes.

3.1 ESTUDO EXPERIMENTAL

Priorizando apenas a análise do pensamento utilizado para presumir possíveis fontes de desperdício, o estudo experimental proposto por DIEDRICH (2011) serve como embasamento simples a ser aproveitado no estudo das características que contribuem ao desenvolvimento ágil. Nele podem ser identificados vários princípios de perda da composição das entregas, que se não tratados durante o processo podem gerar gastos desnecessários.

O estudo experimental de DIEDRICH (2011) propõe o controle de faturas dos clientes que devem ser pagas dentro do vencimento, e faturas dos fornecedores, que devem manter o cadastro de produtos para que o sistema possa gerar um catálogo de produtos aos clientes, isso dentro de uma distribuidora de produtos fictícia denominada *DistriMais*.

No estudo experimental de DIEDRICH (2011) houve um total planejamento sobre as ações a serem tomadas seguindo os artefatos das metodologias ágeis e metas definidas pelo cliente para a entrega da solução. Com esse gerenciamento a equipe manteve o foco em manter a organização e as boas práticas necessárias. Cada iteração foi registrada e mantida em gráficos e quadro de tarefas, mantendo um ciclo definido e sequencial para o projeto.

3.1.1 Casos de uso

Para o estudo experimental hipotético utilizado foi necessária uma equipe pequena devido ao pequeno número de requisitos, no problema definido originalmente por apenas nove casos de uso. Conforme priorizados pela própria equipe os casos de uso foram definidos e podem ser visualizados no QUADRO 1.

Nr.	Descrição do Caso de Uso	Entrada	Caso de Uso	Resposta
01	Cliente envia solicitação de pedido	Dados Solicitação Pedido	RegistrarPedido	Msg01
02	Cliente efetua cadastro	Dados Cliente	CadastrarCliente	Msg02
03	Distribuidora requisita produtos de fornecedores	Dados Requisição Produtos	RequisitarProdutos	Msg03
04	Distribuidora atende pedido	Dados Pedido Atendido	AtenderPedido	Msg04
05	Cliente efetua pagamento	Dados Pagamento	EfetuarPagamento	Msg05
06	Distribuidora mantém produtos	Dados Produto	ManterProduto	Msg06
07	Distribuidora envia catálogo de produtos	Dados Catalogo	EnviarCatalogoProdutos	Msg07
08	Distribuidora lança fatura	Dados Fatura	LancarFatura	Msg08
09	Distribuidora confere produtos	Dados Entrega	ConferirProdutosEntrega	

QUADRO 1 - Lista de casos de uso.
FONTE: (DIEDRICH, 2011)

3.1.2 A Equipe e o tempo

Para o desenvolvimento do estudo experimental proposto foi utilizada uma equipe composta por quatro integrantes que estão dispostos entre desenvolvedores e líder da equipe, juntamente com outros quatro *stakeholders* que irão receber e validar a parte desenvolvida do projeto.

Conforme o trabalho original verificou-se que o tempo necessário para conclusão da lista de requisitos foi de quatro meses para o desenvolvimento e mais quatro meses e meio para a validação de todas as funcionalidades requisitadas e documentar as funcionalidades e telas necessárias para o bom entendimento do

sistema perante os usuários, definidos no estudo experimental por DIEDRICH (2011).

Com a equipe embasada nos casos de uso propostos, a mesma estimou o esforço necessário e o tempo de cada requisito envolvido. Desta maneira o tempo estimado foi cumprido normalmente não surgindo nenhum empecilho que pudesse interromper o desenvolvimento. O tempo utilizado para validação e documentação também não fugiu às estimativas iniciais propostas nas reuniões iniciais do projeto.

3.2 ESTIMAR O ESTUDO EXPERIMENTAL

Conforme o trabalho original descrito por DIEDRICH (2011) para definição dos artefatos do *Scrum* foi utilizada uma prática bem comum, que também é utilizada nos projetos *Kanban*, os *history points*. A técnica de estimar os casos de uso em estórias pequenas, fáceis de entender e de programar, e de conhecimento de todos os membros da equipe, conforme o seu nível de complexidade, importância, relevância e estimativa em horas trabalhadas.

No trabalho original quando aplicou-se *Scrum* e *Kanban*, definiu-se como será seguido com o trabalho, definiu-se as estórias, *backlogs* por *sprint* e mantido sempre atualizado o quadro de tarefas, tem-se um principio *Lean* de sistema*,* ou seja, a equipe irá se comprometer com quanto trabalho será entregue e quando. Desta maneira a equipe torna-se sustentável a ponto de prever possíveis desvios no seu quadro de tarefas.

3.3 DEFINIR MELHORIAS

No estudo experimental proposto por DIEDRICH (2011) priorizam-se as tarefas definidas pela equipe fictícia e o gráfico de *burndown* atualizado durante a execução das mesmas, e fazendo assim uma breve análise das ações de contorno de desvios que puderam ser aplicadas sobre os itens não planejados.

Das estórias verificadas foram recuperadas as priorizações e estimativas definidas para cada membro da equipe que iria executar a mesma. Também foi recuperado o gráfico traçado com a evolução da execução de cada tarefa em especifico, desde sua iniciação até a sua conclusão definitiva.

A descrição do caso de uso por completo foi utilizado para recuperar recursos não planejados que forçaram um desvio no gráfico de tarefas, podendo assim a equipe deixar itens menos importantes para a finalização de um entrega de lado, não entregando assim os mesmos.

Com o desenvolvimento do projeto e o surgimento de itens indesejados a equipe se reúne para redefinir prioridades, e resolver os problemas encontrados. Após a transposição dos obstáculos a equipe se torna mais capacitada e perceptível à resolução dos novos problemas.

Ao redefinir as estórias do projeto a equipe também opta por desqualificar itens que não trarão o resultado desejado ou que tem pouca importância. Assim as novas estórias surgem com melhorias necessárias ao projeto, que são problemas não levantados nas fases iniciais, porém de grande relevância ao produto.

3.4 CONFECÇÃO DO PROJETO

Para a confecção do estudo experimental do trabalho original foram utilizados vários artefatos disponibilizados pelos *frameworks Scrum* e *Kanban* que podem ser utilizados em conjunto para um melhor desempenho de uma equipe experiente, ou para uma equipe não muito experiente em desenvolvimento ágil pode-se escolher apenas uma lista de artefatos que podem ser modificados conforme a equipe vai ganhando confiança no trabalho realizado.

Como alguns artefatos não possuem muita diferença entre um framework e outro, com ressalva os que não são prescritos para um ou para outro, os que são similares podem ser empregados similarmente, obtendo sempre um bom resultado. Para isso o estudo experimental do trabalho original não se restringe a seguir plenamente um framework específico, mas sim a obtenção de uma lista de artefatos conjugados.

4 RESULTADO E DISCUSSÃO

Foi determinada uma lista de artefatos em específico para assim obter um maior controle sobre o desperdício obtido em um projeto mal planejado, ou reduzir ainda mais os projetos em execução que possuem boa gerência, porém assim mesmo não obtiveram os resultados esperados do emprego de métodos ágeis.

4.1 CARACTERISTICAS ENCONTRADAS

Dentre umas das importantes premissas definidas para o *Lean*, está um sistema *Just In Time*, que significa manter uma maior gestão do trabalho a ser aplicado em relação ao tempo proposto. Desta maneira a equipe escolhe quanto trabalho irá desempenhar, ou no caso compor o tempo proposto para entrega de determinadas tarefas.

Para defender o pensamento *Lean* serão utilizadas as características dos próprios *frameworks* enquadrados nos seguintes preceitos:

- Entregas rápidas;
- Decidir o mais tarde possível;
- Eliminar o desperdício;
- Capacite a equipe para construir com qualidade.

4.1.1 Entregas rápidas

É encontrada também a entrega contínua de resultado, que é bastante respeitada pelos frameworks *Scrum* e *Kanban*. Quando o sistema possui entregas rápidas, as concorrências são facilmente dribladas a fim de obter a garantia de um cliente satisfeito e participante.

As prioridades são de grande valia para a gerência da equipe, com uma boa definição das mesmas a equipe não terá surpresas no final de cada ciclo de entregas definidas. Este ciclo de iterações com o usuário também é definido pela equipe e varia conforme o tamanho de itens necessários para cada entrega e as pessoas envolvidas no processo. A lista de entregas e objetivos pode ser visualizada no QUADRO 2.

Iteração	Objetivo Primários	Data de Entrega	Velocidade
I1	Definição de Arquitetura base	15/05/2011 – 25/05/2011	15
I2	Entrega do Caso de Uso 01	26/06 – 10/07	16
I3	Entrega do Caso de Uso 02	11/-7 – 13/07	20
I4	Entrega do Caso de Uso 03	15/07 – 30/07	10
I5	Entrega do Caso de Uso 04	02/08 – 15/08	11
I6	Entrega do Caso de Uso 05	18/08 – 25/09	5
I7	Entrega do Caso de Uso 08	27/09 – 10/10	10
I8	Entrega do Caso de Uso 06	12/10 – 29/10	5
I9	Entrega do Caso de Uso 07	01/11 – 05/11	17

QUADRO 2 - Entregas e objetivos.
FONTE: DIEDRICH (2011)

A definição de uma data para o encerramento de determinado grupo de atividades se torna de extrema importância, tanto como parâmetro de evolução do projeto, quanto para definição de itens concorrentes sem finalização definitiva. Desta forma a estipulação de duas semanas por iteração ou um *sprint* que melhor se enquadrou as visões do cliente e da equipe.

Caso alguma entrega não seja efetuado dentro do prazo haverá um grande conflito entre ambas as partes envolvidas, que pode gerar atrasos nas entregas futuras, ou até insatisfação por parte do cliente. A demora nesse caso é considerada um desperdício muito grande e como consequência haverá desvalorização do trabalho.

4.1.2 Decidir o mais tarde possível

Outra característica bastante encontrada é a parte de retardar ao máximo as decisões para que sejam mais fáceis de encaixar as mudanças requisitadas por todos os clientes que desejam melhorar ao máximo o desempenho e utilização do produto. Isto é um requisito definido também pelo manifesto ágil, como valor de resposta além de um plano único que permite deferimentos e mudanças.

Quando houver a necessidade de implementação de um requisito não planejado, o projeto pode vir acompanhado de uma entrega fora do prazo, e em consequência vários outros atrasos. Por esse motivo as ações de desenvolvimento dos requisitos só devem ser efetivamente aprovadas dentro do *sprint*, quando houver a certeza de que todas as possibilidades de mudança não existirem mais.

Ao inicio de uma nova tarefa por um membro da equipe, é gerado várias ações a serem concretizadas, e são contadas dentro do tempo estimado. Caso alguma interferência surgir nesse ponto o líder da equipe ira perder todo o controle para a entrega do grupo de atividades para a data prevista.

Se uma decisão não tomada for levantada antes do inicio das tarefas a mesma será estimada novamente e outras tarefas consequentemente deverão ser realocadas ou perderem a relevância para a entrega e finalização do *sprint*.

A priorização dos itens conforme a graduação de complexidade, estipulada por *planning poker*, de acordo com a estimativa proposta por cada membro da equipe. Lembrando esta técnica pode ser substituída por métodos similares, que funcionam da mesma maneira, como por exemplo, estimar utilizando elementos da sequência *Fibonacci*. O resultado da discussão efetuada pela equipe resultou no QUADRO 3 de DIEDRICH (2011).

Nome / Descrição	Prioridade	Estimativa (Points)	Responsável	Estimativa(hours)
Suportar um controle de vendas simples	5	8	João	48
Realizar o *design*			Carlos	12
Implementar e testar códigos do servidor (Server)			Lucas	14
Implementar e testar códigos do cliente (PDV)			Lucas	28
Atualizar documentação de usuário			Maria	6
Produzir uma versão demo e disponibilizar para stakeholders	10	5	Lucas	4
Atualizar documentação final	2	5	João	65
Atualizar manual de instalação	2	1	João	5
Atualizar notas de versão	2	1	João	4

QUADRO 3 - Priorização dos itens de trabalho.
FONTE: DIEDRICH (2011)

Estimar os casos de uso de um projeto é a principal forma de manter um controle sobre todas as ações a serem desenvolvidas, por isso é uma das primeiras tarefas a serem realizadas de forma minuciosa pela equipe para não haver problemas futuros.

4.1.3 Eliminar o desperdício

A maneira mais rápida de eliminar o desperdício é saber se o cliente realmente irá gostar do requisito apresentado, com isso a integração permite que além das mudanças motivadas por decisões tardias também auxiliem na modelagem perante a equipe que esta desenvolvendo podendo assim prever problemas que uma visão fechada da entrega não permitiria.

O desperdício geralmente é encontrado em decisões não levantadas corretamente e que não ganharam a devida priorização. Isso ocorre quando um membro da equipe sofre influência de outro durante a reunião de definição e

priorização das estórias, ou quando o mesmo não possui total habilidade sobre o requisito a ser implementado.

Os *frameworks* preveem alguns desvios de decisões perante a equipe, que são automaticamente descartados, porém se mesmo assim houver um desnível muito grande entre o conhecimento dos membros da equipe o desperdício estará fadado a ocorrer, não havendo ação de contorno a ser tomada. Como resultado da evolução dos trabalhos a equipe mantém o gráfico de *burndown* (FIGURA 9), para verificar desvios e poder priorizar a aplicação de mudanças. Este gráfico geralmente é disposto à equipe inteira juntamente com as estórias simplificadas em alguma parede, onde a equipe inteira possa se periciar e identificar itens não planejados. Nos casos dispostos para a elaboração e conclusão do projeto original de gerenciamento de pedidos, faturas e catálogos de produtos, os itens não planejados não foram encontrados, porém se houvesse uma analise mais profunda em nível de obtenção de desperdício, poderá se encontrar pontos não tratados, ou que poderia gerar itens mesclados para uma melhor distribuição das tarefas. A pré-disposição das tarefas fica a cargo da equipe, ou seja, se algum item a ser aplicado não for totalmente entendível ao membro, este fluxo deverá ser quebrado e suas estórias priorizadas novamente.

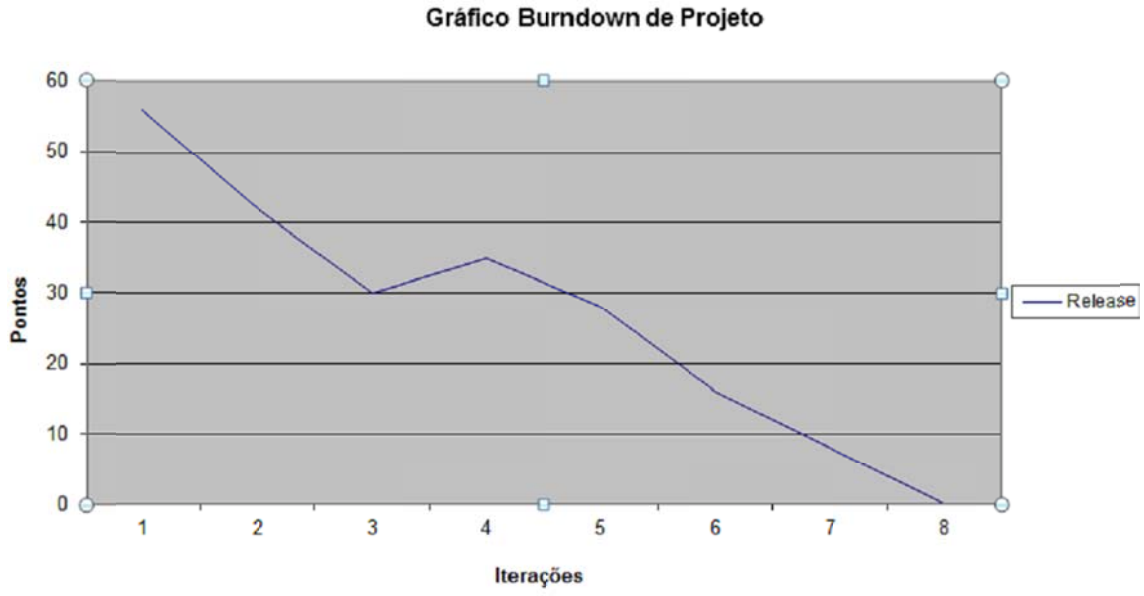

FIGURA 9 - Gráfico *burndown* do projeto.
FONTE: DIEDRICH (2011)

Como verificado no gráfico do projeto orignal, o mesmo possui um indicio de desperdício, que julgado no nível do tamanho do projeto não foi levado em consideração. Em um projeto de grande porte o desvio na linha de *release*, para realização de entrega de valores do projeto, leva a equipe a repensar vários pontos e estórias para que a mesma diferença não ocorra novamente.

Outro ponto a levar em consideração a um desvio apresentado é a aparição de itens não planejados, que leva como resultado a realocação de tarefas. Como resultado negativo desses itens a equipe terá um atraso de itens propostos para a entrega planejada com o cliente, e com isso até a perda da confiança do cliente perante a equipe.

4.1.4 Capacite a equipe para construir com qualidade

Como é demonstrado pelos *frameworks* através dos seus artefatos, nada é fixo, ou seja, a equipe esta livre para alterar quaisquer artefatos que julgarem necessários para o bem do projeto e continuação da equipe. Com isso os *frameworks* são moldados facilmente, o que nem sempre os torna previsíveis dentro de uma única premissa, *Kanban* ou *Scrum*, e sim permite moldar processos únicos com artefatos variados que nem sempre são encontrados em um *framework* especifico.

Para cada decisão sobre um artefato é realizada no mínimo uma reunião, onde são envolvidos todos os membros da equipe, e é discutido o que foi ou será feito e o que a equipe obteve de ponto relevante que deverá ser lembrado daquele ponto em diante por todos, para uma melhora dos projetos futuros.

Os pontos levantados como prioritários, são considerados as lições a serem levadas adiante, como discussão e capacitação. Quando o quesito é levantado durante o projeto, ele deverá ser levado em consideração para as novas iterações, melhorando a qualidade da construção.

Com a equipe focada nas entregas contínuas o cliente irá perceber pontos que se julgam necessários ao sistema de pedidos, desta forma alguns itens serão priorizados e esses novos requisitos devem ser calculados como extras para finalização, ou alguns devem ser removidos da lista de prioridades pelo próprio

cliente em consentimento com o líder da equipe. Para a situação hipotética utilizada os itens realocados já estão presentes, com isso os itens descartados não constam mais como priorização de entrega.

5 CONSIDERAÇÕES FINAIS

Esta pesquisa teve como principal objetivo apresentar as características encontradas nos artefatos definidos por cada *framework* utilizado para desenvolvimento ágil que fazem parte do pensamento *Lean*, prevendo assim os desperdícios que não são visíveis ou analisados corretamente em um projeto de grande porte, podendo também ser utilizados por equipes que não têm total domínio sobre metodologias ágeis.

Para projetos maiores aplicam correções a itens não planejados ou que não foram levantados como requisitos inicias, empregando a deliberação do desperdício e definindo ações antecipadas a isso traz uma melhora expressiva na qualidade do projeto e cumprimento das datas previstas. Estes desvios são facilmente identificados quando há a manutenção de um quadro de tarefas, em conjunto com um gráfico de *burndown*, também artefatos encontrados em *frameworks* que empregam as metodologias ágeis.

As equipes que não possuem total domínio sobre os métodos ágeis podem iniciar a aplicação dos mesmos juntamente com os princípios dos pensamentos *Lean*, tendo assim bons resultados e tempos de resposta a essa escolha por metodologias ágeis mais rapidamente, por terem uma melhor visão das características que trazem as melhorias propostas. Assim a troca irá se adaptar mais rapidamente à equipe, que já poderá contar com os lucros desejados.

Como resultado da obtenção das características *Lean* encontradas no estudo experimental aplicado, encontram-se entregas rápidas e continuas como garantia do ciclo de tarefas e valorização do trabalho realizado, e o retardamento das decisões de mudanças para que não haja alterações nos requisitos ocasionando entregas fora do prazo. Outro ponto relevante também é a capacitação da equipe, ou seja, quanto mais a equipe obtém novas experiências dentro de um *sprint*, que podem ser levadas adiante dentro do projeto para redução dos pontos deduzidos como desperdício, ajudando assim na melhora e obtenção de maiores lucros pela equipe.

5.1 TRABALHOS FUTUROS/CONTINUAÇÃO DO TRABALHO

Uma abordagem que pode ser estudada seria a aplicação das características que visam eliminação do desperdício em um estudo de caso real, onde cada requisito levantado teria o seu respectivo valor e funcionalidade desejada. Para elaboração do estudo de caso pode ser levado em consideração outras metodologias, tais como a metodologia *Feature Driven Development* ou *Dynamic Systems Development Method* que possuem princípios do pensamento *Lean* visando um projeto enxuto.

REFERÊNCIAS

BAUCH, C. **Lean Product Development: Making waste transparent**. Lean Advancement Initiative - Massachusetts Institute of Technology. Massachusetts, p. 140. 2004.

CRESCÊNCIO, S. A Pirâmide Lean. O Equilíbrio das Forças Ágeis. **Engenharia de Software Magazine**, São Paulo, v. 1, n. 38, p. 7-19, Julho 2011. ISSN 1983127-7.

CUNNINGHAM, W. Manifesto para Desenvolvimento Ágil de Software. **Manifesto para Desenvolvimento Ágil de Software**, 2001. Disponivel em: <http://agilemanifesto.org/iso/ptbr/>. Acesso em: 30 jul. 2011.

DIEDRICH, L. G. **Integração da Metodologia Ágil Openup nos Processos de Engenharia de Software**. Universidade Tecnológica Federal do Paraná. Medianeira. 2011.

FAYAD, M.; SCHMIDT, D. Object-Oriented Application Frameworks. New York: Communications of the ACM, v. 40, 1997. Cap. 10.

FRAMEWORK. **Wikipédia, a enciclopédia livre**, 2008. Disponivel em: <http://pt.wikipedia.org/wiki/Framework>. Acesso em: 03 mar. 20125.

GADIOLI, J. A. D. S.; CÓ, F. D. A.; ANDRADE, L. C. M. A organização industrial pela logística da manutenção: uma abordagem lean manutence. **Caderno Setec do MEC**, 2007.

GONÇALVES, G. M. D. A gerência de projetos de software em duas perspectivas – Parte 2. **Engenharia de Software Magazine**, São Paulo, v. 1, n. 38, p. 20-26, Julho 2011. ISSN 1983127-7.

GRANDO, N. Metodologias Ágeis no Desenvolvimento de Projetos de Software, 06 set. 2010.

HIGHSMITH, J. History: The Agile Manifesto. **Manifesto for Agile Software Development**, 2001. Disponivel em: <http://agilemanifesto.org/history.html>. Acesso em: 30 jul. 2011.

KNIBERG, H. **Scrum e XP direto das Trincheiras**. 1. ed. Estados Unidos da América: C4Media Inc, v. 1, 2007.

KNIBERG, H.; SKARIN, M. **Kanban e Scrum obtendo o melhor de ambos**. 1. ed. Estados Unidos da América: C4Media Inc., v. 1, 2009.

LEFFINGWELL, D. **Agile software requirements:** lean requirements practices for teams, programs, and the enterprise. 1. ed. Upper Saddle River, NJ: Pearson Education, v. 1, 2010.

OHNO, T. **O sistema Toyota de Produção:** além da produção em larga escala. Tradução de Cristina Schumacher. 1. ed. Porto Alegre: Bookman, 1997.

POPPENDIECK, M.; POPPENDIECK, T. **Lean Software Development:** An Agile Toolkit. 1. ed. Upper Saddle River, NJ: [s.n.], v. 1, 2003.

SCHWABER, K. **Agile Project Management with Scrum**. 1. ed. Washington: Microsft Press, v. 1, 2004.

SUTHERLAND, J.; SCHWABER, K. **The Scrum Papers:** Nuts, Bolts, and Origins of an Agile Process. 1. ed. Newton: PatientKeeper, Inc., v. 1, 2007.

SUTHERLAND, J.; SCHWABER, K. Scrum Guide. **Scrum.org**, 2011. Disponivel em: <http://www.scrum.org/>. Acesso em: 22 jan. 2012.