FLAVIO YUITI NAKASHIMA

APLICAÇÃO DO KANBAN E METODOLOGIAS ÁGEIS NO DESENVOLVIMENTO DE SOFTWARE

SÃO PAULO
2010

FLAVIO YUITI NAKASHIMA

APLICAÇÃO DO KANBAN E METODOLOGIAS ÁGEIS NO DESENVOLVIMENTO DE SOFTWARE

Monografia apresentada ao Programa de Educação Continuada (PECE) da Escola Politécnica da Universidade de São Paulo para obtenção do título de MBA em Tecnologia da Informação.

SÃO PAULO
2010

FLAVIO YUITI NAKASHIMA

APLICAÇÃO DO KANBAN E METODOLOGIAS ÁGEIS NO DESENVOLVIMENTO DE SOFTWARE

Monografia apresentada ao Programa de Educação Continuada (PECE) da Escola Politécnica da Universidade de São Paulo para obtenção do título de MBA em Tecnologia da Informação.

Área de concentração:
Engenharia de Software.

Orientador:
Prof. Dr. Jorge Luis Risco Becerra.

SÃO PAULO
2010

FICHA CATALOGRÁFICA

Nakashima, Flavio Yuiti
Aplicação do Kanban e metodologias ágeis no desenvolvimento de software / F.Y. Nakashima. -- São Paulo, 2010.
p.

Monografia (MBA em Tecnologia da Informação) - Escola Politécnica da Universidade de São Paulo. Programa de Educação Continuada em Engenharia.

1. Engenharia de software 2. Processo de software 3. Métodos ágeis I. Universidade de São Paulo. Escola Politécnica. Programa de Educação Continuada em Engenharia II. t.

DEDICATÓRIA

Dedico este trabalho a minha mãe Alice Katsuko Nakashima.

AGRADECIMENTOS

Ao Prof. Dr. Jorge Luis Risco Becerra, pela orientação, ensinamentos e estímulo a reflexão para a elaboração deste trabalho.

Aos meus familiares, amigos e colegas de trabalho que incentivaram e colaboraram para a realização deste trabalho.

O nosso negócio não é determinado pelo produtor, mas pelo cliente. Não é definido pelo nome da empresa, seus estatutos ou requisitos, mas pelo desejo que o cliente satisfaz quando compra um produto ou serviço. Trata-se, pois, de uma questão que só se resolve olhando para o negócio do lado de fora, do ponto de vista do cliente.

(Peter F. Drucker)

RESUMO

Este trabalho analisa o tema do Desenvolvimento Enxuto de Software e das Metodologias Ágeis para dar base a um modelo de gerenciamento de desenvolvimento de software.

Esta abordagem utiliza o sistema *Kanban*, que tem sua origem na produção enxuta, como elemento central para fazer fluir todo o trabalho necessário para atender uma demanda da lista de desejos do cliente, transformando-o em software funcional. O foco do modelo é a entrega de software *Just-in-time*. Isso significa essencialmente entregar o software certo na hora certa e fazer isso repetidamente com o objetivo de diminuir ao máximo o tempo entre a demanda e a entrega, por meio da detecção e eliminação das perdas que ocorrem nesse intervalo.

Este modelo propõe o uso do sistema *Kanban* como elemento chave de gestão para incrementar um desenvolvimento de software baseado na metodologia ágil *Extreme Programming (XP)*.

Palavras-Chave: Desenvolvimento Enxuto de Software. Metodologias Ágeis.

ABSTRACT

This paper analyzes the topic of Lean Software Development and Agile Methodologies to base a management model of software development.

This approach uses the Kanban system, which has its origin in lean production as a key element to flow all the work necessary to meet the demands of the wish list of the customer, turning it into working software. The focus of the model is to deliver software just-in-time. This essentially means delivering the right software on time and do it repeatedly in order to reduce the maximum time between demand and supply, through the detection and elimination of wastes that occur in that range.

This model proposes the use of Kanban system as a key element management to enhance a software development based on agile methodology Extreme Programming (XP).

Keywords: Lean Software Development. Agile Methodologies.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

BPMN – Business Process Management Notation

CASE – Computer-Aided System Engineering

CFD – Cumulative Flow Diagram

DSDM – Dynamic Systems Development Method

FDD – Feature Driven Development

JIT – Just-In-Time

XP – Extreme Programming

WIP – Work-In-Process

SUMÁRIO

1 INTRODUÇÃO

Os negócios atualmente operam em um ambiente global sujeito a rápidas mudanças. Eles têm de responder a novas oportunidades e mercados, mudanças de condições econômicas e ao surgimento de produtos e serviços concorrentes. O software é parte de quase todas as operações de negócios e, assim, é essencial que um novo software seja desenvolvido rapidamente para aproveitar as novas oportunidades e responder às pressões competitivas. Desenvolvimento e entrega rápidas são, portanto, muitas vezes o requisito mais crítico para sistemas de software (SOMMERVILLE, 2007).

Os processos de desenvolvimento de software baseados em especificações completas de requisitos, projeto, construção e teste de sistema, tal como um processo em cascata convencional, não são adequados para este ambiente de negócios de rápidas mudanças.

Neste contexto onde os requisitos do sistema mudam rapidamente durante o processo de desenvolvimento, surgiram metodologias que contam com uma abordagem iterativa para especificação, desenvolvimento e entrega de software. São baseadas na noção de desenvolvimento e entregas incrementais onde um software de trabalho é entregue rapidamente aos clientes, que podem então propor novos requisitos e alterações a serem incluídos nas iterações posteriores do sistema. Estas metodologias são conhecidas como métodos ágeis. Dentre as mais conhecidas podemos citar o *Extreme Programming (XP)* e o *Scrum*.

Dentro da comunidade que desenvolve e dissemina os métodos ágeis há uma nova abordagem chamada de *Lean Software Development*, ou Desenvolvimento Enxuto de Software. Esta abordagem esta baseada na aplicação e interpretação dos princípios e valores enxutos ao processo de desenvolvimento de software.

1.1 Objetivos

O principal objetivo deste trabalho é a proposição de um modelo de gerenciamento de desenvolvimento de software baseado em um sistema *Kanban*,

que implementa um sistema baseado no fluxo de valor, fluxo unitário e um paradigma de cadência de entrega, que é muito diferente do paradigma tradicional de projeto com gestão de escopo, cronograma, dedicado conjunto de recursos, e data de entrega.

Neste trabalho é apresentado como montar e operar um quadro *Kanban*, que é a ferramenta central do Sistema *Kanban* aplicado ao desenvolvimento de software, implantando um sistema puxado com os limites de trabalhos em processo definidos, para expor os gargalos do processo e que em um processo de melhoria contínua para eliminar os desperdícios pode-se reduzir o tempo entre a demanda e a entrega de software funcional para os clientes.

Neste caso o Sistema *Kanban* é aplicado para incrementar o desenvolvimento de software baseado em *Extreme Programming*.

Outro objetivo deste trabalho é mostrar como a implantação de um sistema *Kanban* pode orientar uma organização para adoção do desenvolvimento enxuto de software.

1.2 Justificativa

Kanban tem provado ser uma ferramenta útil para a evolução cultural e gestão da mudança. *Kanban* expõe o mecanismo do fluxo de valor para o desenvolvimento. Isso permite a identificação de gargalos e uma melhor compreensão dos entraves ao bom fluxo. O resultado é uma abordagem gradual e incremental de mudança que se habilita para todos na equipe. *Kanban* reduz as barreiras à adoção de métodos ágeis. Ele permite que se comece com o processo existente, mesmo que seja um processo cascata. *Kanban* abraça a especialização da força de trabalho, enquanto expõe os efeitos que muitos especialistas apresentam como gargalos, filas entre transferências, e conseqüentemente *WIP* maior (ANDERSON, 2009).

Kanban permite uma abordagem evolutiva para uma transição ágil. Ela proporciona evolução baseada em princípios enxutos em vez de revolução com base no Manifesto Ágil. Ele provou ser fácil de adotar e reduz a resistência à mudança. *Kanban* elimina os custos dos desperdícios de coordenação sem

sacrificar a satisfação do cliente. Muitos procedimentos e artefatos da gestão de projeto que muitas vezes são permitidos, realmente deveriam ser vistos como desperdício em termos enxutos. *Kanban* reduz ou elimina os desperdícios de sobrecarga de coordenação que os métodos de gestão de projetos típicos introduzem (ANDERSON, 2009).

1.3 Metodologia

A metodologia aplicada para se atingir o objetivo deste trabalho são:

Pesquisa Bibliográfica: Estudo das referências de sustentação dos conceitos, princípios e valores da produção enxuta que são base para a definição do Desenvolvimento Enxuto de Software e do Sistema *Kanban* aplicado ao gerenciamento de desenvolvimento de software. E como os princípios da Mentalidade Enxuta se relacionam com os princípios das Metodologias Ágeis.

Proposição do Método: Nesta fase serão organizados os conceitos estudados para compor um método de gerenciamento de desenvolvimento de software baseado em um sistema *Kanban* combinado com o *Extreme Programming*.

Conclusão: Nesta última fase são apresentadas as conclusões sobre o método proposto.

1.4 Estrutura do Trabalho

A apresentação deste trabalho segue a seguinte seqüência de capítulos:

Capítulo 1: introdução ao trabalho, seus objetivos, as motivações e justificativas que levaram ao desenvolvimento desta dissertação, assim como a abrangência e a metodologia utilizada.

Capítulo 2: descreve sobre os princípios e valores da Mentalidade Enxuta. Sua origem na produção enxuta e como estes conceitos são levados para o desenvolvimento de software. E a sua relação com os princípios e valores das Metodologias Ágeis.

Capítulo 3: descreve sobre o modelo proposto para gerenciamento do desenvolvimento de software aplicando o sistema *Kanban* combinado com o *Extreme Programming*.

Capítulo 4: conclusões.

2 Desenvolvimento Enxuto de Software

O Desenvolvimento Enxuto de Software está relacionado à interpretação e aplicação dos princípios enxutos para o processo de desenvolvimento de software. O termo enxuto foi popularizado pelo trabalho desenvolvido por James Womack, Daniel Jones e Daniel Roos (1990), com o livro *A Máquina que Mudou o Mundo*. Este trabalho apresentou inúmeros dados de *benchmarking* mostrando que há uma forma melhor de organizar e gerenciar os relacionamentos com clientes, cadeia de fornecedores, desenvolvimento de produto em contraponto aos métodos e práticas gerenciais desenvolvidas pelas indústrias voltadas a produção em massa. Tratava-se de uma forma de se fazer mais com cada vez menos. Após a Segunda Guerra Mundial, a Toyota foi pioneira em utilizar esta abordagem, desenvolvendo o Sistema Toyota de Produção (STP) que é a base para o que se chama de produção enxuta.

No outono de 1973 houve a primeira crise do petróleo, gerentes japoneses acostumados à inflação e às altas taxas de crescimento, se viram confrontados com crescimento zero e forçados a lidar com a queda de produção. Pela primeira vez foi notado os resultados que a Toyota estava conseguindo com a sua perseguição à eliminação do desperdício.

2.1 A Mentalidade Enxuta

A mentalidade enxuta ou pensamento enxuto (*lean thinking*) é um termo cunhado por Womack e Jones(2004) que determina uma filosofia de negócios baseada no modelo de gestão que a Toyota desenvolveu, que vai além da produção enxuta, envolvendo a organização como um todo.

Taiichi Ohno (1997) definiu o modelo Toyota da seguinte forma:

> *"O que estamos fazendo é observar a linha de tempo desde o momento e que o cliente nos faz um pedido até o ponto em que recebemos o pagamento. E estamos reduzindo essa linha de tempo, removendo as perdas que não agregam valor."*

Womack e Jones (2004) definiram cinco princípios que determinam o pensamento enxuto:

1. *Determinar precisamente o **valor** por produto específico*. O ponto de partida essencial para o pensamento enxuto é o valor. O valor só pode ser definido pelo cliente final. E só é significativo quando expresso em termos de um produto específico (um bem ou um serviço e, muitas vezes, ambos simultaneamente) que atenda às necessidades do cliente a um preço específico em momento específico.

2. *Identificar o **fluxo de valor** para cada produto*. O fluxo de valor é o conjunto de todas as ações específicas para se levar um produto específico a passar pelas três tarefas gerenciais críticas em qualquer negócio: a tarefa de solução de problemas que vai da concepção até o lançamento do produto, passando pelo projeto detalhado e pela engenharia, a tarefa de gerenciamento da informação, que vai do recebimento do pedido até a entrega, seguindo um detalhado cronograma, e a tarefa de transformação física, que vai da matéria-prima ao produto acabado nas mãos do cliente. Identificando os processos que efetivamente geram valor, aqueles que não geram valor, mas são importantes para a manutenção dos processos e da qualidade e, por fim, aqueles que não agregam valor, devendo ser eliminados imediatamente.

3. *Fazer o valor **fluir** sem interrupções*. Criar o fluxo contínuo com os processos e atividades que restaram após a identificação do fluxo de valor. Isso exige uma mudança na mentalidade das pessoas, porque precisam deixar de lado a idéia que têm de produção por departamentos como a melhor alternativa.

4. *Deixar que o cliente **puxe** valor do produtor*. Significa inverter o fluxo produtivo, onde as empresas não mais empurram os produtos para o consumidor através de descontos e promoções. O consumidor passa a puxar o fluxo de valor, reduzindo a necessidade de estoques e

valorizando o produto. Sempre que não se consegue estabelecer o fluxo contínuo, conectam-se os processos através de sistemas puxados.

5. *Buscar a **perfeição***. A busca do aperfeiçoamento contínuo em direção a um estado ideal deve nortear todos os esforços da empresa, em processos transparentes onde todos os membros da cadeia (montadoras, fabricantes de diversos níveis, distribuidores e revendedores) tenham conhecimento profundo do processo como um todo, podendo dialogar e buscar continuamente melhores formas de criar valor.

O pensamento enxuto envolve uma estratégia de negócios para aumentar a satisfação dos clientes através da melhor utilização dos recursos, fornecendo consistentemente valor aos clientes com custos mais baixos, através da melhoria contínua dos processos e pela compreensão das pessoas e da motivação humana.

2.2 A Produção Enxuta

A história da produção enxuta tem suas raízes no Japão com a família Toyoda que antes de entrar para o ramo automobilístico, começou com o ramo da tecelagem produzindo teares.

Em 1926, Sakichi Toyoda inaugurou a Toyoda Automatic Loom Works, empresa-mãe do Grupo Toyota e ainda hoje um participante importante no conglomerado Toyota. Em seu interminável trabalho como funileiro e inventor, Sakichi criou sofisticados teares automáticos. Entre suas invenções, havia um mecanismo especial para interromper o funcionamento de um tear toda vez que um fio se partisse. Esta invenção evoluiu para um sistema mais amplo que se tornou um dos pilares do Sistema Toyota de Produção, chamado autonomação (automação com um toque humano). Essencialmente, autonomação significa acréscimo de qualidade enquanto se produz o material indicando que há um problema. Refere-se também a criação de operações e de equipamento para que os funcionários não

fiquem amarrados às máquinas, e sim livres para desempenhar tarefas que agregam valor ao produto.

Sakichi Toyoda sabia que o mundo estava mudando e que os teares automáticos se tornariam tecnologia do passado enquanto os automóveis eram a tecnologia do futuro. Em função disso Sakichi deu a seu filho Kiichiro Toyoda a tarefa de construir uma empresa de automóveis.

A Toyota Motor Company começou produzindo caminhões simples e nos primeiros anos os veículos eram de baixa qualidade, produzidos com tecnologia primitiva e desta forma não obteve muito sucesso. Nos anos 30, os líderes da Toyota visitaram a Ford e a GM para estudar suas linhas de montagem e leram atentamente o livro de Henry Ford, *Today and Tomorrow* (1926).

A Toyota percebeu que o mercado japonês era muito reduzido a que a demanda era muito fragmentada para suportar os grandes volumes de produção nos Estados Unidos. Os administradores da Toyota sabiam que, para sobreviver em longo prazo, teriam que adaptar a abordagem de produção em massa ao mercado japonês. O desafio era aperfeiçoar o processo de produção da Toyota de modo que se igualasse à produtividade da Ford. Este foi o desafio proposto a Taiichi Ohno, então administrador da empresa, por Eiji Toyoda, sobrinho de Sakichi Toyoda, que assumiu a presidência da Toyota Motor Company após Kiichiro Toyoda perdir demissão da presidência, em função da crise vivida na empresa no início do pós-guerra.

Ohno diante do desafio recebido de Eiji Toyoda de “igualar-se a Ford em produtividade”, também observou a concorrência em visitas posteriores aos Estados Unidos. Um dos principais componentes que Ohno acreditava que a Toyota precisava dominar era o fluxo contínuo e o melhor exemplo disso na época era a linha de montagem da Ford em operação.

Apesar de em seu livro, Ford pregar a importância de criar um fluxo contínuo de material no decorrer do processo de produção, padronizar os processos e eliminar as perdas, nem sempre isso era praticado em sua empresa. A Toyota viu métodos esbanjadores de produção por lotes que formavam grandes depósitos de estoque em processo na cadeia de valor, empurrando o produto para o próximo passo da produção, e entendeu isso como uma falha inerente ao sistema de produção em massa da Ford.

A Toyota acreditou que poderia usar a idéia original de Ford, do fluxo contínuo de material, para desenvolver um sistema de fluxo unitário de peças que flexivelmente mudasse de acordo com a demanda dos clientes e que ao mesmo tempo fosse eficiente. A flexibilidade exigia que a engenhosidade dos funcionários fosse direcionada para que melhorassem continuamente o processo.

2.2.1 As Bases do Sistema Toyota de Produção

As bases do Sistema Toyota de Produção foram desenvolvidas após anos e décadas de prática. Evoluiu para atender os desafios específicos que a Toyota enfrentava à medida que crescia como empresa. Evoluiu à medida que Taiichi Ohno, seus engenheiros, administradores e operários aplicaram os princípios de autonomação e de fluxo unitário de peças durante anos de tentativa e erro.

A base do Sistema Toyota de Produção é a absoluta eliminação do desperdício. Ohno passava boa parte de seu tempo na fábrica, aprendendo a mapear as atividades que agregavam valor ao produto e livrando-se das atividades que não agregavam valor.

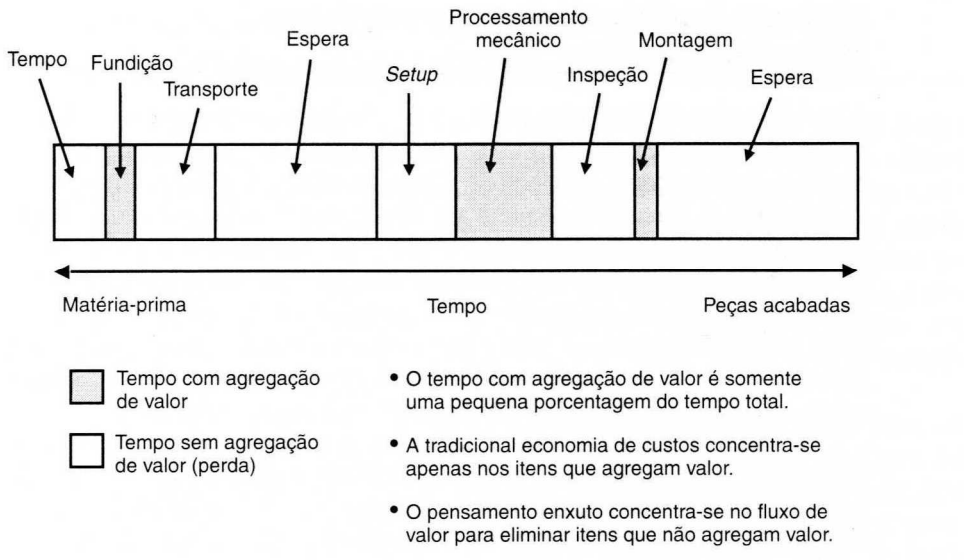

Figura 1 – Perdas em um sistema de valor (Liker, 2005)

A Toyota identificou sete grandes tipos de perdas sem agregação de valor em processos administrativos ou de produção que devem ser identificados como passo

preliminar para a aplicação do Sistema Toyota de Produção. Abaixo está uma tabela com as descrições dos desperdícios feitas por Liker (2005):

Desperdício	Descrição
Superprodução	Produção de itens para os quais não há demanda, o que gera perda com excesso de pessoal e de estoque e com custos de transporte devido ao estoque excessivo.
Espera (tempo sem trabalho)	Funcionários que servem apenas para vigiar uma máquina automática ou que ficam esperando pelo próximo passo no processamento, ferramenta, suprimento, peça, etc., ou que simplesmente não têm trabalho para fazer devido a uma falta de estoque, atrasos no processamento, interrupção do funcionamento de equipamentos e gargalos de capacidade.
Transporte ou movimentação desnecessário	Movimento de estoque em processo por longas distâncias, criação de transporte ineficiente ou movimentação de materiais, peças ou produtos acabados para dentro ou fora do estoque ou entre processos.
Superprocessamento ou processamento incorreto	Passos desnecessários para processar as peças. Processamento ineficiente devido a uma ferramenta ou ao projeto de baixa qualidade do produto, causando movimento desnecessário e produzindo defeitos. Geram-se perdas quando se oferecem produtos com qualidade superior à que é necessária.
Excesso de estoque	Excesso de matéria-prima, de estoque em processo ou de produtos acabados, causando *lead times* mais longos, obsolescência, produtos danificados, custos de transporte e de armazenagem e atrasos. Além disso, o estoque extra oculta problemas, como desbalanceamento de produção, entregas atrasadas dos fornecedores, defeitos, equipamentos em conserto e longo tempo de *setup* (preparação).
Movimento desnecessário	Qualquer movimento inútil que os funcionários têm que fazer durante o trabalho, tais como procurar, pegar ou empilhar peças, ferramentas, etc. Caminhar também é perda.
Defeitos	Produção de peças defeituosas ou correção. Consertar ou re-trabalhar, descartar ou substituir a produção e inspecionar significam perdas de manuseio, tempo e esforço.

Liker(2005) acrescentou mais um tipo de desperdício em sua análise:

- *Desperdício da criatividade dos funcionários*. Perda de tempo, idéias, habilidades, melhorias e oportunidades de aprendizagem por não envolver ou ouvir seus funcionários.

Durante décadas o Sistema Toyota de Produção foi desenvolvido e aperfeiçoado, mas sem que a teoria fosse documentada. Isso teve início com Fujio Cho, discípulo de Taiichi Onho, que desenvolveu uma representação simples para o STP, uma casa.

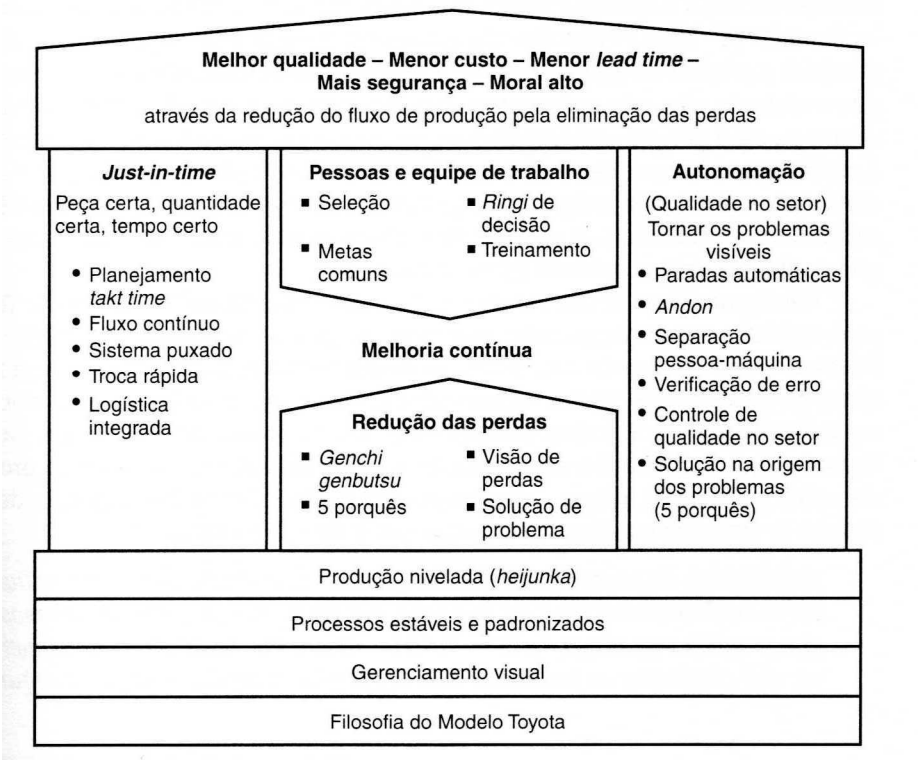

Figura 2 – O Sistema Toyota de Produção (Liker, 2005).

A representação utilizando uma casa transmite a idéia de que o sistema é baseado em uma estrutura e não apenas em um conjunto de técnicas. A casa só é forte se o telhado, as colunas e as fundações são fortes. Uma conexão fraca fragiliza todo o sistema.

O telhado representa as metas de melhor qualidade, menor custo e menor *lead time*. Nos pilares de sustentação estão o *Just-In-Time (JIT)* e a autonomação. Nos alicerces há vários processos e o nivelamento da produção, que significa nivelar a programação de produção tanto em volume quanto em variedade, para manter a estabilidade do sistema e permitir um mínimo de estoque. No centro do sistema

estão as pessoas, para que com a melhoria contínua atinjam a estabilidade necessária da operação. E as pessoas devem ser treinadas para encontrar os desperdícios e eliminar os problemas pela raiz.

Just-In-Time significa que, em um processo de fluxo, as partes corretas necessárias à montagem alcançam a linha de montagem no momento em que são necessários e somente na quantidade necessária. Uma empresa que estabeleça esse fluxo integralmente pode chegar ao estoque zero (Ohno, 1997).

Com a autonomação não é necessário um operador enquanto a máquina estiver funcionando normalmente, apenas quando há uma situação anormal. Desta forma um operador pode atender diversas máquinas, tornando possível reduzir o número de operadores e aumentar a eficiência da produção. A parada da máquina força a parada da linha e mostra o sentido de urgência para resolver o problema. Quando o problema é compreendido, a melhoria é possível.

2.2.2 O Fluxo de Processo Contínuo

No modo tradicional da produção em massa, máquinas semelhantes e pessoas com habilidades semelhantes são agrupados em departamentos. O pensamento enxuto observa nesse modo de organizar a produção, uma empresa produzindo um grande estoque de trabalho em processo *(work-in-process – WIP)*.

Para eliminar desperdícios, a produção enxuta organiza o trabalho de modo a criar um fluxo contínuo em busca de um fluxo unitário de peças. Os processos são alinhados fisicamente na seqüência que produzirá o que foi solicitado pelo cliente no menor período de tempo. O resultado disso é o aumento da eficiência da produção, porque os produtos se movem continuamente no processamento com um tempo mínimo de espera entre as etapas e a menor distância de deslocamento. O tempo de produção será reduzido, diminuindo o custo do ciclo.

Liker (2005) utiliza o exemplo abaixo para mostrar as diferenças entre um processamento em lote e um fluxo contínuo:

- A figura 3 ilustra uma visão simplificada de um fabricante de computadores organizado em três departamentos. Um departamento fabrica as bases dos computadores, o segundo produz os monitores e os conecta e o terceiro testa o

equipamento. Nesse modelo, o departamento de movimentação de material decidiu movimentar um lote de 10 unidades por vez. Cada departamento precisa de um minuto por unidade para fazer seu trabalho, de maneira que são necessários 10 minutos para que um lote de computadores passe de um departamento para outro. Mesmo sem considerar o tempo de movimentação de material entre departamentos, levaria 30 minutos para fabricar e testar o primeiro lote de 10 computadores a ser enviado para o cliente. E levaria 21 minutos para obter o primeiro computador pronto para ser embarcado, embora somente três minutos de trabalho com agregação de valor sejam necessários para produzir aquele computador.

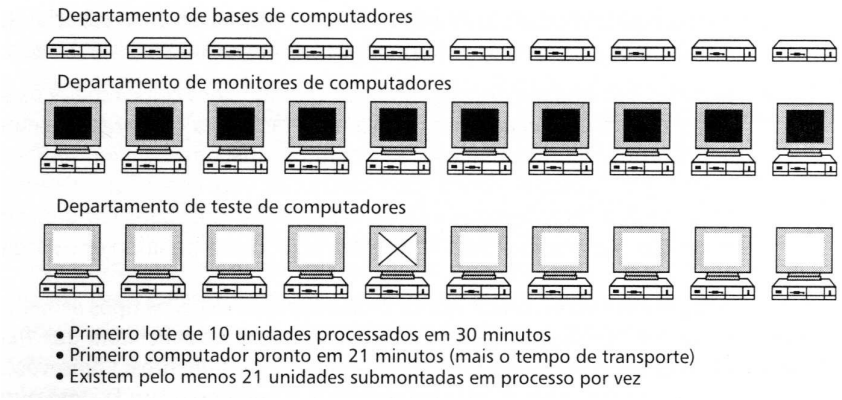

Figura 3 – Exemplo de processamento de lotes (Liker, 2005).

- A figura 4 apresenta uma perspectiva do mesmo processo de fabricação de computadores visto acima, organizado em uma célula de trabalho de fluxo unitário de peças. O funcionário responsável pela produção da base não faria outra base antes de o encarregado dos monitores terminar de preparar um monitor e montá-lo na última base. Em outras palavras, ninguém construiria mais do que fosse imediatamente necessário. O resultado é que os operadores na célula levariam 12 minutos para fazer 10 computadores, enquanto que o processo de fluxo de lotes leva 30. E o processo enxuto precisa de apenas três minutos em vez de 21 para produzir o primeiro computador pronto para ser embarcado. De fato, os três minutos significam somente tempo que agrega valor. O que o fluxo faz é eliminar a superprodução e o estoque.

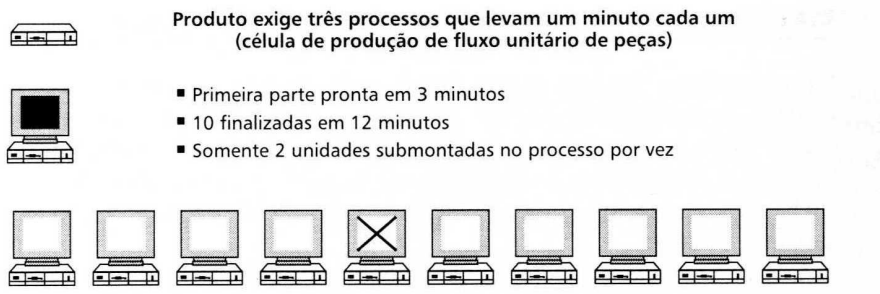

Figura 4 – Exemplo de fluxo contínuo (Liker, 2005).

- É comum achar que o aumento da velocidade de um processo implica comprometimento da qualidade, que mais rápido quer dizer mais desleixado. Mas o fluxo proporciona justamente o oposto – geralmente aumenta a qualidade. Nas figuras 3 e 4, há um computador com defeito, marcado com um X no monitor. Essa unidade não pôde ser ligada no estágio de teste. Na abordagem de grandes lotes apresentada na figura 3, quando o problema é descoberto, há pelo menos 21 unidades em processo que também poderão apresentar esse problema. E, se o defeito ocorresse no departamento de bases, 21 minutos poderiam ser necessários para descobri-lo no departamento de testes. Na figura 4, por outro lado, quando se descobre um defeito, pode haver somente outros dois computadores em processo que também apresentem o problema, e o tempo máximo necessário para encontrar o problema é de dois minutos depois de o computador ter sido feito. A realidade é que, em uma operação de grandes lotes, provavelmente decorrem semanas de estoque em processo entre operações e pode levar semanas ou mesmo meses desde o momento em que um defeito foi causado até que seja descoberto. Nesse instante, o mapeamento de causa e efeito fica confuso, tornando quase impossível rastrear e identificar o que causou o problema.

O exemplo acima mostra um processo de fluxo contínuo ideal, mas sabe-se que estabelecer um fluxo unitário de peças é extremamente difícil e exige um processo altamente elaborado e condições muito específicas. Esse nível de precisão seria excepcionalmente difícil e somente possível em casos em que o equilíbrio do tempo de ciclo fosse perfeito.

Na maioria das operações de fabricação que utilizam o fluxo unitário de peças, uma única peça é colocada entre as estações de trabalho, permitindo uma pequena variação no tempo de ciclo de cada funcionário sem causar tempo de espera. Mesmo nesse nível o equilíbrio do tempo de ciclo entre as operações precisa ser excepcionalmente alto.

2.2.3 *Takt-time*: Definindo a Cadência

Takt é uma palavra alemã para ritmo ou compasso. O *takt-time* é um conceito usado para projetar o trabalho e mede o ritmo da demanda do cliente. É o tempo disponível para produzir peças em um intervalo específico de tempo dividido pelo número de peças demandadas naquele intervalo.

Por exemplo, se o tempo disponível de operação para um turno for de 400 minutos e a demanda do produto é de 400 por turno, o tempo de dedicado por peça (*takt-time*) é um minuto. O tempo de ciclo de cada operação precisa ser um minuto ou menos em média para atender à demanda. Se o tempo de ciclo (tempo real para completar as tarefas em um único trabalho) for maior do que o *takt*, a operação será um gargalo e será necessário um tempo adicional para acompanhar a programação da produção, e se for mais rápido haverá superprodução.

O *takt-time* é a referência para se estabelecer com qual velocidade a célula de trabalho deve funcionar, qual a capacidade necessária de cada equipamento, e quantas pessoas serão necessárias para operar a célula.

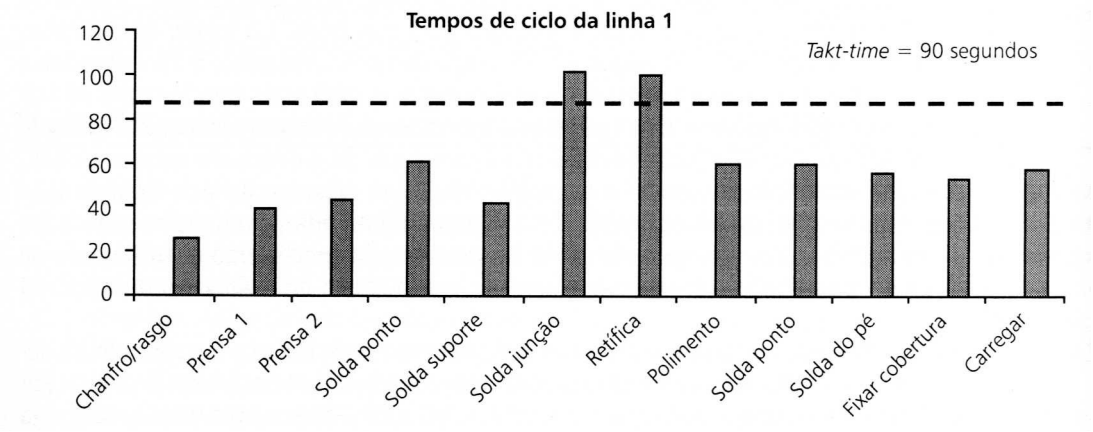

Figura 5 – Gráfico de equilíbrio de operação para comparar tempos de ciclo (Liker, 2007).

2.2.4 Sistemas Puxados (*Pull*)

Na produção enxuta, puxar significa o estado ideal da fabricação *just-in-time*, ou seja, dar ao cliente (que pode ser o próximo passo no processo de produção) o que ele quer, quando ele quer e na quantidade que deseja. O sistema puxado indica quando o material é movimentado e quem (o cliente) determina esse movimento. Ao contrário da produção em massa, onde o sistema é empurrado, baseado pela demanda do cliente projetada.

Taiichi Ohno desenvolveu o sistema puxado a partir do conceito de funcionamento de um supermercado. Ohno pegou do supermercado a idéia de visualizar o processo inicial numa linha de produção como um tipo de loja. O processo final (cliente) vai até o processo inicial (supermercado) para adquirir as

peças necessárias (gêneros) no momento e na quantidade que precisa. O processo inicial imediatamente produz a quantidade recém retirada (reabastecimento das prateleiras).

Ohno, desde o início, tinha reconhecido que estabelecer o fluxo unitário de peças ideal seria muito difícil, em função das variações dos tempos de ciclo de cada operação, e desta forma viu que seria necessário um estoque entre as operações de produção para se conseguir o fluxo contínuo. Esses estoques são conhecidos como estoques amortecedores (*buffers*), pequenos "armazéns" de peças entre as operações. Nestes "armazéns" só há reposição dos itens retirados, desta forma não haverá uma superprodução maior do que a pequena quantidade da prateleira.

2.2.5 Sistema *Kanban*

Kanban é uma palavra japonesa que significa sinal, letreiro, placa, pôster, anúncio, cartão, mas é entendido de maneira geral como algum tipo de sinal.

Um verdadeiro sistema de fluxo unitário de peças seria de estoque zero, em que os produtos aparecem exatamente quando se tornam necessários para o cliente (*just-in-time*). O sistema mais próximo disso desenvolvido pela Toyota para alcançar esse objetivo é a célula de fluxo unitário de peças, que fabrica por pedido somente no exato momento em que surge a necessidade do produto. Mas quando o fluxo puro não é possível porque os processos estão muito distantes ou porque os tempos de ciclo para desempenhar as operações variam muito, a próxima escolha é freqüentemente o Sistema *Kanban*.

Carrinhos vazios, ou latas vazias enviadas ao processo anterior para sinalizar que a linha de montagem tinha utilizado as peças e precisava de mais, são *kanbans*. A forma mais freqüentemente usada é um pedaço de papel dentro de um envelope de vinil retangular. Neste pedaço de papel a informação pode ser dividida em três categorias: (1) informação de coleta, (2) informação de transferência, e (3) informação de produção. O *kanban* carrega a informação vertical e lateralmente dentro da própria Toyota e entre a Toyota e as empresas colaboradoras.

O Sistema *Kanban* é o método para operar o Sistema Toyota de Produção, deixando claro o que deve ser feito por gerentes, supervisores e operadores. Sua

utilização mostra imediatamente o que é desperdício, permitindo um estudo criativo e propostas de melhorias. É uma força poderosa para reduzir mão-de-obra e estoques, eliminar produtos defeituosos, e impedir a recorrência de panes.

Regras para a utilização do Sistema *Kanban*:

Funções do Kanban	***Regras para Utilização***
1. Fornecer informação sobre apanhar ou transportar.	1. O processo subseqüente apanha o número de itens indicados pelo *kanban* no processo precedente.
2. Fornecer informação sobre a produção.	2. O processo inicial produz itens na quantidade e seqüência indicadas pelo *kanban*.
3. Impedir a superprodução e o transporte excessivo.	3. Nenhum item é produzido ou transportado sem um *kanban*.
4. Servir como uma ordem de fabricação afixada às mercadorias.	4. Serve para afixar um *kanban* às mercadorias.
5. Impedir produtos defeituosos pela identificação do processo que os produz.	5. Produtos defeituosos não são enviados para o processo seguinte. O resultado é mercadorias 100% livres de defeitos.
6. Revelar problemas existentes e mantém o controle de estoques.	6. Reduzir o número de *kanbans* aumenta sua sensibilidade aos problemas.

2.2.6 Produção Nivelada (*Heijunka*)

Uma abordagem comum em um processo de implementação de ferramentas enxutas é o foco apenas em eliminação de desperdícios, ou seja, os estoques nos sistemas são reduzidos, organiza-se melhor o local de trabalho para a eliminação de movimentos inúteis, verifica-se o nível de trabalho e se reduz o número de pessoas no sistema. Utilizando-se apenas desta abordagem em um plano de produção que flutue muito, a conclusão será que a produção enxuta não funciona para este caso, porque a oscilação da demanda do cliente fará com que pessoas e equipamentos trabalhem sobrecarregados, reduzindo a qualidade do produto, maior parada para manutenção de equipamentos e falta de peças.

O foco apenas em eliminação de desperdícios é comum, porque é fácil identificar e eliminar perdas, mas o que muitas empresas não conseguem é o

processo mais difícil de estabilizar o sistema e criar um verdadeiro fluxo de trabalho enxuto e equilibrado.

Os administradores e funcionários da Toyota usam o termo japonês *muda* quando falam sobre desperdícios, mas há outros dois termos importantes que são utilizados para se atingir um trabalho enxuto. A Toyota refere-se a isso como a "eliminação de *Muda*, *Muri* e *Mura*". Os significados destes termos são:

- *Muda – nenhuma agregação de valor*. Trata-se de atividades supérfluas que aumentam os *lead times*, causam movimentos extras para obter peças ou ferramentas, criam excesso de estoque ou resultam em alguma forma de espera.
- *Muri – sobrecarga de pessoas ou de equipamentos*. Significa colocar uma máquina ou uma pessoa além de seus limites naturais. A sobrecarga de pessoas resulta em problemas de segurança e qualidade. A sobrecarga do equipamento causa interrupções e defeitos.
- *Mura – desnivelamento*. Em sistemas de produção normais, às vezes há mais trabalho do que as pessoas podem realizar e outras vezes há falta de trabalho. O desnivelamento resulta de um programa de produção irregular ou de volumes de produção flutuantes devido a problemas internos, como paralisações, falta de peças ou defeitos. *Muda* é resultado de *Mura*.

Atingir o nivelamento da produção é fundamental para a eliminação de *mura*, que, por sua vez, é fundamental para a eliminação de *muri* e de *muda*.

Heijunka significa nivelar a combinação de produtos durante um período específico de tempo com o objetivo de produzir todas as peças todos os dias (ou mesmo dentro de algumas horas). Este conceito reforça a necessidade de manter os lotes pequenos e produzir o que o cliente (interno ou externo) deseja. E também a necessidade de um processo com um alto nível de flexibilidade e capacidade de resposta às mudanças na demanda do cliente.

O trabalho de acordo com um plano nivelado aplica-se a todas as áreas da Toyota, incluindo as vendas. Todos na organização trabalham juntos para realizá-lo.

2.3 Os Princípios Enxutos Aplicados ao Desenvolvimento de Software

A aplicação dos princípios enxutos no processo de desenvolvimento de software requer interpretação, e há mais de uma escola de pensamento sobre a melhor interpretação do Desenvolvimento Enxuto de Software. Alguns se concentram nos princípios enxutos aplicados às práticas comuns de desenvolvimento, alguns se concentram no gerenciamento do fluxo de trabalho, e outros se concentram nos processos complementares do desenvolvimento de produtos utilizados pela Toyota e outras empresas enxutas (LADAS, 2009).

2.3.1 O Mapeamento dos Princípios Enxutos

O primeiro mapeamento dos princípios enxutos para o desenvolvimento de software foi proposto por Mary e Tom Poppendieck (2003). Pode-se considerar a primeira escola de pensamento do desenvolvimento enxuto de software, onde os princípios enxutos são interpretados em termos de metodologia de desenvolvimento de software nativo.

Segundo Ladas (2009), esta abordagem de desenvolvimento enxuto de software tem sido atraente para alguns que acreditam que o software é fundamentalmente uma disciplina artesanal. O movimento de softwares ágeis têm sido os defensores mais visíveis da filosofia artesanal nos últimos anos. A parte da comunidade que adotou o Pensamento Enxuto (*Lean Thinking*) normalmente chama esta abordagem de *Lean/Agile Development*. Proponentes de *Lean/Agile* tipicamente descrevem práticas de métodos *Scrum* ou *Extreme Programming* em termos de princípios enxutos.

Os sete princípios identificados por Mary e Tom Poppendieck (2003) são:

- Eliminar o desperdício;
- Construir com qualidade;
- Criar conhecimento;
- Adiar compromisso;
- Entregue rápido;

- Respeito às pessoas;
- Aperfeiçoar o todo.

2.3.1.1 Eliminar o Desperdício

Tudo que não agrega valor na perspectiva do cliente é desperdício e deve ser eliminado. Os sete tipos de desperdícios no desenvolvimento de software são:

Defeitos	Defeitos que não são rapidamente identificados nos testes.
Funcionalidades extras	Funcionalidades que não serão utilizadas pelos usuários.
Transferência	Conhecimento tácito que são perdidos durante a transferência de trabalho.
Atrasos	Esperas por informação, documentação.
Trabalho parcialmente finalizado	Trabalhos iniciados, mas não finalizados. “Inventário” no processo de desenvolvimento.
Chaveamento de tarefa	Interrupções e chaveamento de tarefas causam perda de produtividade.
Processos desnecessários	Documentos que não são lidos. Tarefas manuais que poderiam ser automatizadas.

2.3.1.2 Construir com Qualidade

Na produção enxuta cada etapa do processo deve ser a prova de erros e auto-inspecionado. Quando um problema é detectado, a linha de montagem é interrompida até que a causa raiz do problema seja encontrado e corrigido, para que não haja a repetição do problema.

No desenvolvimento de software, o código deve ser a prova de erros utilizando-se uma abordagem orientada a testes. Testes unitários, testes de integração, testar freqüentemente utilizando-se de testes automatizados para prevenir mudanças de códigos por erros não detectados.

2.3.1.3 Criar Conhecimento

Não esquecer as lições aprendidas. Encontrar meios de registrar o conhecimento da equipe que permita uma fácil localização na próxima vez que seja necessária.

Por exemplo, se uma nova funcionalidade implementada exigiu a leitura de todo código para entender como o subsistema funciona. O que foi aprendido deveria ser registrado em algum lugar. Isso poderia ser adicionado em documento detalhado do sistema, mas seria mais eficiente se fosse registrado como um comentário no código.

Quando se está criando uma arquitetura, um projeto, ou um código, constantemente será necessário considerar alternativas para a tomada de decisão. A escolha por uma alternativa ou outra deveria ser registrada, porque este conhecimento poderá poupar muito tempo no futuro, mas outras vezes será um exagero. Deve-se criar um balanço do que é necessário registrar, aprendendo constantemente a fazer este julgamento.

2.3.1.4 Adiar Compromisso

As melhores decisões são feitas quando se tem mais informações disponíveis. Desta forma é melhor esperar o último momento responsável para fazer uma decisão irreversível.

Por exemplo, quando se precisa escolher a arquitetura para um sistema, primeiro determina-se quando é o último momento responsável para fazer esta decisão. Utiliza-se este tempo para acumular conhecimento sobre as reais necessidades de outros componentes do sistema, e explorar as características das alternativas de escolha.

2.3.1.5 Entregue Rápido

Entregue rápido significa desenvolver funcionalidades em lotes pequenos que são entregues rapidamente, em iterações pequenas. Estas funcionalidades pode ser implementadas e entregues antes que requisitos associados possam mudar. Isto significa que o cliente tem a oportunidade de usar estas funcionalidades e dar o *feedback* que possa mudar outro requisito antes que sejam implementadas.

A conclusão de cada pequena iteração provê a oportunidade de mudar e re-priorizar os requisitos baseado no uso e no *feedback* real. O resultado final é um produto que se encontra mais estreitamente com as reais necessidades do cliente, enquanto elimina os desperdícios e retrabalhos de requisitos voláteis.

2.3.1.6 Respeito às Pessoas

Respeito às pessoas significa confiar que eles conhecem a melhor maneira de fazer o seu trabalho, engajando-os a expor falhas no processo atual, e encorajando-os a encontrar formas de melhorar seus trabalhos e os processos. Respeito às pessoas significa reconhecê-los por suas realizações e ativamente solicitando os seus conselhos.

2.3.1.7 Aperfeiçoar o Todo

Muitas teorias de como gerenciar um projeto de software são baseadas na teoria da desagregação: quebre o conjunto em partes individuais e otimize cada um. O pensamento enxuto sugere que otimizações individuais sempre levam para a sub-otimização do sistema como um todo.

A melhor maneira de evitar a sub-otimização e incentivar a colaboração é fazer as pessoas responsáveis por aquilo que pode influenciar, não apenas o que eles podem controlar. Isso significa medir o desempenho de um nível superior ao que se espera. Medir a equipe por contagem de defeitos, não a dos indivíduos. Para

alguns, parece injusto manter uma equipe responsável por cada desempenho, mas as organizações enxutas descobriram que os indivíduos são raramente capazes de mudar o sistema que influenciam o desempenho deles. No entanto, uma equipe, trabalhando em conjunto e responsável por seus próprios processos, pode e vai fazer melhorias consistentes.

2.3.2 A Escola do Fluxo de Trabalho

A escola do fluxo de trabalho é outra escola de pensamento enxuto no desenvolvimento de software. O foco desta escola descreve a maioria dos processos de desenvolvimento de software em termos de fluxo de trabalho e qualquer desses processos do fluxo de trabalho está sujeito aos cincos princípios do Pensamento Enxuto, sem a necessidade de se abstrair os detalhes.

Uma das metodologias derivadas desta escola é a que utiliza o Sistema *Kanban* para gestão do fluxo de trabalho, popularizada por David Anderson (2009). Da mesma forma que na produção enxuta, o Sistema *Kanban* é o elemento principal para a operação e controle de um fluxo contínuo e de um sistema puxado, mas neste caso aplicado a gestão do desenvolvimento de software.

Este trabalho explora a utilização do Sistema *Kanban* para a gestão do desenvolvimento de software combinado com as práticas de desenvolvimento do *Extreme Programming* para definir uma metodologia de software que estabeleça o Desenvolvimento Enxuto de Software.

2.4 Metodologias Ágeis

As metodologias ágeis surgiram ao final da década de 90, por desenvolvedores que estavam insatisfeitos com as abordagens pesadas que predominavam no cenário do desenvolvimento de software.

As abordagens pesadas de desenvolvimento eram apoiadas por uma visão geral que a melhor maneira de obter o melhor software era por meio de um cuidadoso planejamento de projeto, garantia de qualidade formalizada, uso de

métodos de análise e projeto apoiados por ferramentas CASE e controlados por um rigoroso processo de desenvolvimento de software. Essa visão esteve ligada aos softwares grandes e de vida longa, desenvolvidos por equipes grandes e por longos períodos de tempo.

Essa abordagem pesada de desenvolvimento aplicada a desenvolvimentos de sistemas de pequenas e médias empresas mostraram-se inviáveis porque muitas vezes o tempo gasto para se determinar como o sistema deveria ser desenvolvido era maior do que o empregado no desenvolvimento do programa e em testes. E à medida que os requisitos de sistema mudavam, o retrabalho era essencial e a especificação e o projeto tinham que mudar com o programa.

Ao final da década de 90 alguns desenvolvedores começaram a propor novos métodos ágeis de desenvolvimento de software para se adaptar a nova realidade de mercado, um ambiente de negócios de rápidas mudanças. Estes métodos permitiam que a equipe de desenvolvimento se concentrasse mais no software do que em seu projeto e documentação, utilizando uma abordagem iterativa para especificação, desenvolvimento e entrega de software.

O método ágil mais conhecido é o *Extreme Programming*, ou simplesmente XP, proposto por Kent Beck. Outros métodos ágeis incluem o *Scrum*, *Crystal*, *Adaptive Software Development*, *Dynamic Systems Development Method (DSDM)* e *Feature Driven Development (FDD)*.

Apesar de todos esses métodos possuírem processos diferentes, todos são baseados na noção de desenvolvimento e entrega incrementais e compartilham um conjunto de princípios conhecidos por Manifesto Ágil.

O Manifesto Ágil, ou formalmente, Manifesto para o Desenvolvimento Ágil de Software foi definido por 17 líderes de desenvolvimento de software que decidiram se reunir em uma estação de esqui em Utah, nos Estados Unidos, para discutir formas de melhorar o desempenho de seus projetos. Desta reunião eles encontraram uma série de princípios comuns descrito a seguir:

"Estamos descobrindo maneiras melhores de desenvolver software fazendo-o nós mesmos e ajudando outros a fazê-lo. Através desse trabalho, passamos a valorizar:

- *Indivíduos e interação entre eles mais que processos e ferramentas;*
- *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, valorizamos mais os itens à esquerda."

Além do Manifesto Ágil, foram decididos nesta reunião o uso do termo "Ágil" e a criação do *Agile Alliance*, uma organização sem fins lucrativos para promover o desenvolvimento e a disseminação de informação sobre os processos "Ágeis".

Os quatro princípios originais do Manifesto Ágil foram refinados por seus autores gerando doze princípios:

- *Nossa maior prioridade é satisfazer o cliente, através da entrega adiantada e contínua de software de valor.*
- *Aceitar mudanças de requisitos, mesmo no fim do desenvolvimento. Processos ágeis se adéquam a mudanças, para que o cliente possa tirar vantagens competitivas.*
- *Entregar software funcionando com freqüência, na escala de semanas até meses, com preferência aos períodos mais curtos.*
- *Pessoas relacionadas a negócios e desenvolvedores devem trabalhar em conjunto e diariamente, durante todo o curso do projeto.*
- *Construir projetos ao redor de indivíduos motivados. Dando a eles o ambiente e suporte necessário, e confiar que farão seu trabalho.*
- *O Método mais eficiente e eficaz de transmitir informações para, e por dentro de um time de desenvolvimento, é através de uma conversa cara a cara.*
- *Software funcional é a medida primária de progresso.*
- *Processos ágeis promovem um ambiente sustentável. Os patrocinadores, desenvolvedores e usuários, devem ser capazes de manter indefinidamente, passos constantes.*
- *Contínua atenção a excelência técnica e bom design aumentam a agilidade.*
- *Simplicidade: a arte de maximizar a quantidade de trabalho que não precisou ser feito.*
- *As melhores arquiteturas, requisitos e projetos emergem de times auto-organizáveis.*
- *Em intervalos regulares, o time reflete em como ficar mais efetivo, então, se ajustam e otimizam seu comportamento de acordo.*

2.4.1 Extreme Programming

O *Extreme Programming (XP)* é uma das metodologias ágeis mais conhecidas. O *XP* define a codificação como a principal atividade de um projeto de software, utilizando uma abordagem de desenvolvimento que combina princípios e práticas usadas por muitos desenvolvedores, mas levadas a níveis extremos.

Kent Beck (2004) justifica o termo *"Extreme"* da seguinte forma:

- *Se revisar o código é bom, revisaremos código o tempo inteiro (programação em pares);*
- *Se testar é bom, todos vão testar o tempo inteiro (testes de unidade), até mesmo os clientes (testes funcionais);*
- *Se o projeto é bom, ele fará parte das funções diárias de todos (refatoração);*
- *Se simplicidade é bom, sempre deixaremos o sistema com o projeto mais simples que suporte a funcionalidade atual (a coisa mais simples que possa funcionar);*
- *Se arquitetura é importante, todos trabalharão para definir e refinar a arquitetura o tempo inteiro (metáfora);*
- *Se testes de integração são importantes, então vamos integrar e testar várias vezes ao dia (integração contínua);*
- *Se iterações curtas são boas, faremos iterações muito, muito pequenas – segundos, minutos e horas, não semanas, meses e anos (o Jogo do Planejamento).*

O *XP* é uma metodologia que está organizada em torno de valores e práticas que atuam de forma harmônica e coesa para assegurar que o cliente sempre receba o máximo de valor de cada dia de trabalho da equipe de desenvolvimento de software.

2.4.1.1 Os Valores do *XP*

Os quatro valores fundamentais do *XP* são:

- ***Comunicação***. Muitos problemas nos projetos estão ligados a falhas na comunicação entre a equipe de desenvolvimento e também da equipe com o cliente. Mudanças no projeto não são comunicadas entre

os membros da equipe, ou uma pergunta essencial não feita para o cliente acaba prejudicando uma decisão importante. O *XP* procura manter a comunicação fluindo através de práticas que não podem ser feitas sem comunicação, como o teste de unidade, programação em pares e a estimativa de tarefas. O efeito de testar, programar em pares e estimar é a comunicação entre programadores, clientes e gerentes.

- ***Simplicidade***. O *XP* aposta que é melhor fazer uma coisa simples hoje e pagar um pouco mais amanhã para fazer alguma modificação nela se for necessário do que fazer uma coisa mais complicada hoje que talvez nunca será usada.
- ***Feedback***. O *feedback*, ou realimentação, acontece constantemente no desenvolvimento *XP*, para que os problemas sejam evidenciados e rapidamente corrigidos para serem incorporados ao sistema. Seja através de testes unitários, avaliações dos programadores das estórias escritas pelos clientes, testes de funcionalidade, integração contínua e de colocar o sistema em produção assim que for possível.
- ***Coragem***. É necessário coragem para apontar uma falha na arquitetura essencial de um sistema, solicitar ajuda quando necessário, simplificar um código já esteja funcionando, negociar o escopo do projeto para atender o prazo.

2.4.1.2 As Práticas do *XP*

O *XP* reúne doze práticas para o desenvolvimento de software que se apóiam umas as outras:

- ***O Jogo do Planejamento***. O projeto em *XP* é dividido em releases e iterações.Releases são módulos do sistema que geram um valor bem definido para o cliente. Iterações são ciclos de poucas semanas, em que a equipe implementa um conjunto de funcionalidades acordado pelo cliente. No início de cada release e iteração ocorre o Jogo do Planejamento, que é uma reunião onde o cliente avalia as

funcionalidades que devem ser implementadas e prioriza aquelas que farão parte do próximo release ou da próxima iteração.

- ***Entregas Freqüentes***. A equipe produz um conjunto mínimo de funcionalidades e as coloca em produção rapidamente para que o cliente já utilize o software beneficiando-se dele. Releases do sistema são freqüentes e adicionam mais funcionalidades, agregando mais valor ao sistema.
- ***Metáfora***. Cada projeto em *XP* é guiado por uma única metáfora abrangente, ajudando a transmitir idéias complexas de forma simples, através de uma linguagem comum que é estabelecida entre a equipe de desenvolvimento e o cliente.
- ***Projeto Simples***. O projeto é realizado para atender os requisitos atuais do sistema. Não são criadas generalizações dentro do código, de modo a prepará-lo para possíveis necessidades futuras. Os desenvolvedores se baseiam na premissa que serão capazes de incorporar qualquer necessidade futura quando e se ela surgir.
- ***Testes***. O *XP* utiliza a técnica de desenvolvimento guiado pelos testes. Os desenvolvedores escrevem teste de unidade para que sua confiança na operação do sistema possa se tornar parte do sistema em si. Os clientes escrevem testes de funcionalidade para que sua confiança na operação do sistema possa também se tornar parte do sistema. O resultado é um sistema que se torna cada vez mais confiável com o tempo, capaz de aceitar modificações e não menos.
- ***Refatoração***. A refatoração é a melhoria do código sem afetar a funcionalidade que ele implementa, retirando duplicações de código, simplificando a implementação.
- ***Programação em Pares***. Dois desenvolvedores trabalham juntos no mesmo código e no mesmo computador. Desta forma o código é revisado permanentemente, enquanto é construído. Os desenvolvedores se complementam gerando um código mais simples e eficaz.
- ***Propriedade Coletiva***. A qualquer momento, qualquer um que perceba uma oportunidade de acrescentar valor a alguma parte do código é obrigado a fazê-lo. No *XP* todos são responsáveis pelo sistema inteiro,

gerando maior agilidade ao processo e cria mais um mecanismo de revisão e verificação do código.

- ***Integração Contínua***. Uma nova funcionalidade incorporada ao sistema pode afetar as outras que já estavam funcionando. Desta forma os pares integram seus códigos com o sistema, várias vezes ao dia, executando os testes para garantir que a integração ocorreu sem problemas.
- ***Semana de 40 Horas***. A semana de 40 horas é uma referência para indicar sintomas de problemas dentro do projeto. A regra do *XP* é não trabalhe uma segunda semana com horas extras. Dificilmente as pessoas continuarão dispostas, criativas, cuidadosas e confiantes em seguidas semanas de horas extras. Isto afetará a qualidade do desenvolvimento.
- ***Cliente Presente***. No *XP* o cliente participa ativamente do processo de desenvolvimento, ficando disponível para responder questões, resolver disputas e definir prioridades, conduzindo o desenvolvimento a partir do feedback que recebe do sistema.
- ***Padrões de Codificação***. Com a programação em pares e o código coletivo é essencial que a equipe possua padrões de codificação para que o sistema se torne mais homogêneo e permita que qualquer manutenção futura seja efetuada mais rapidamente.

2.4.1.3 O Processo *XP*

No processo *XP* os clientes participam ativamente do desenvolvimento, especificando e priorizando os requisitos do sistema. Todos os requisitos são expressos como cenários, ou estórias de usuário, que são discutidos com os membros da equipe.

O cliente e a equipe de desenvolvimento desenvolvem “cartões de estórias” que englobam as necessidades do cliente. A equipe de desenvolvimento dividirá cada estória em tarefas e estimará o esforço e os recursos necessários para a implementação. O cliente prioriza as estórias para implementação, escolhendo as

que podem ser usadas imediatamente para proporcionar maior valor ao negócio. Se os requisitos mudam, as estórias que não foram implementadas mudam ou são descartadas. Se forem necessárias mudanças em um sistema que já foi entregue, novos cartões de estórias são desenvolvidos e, novamente, o cliente decide se essas mudanças devem ter prioridade sobre a nova funcionalidade.

Novas versões do sistema podem ser compiladas e integradas várias vezes ao dia e os incrementos são entregues para os clientes aproximadamente a cada duas semanas (iterações). Uma iteração do sistema só será aceita se todos os testes forem executados com sucesso.

3 O *Extreme Programming* e o *Kanban* no Desenvolvimento de Software

A produção enxuta surgiu com métodos mais eficazes de manufatura em contraponto a produção em massa baseada em processamento de grandes lotes. Da mesma forma metodologias ágeis surgiram com métodos mais eficazes de desenvolvimento de software para atender um mercado de rápidas mudanças, que métodos tradicionais, como o processo cascata, não se adequavam.

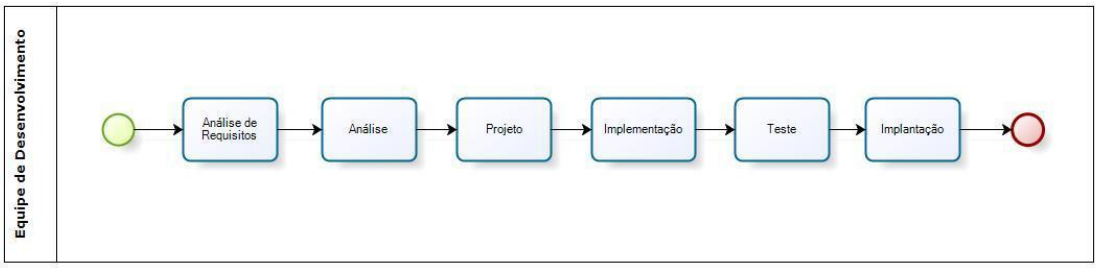

Figura 6 - BPMN de um processo cascata.

O processo cascata possui uma semelhança com a produção em massa, onde o fluxo de trabalho é organizado em etapas bem definidas onde as atividades são realizadas em seqüência. Todos os passos de uma etapa são concluídos para que o próximo inicie, ou seja, como na produção em massa os itens de trabalho são processados em lotes.

Já metodologias ágeis como o *Extreme Programming*, que usam uma abordagem iterativa e incremental de desenvolvimento se assemelham a uma célula de trabalho de fluxo unitário da produção enxuta. Cada estória é desenvolvida por um par efetuando as atividades de análise, projeto, implementação e testes.

Como o *Extreme Programming* define a codificação como principal atividade de um projeto de software, a maioria de suas práticas tem o foco no desenvolvimento de software, não enfatizando os aspectos de gestão de projeto.

O objetivo deste trabalho é propor o uso do Sistema *Kanban* como elemento de gestão de projeto de software que utilize as práticas de desenvolvimento de software do *Extreme Programming*.

3.1 Combinando o Extreme Programming com o Sistema Kanban

Em ambientes de equipes que utilizam metodologias ágeis como *Extreme Programming* e *Scrum* pode-se encontrar quadros com cartões sinalizando o estado (ex.: “não iniciadas”, “em andamento” e “finalizadas”) do desenvolvimento de funcionalidades de um sistema que muitos chamam de *kanban*. Neste caso o nome *kanban* está apenas associado ao uso de cartões. Apesar de serem utilizados como ferramenta de gestão, não se pode chamar este quadro com cartões de um Sistema *Kanban*.

O Sistema *Kanban* no desenvolvimento de software tem a mesma função que na produção enxuta, ou seja, é utilizado para amortecer as demandas e permitir o estabelecimento de um fluxo contínuo e de um sistema puxado de entregas unitárias.

Os principais elementos de um Sistema *Kanban* são:

- Limite definido de trabalhos em processo (*work-in-process*);
- Cartões de sinal são criados para representar esse trabalho;
- Os cartões são usados para puxar o trabalho através do sistema.

A figura 7 mostra o BPMN para a implantação inicial de um Sistema *Kanban*.

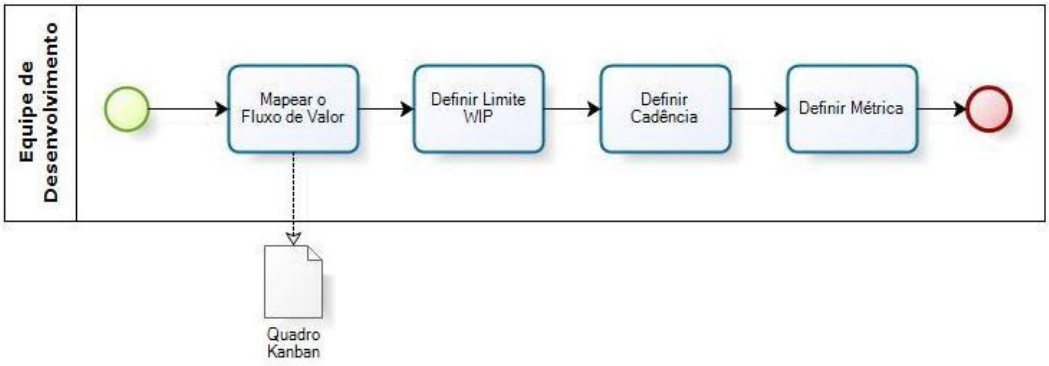

Figura 7 – BPMN para a implantação do Sistema *Kanban*.

Um Sistema *Kanban* pode ser implantando em qualquer processo de desenvolvimento de software, mas neste caso será mostrada a implantação com o *Extreme Programming*.

3.1.1 Mapear o Fluxo de Valor

A etapa inicial para a implantação de um Sistema *Kanban* é definir o tipo de trabalho valorizado pelo cliente e mapear o fluxo de valor dos itens de trabalho. No *Extreme Programming* pode-se definir que as estórias de usuário são um tipo de item de trabalho que será controlado.

O mapeamento do fluxo de valor é feito identificando a série de estados em que o item de trabalho, neste caso a estória de usuário, atravessa ao longo da cadeia de valor.

A figura 6 mostra a cadeia de valor de um processo cascata. O tempo total de desenvolvimento é a soma dos tempos de cada etapa, mais os tempos de espera causados pela transição de cada etapa. O cliente só perceberá o valor do software ao final do processo quando terá o software pronto.

Já no processo utilizando *XP*, procura-se maximizar o valor do software, através de ciclos de desenvolvimentos iterativos e incrementais, entregando software funcional o mais cedo possível, priorizando os requisitos mais importantes para os negócios.

A identificação dos estados pelo qual as estórias fluem no desenvolvimento utilizando *XP* deve ser feita analisando-se o seu processo.

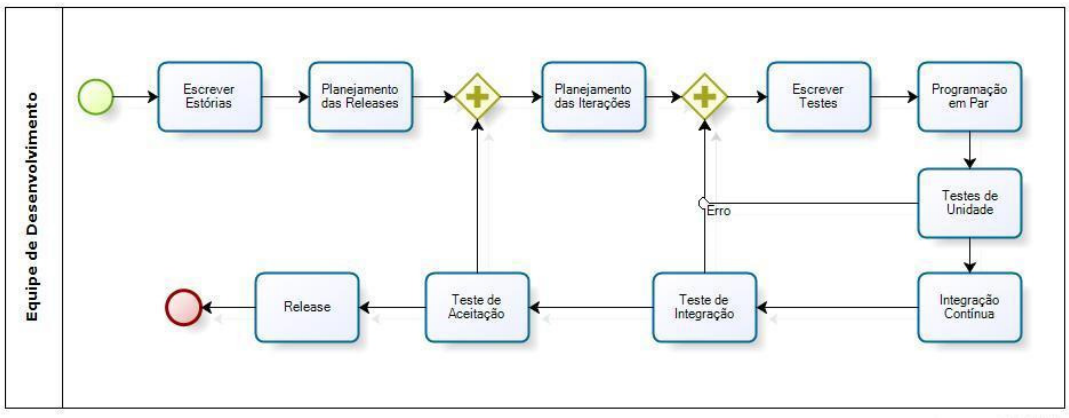

Figura 8 – BPMN do Extreme Programming.

O importante é se concentrar nos estados em que a estória atravessa e não nas especializações da força de trabalho ou nas entregas ao longo da cadeia de valor.

Com os estados definidos pode-se montar o quadro *Kanban* desenhando o fluxo de valor da esquerda para a direita, com uma série de colunas. Cada coluna representa um estado do fluxo de trabalho. Na figura 9 está à representação do quadro *Kanban* aplicado ao *XP*.

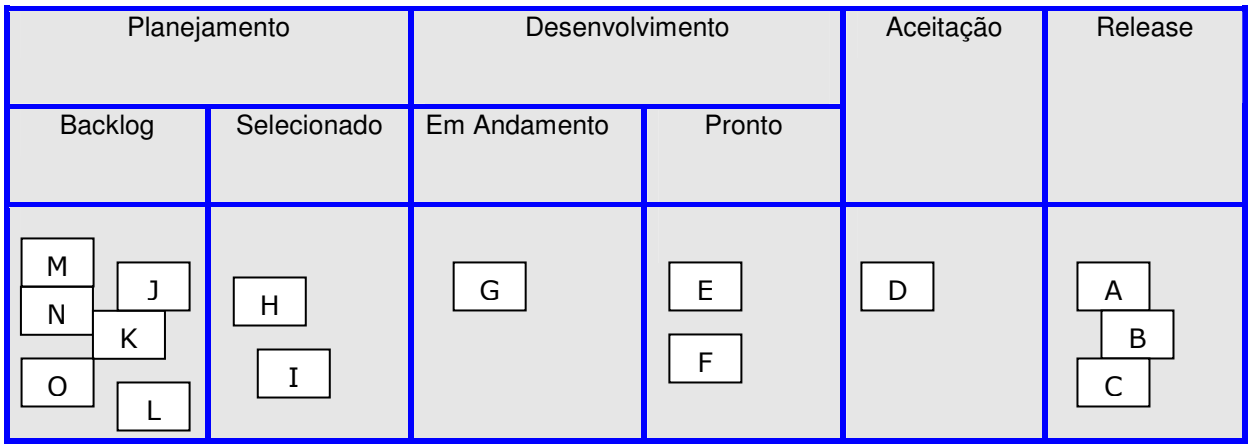

Figura 9 – Quadro Kanban.

A coluna "*Planejamento*" representa o Jogo do Planejamento do XP e está dividido em dois estados, o "*Backlog*" e "*Selecionado*". O termo "*Backlog*" foi "emprestado" do *Scrum* e representa a lista de desejos do cliente onde cada cartão representa uma estória escrita pelo cliente. Na coluna "*Selecionado*" do "*Planejamento*" estão as estórias que foram puxadas do "*Backlog*" e priorizadas para serem trabalhadas na iteração. A coluna "*Desenvolvimento*" foi dividida em duas colunas para representar o estado "*Em Andamento*", que engloba todas as fases do desenvolvimento *XP* (escrever testes, codificar, testar, integrar e testar a integração) e o estado "*Pronto*" que identificam as estórias que estão implementadas e testadas. Da coluna "*Pronto*" as estórias são puxadas para o teste de aceitação com o cliente. As estórias aprovadas são colocadas na coluna "Release" identificando que estão disponíveis para a entrega.

3.1.2 Definir os Limites dos Trabalhos em Processo

O passo seguinte para a implantação de um Sistema Kanban é a definição dos limites dos trabalhos em processo (limite WIP) de cada etapa do fluxo de trabalho. A definição dos limites é essencial para que se ajuste a demanda à capacidade de desenvolvimento da equipe, para que não se tenha sobrecarga, identifique os gargalos do processo, se implemente o sistema puxado e se estabeleça um processo de fluxo contínuo.

A definição precisa dos limites WIP não é fácil de determinar, e de um modo geral em sistemas enxutos, estes limites são ajustados empiricamente em ciclos de melhoria contínua. Uma regra inicial simples é definir que cada pessoa só irá trabalhar com um item de cada vez. Como este trabalho utiliza-se das práticas do *XP*, um par representa um recurso. Por exemplo, se há três duplas de desenvolvimento o limite WIP para o estado "*Desenvolvimento*" será definido em três.

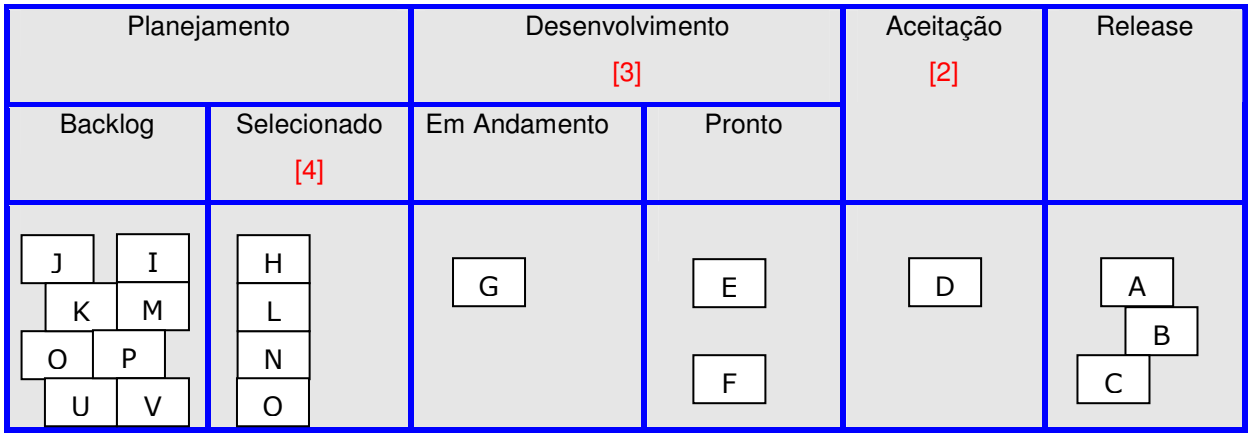

Figura 10 – Quadro Kanban com os limites de trabalhos em processo definidos.

Na coluna "*Backlog*" não há limite WIP definido porque ele só representa a lista de desejos do cliente, ou seja, as estórias que foram escritas pelo cliente. Na coluna "*Selecionado*" há um limite WIP de quatro que definem os quatro itens de maior prioridade que podem ser puxados para o desenvolvimento. A coluna "Selecionado" funciona como um estoque amortecedor (*buffer*) para garantir que tenham itens suficientes para que o fluxo não tenha interrupções ou espera durante o desenvolvimento. As colunas "*Em Andamento*" e "*Pronto*" compartilham o limite

WIP de três definido para o desenvolvimento. Isto significa que um par só poderá puxar uma nova estória para o desenvolvimento se o limite não for excedido, mesmo que existam estórias no estado "*Pronto*". Isto incentiva a equipe a olhar todo o processo e ajudar a equipe de testes de aceitação para identificar algum problema que esteja acontecendo na "*Aceitação*" que esteja impedindo que puxe uma estória que esteja no estado "*Pronto*".

3.1.3 Definir a Cadência de Entrega

A cadência de entrega em projetos XP é definida em uma estratégia conhecida como "*timeboxed*", ou seja, o cronograma de entregas é dividido em um número de períodos de tempo definido. O software será entregue de forma incremental, de modo que após cada entrega o cliente possa começar a utilizar o sistema e obter os benefícios que ele oferece. Desta forma os desenvolvedores poderão receber o *feedback* dos usuários finais do sistema, permitindo que se façam ajustes para aprimorar a qualidade dos releases subseqüentes. Essas entregas são os *releases* em *XP*.

Os projetos em *XP* procuram trabalhar com o conceito de releases pequenos, que preferencialmente devem ter em torno de dois meses. Mesmo pequenos, um *release* representa um tempo muito longo para o desenvolvimento em *XP*. E por esta razão, cada release é dividido em unidades de tempo menores, conhecidas como iteração. Normalmente, uma iteração pode variar de uma a três semanas dependendo das características do projeto. Uma vez definido o tamanho de uma iteração, deve-se, preferencialmente, mantê-lo inalterado ao longo de todo projeto, facilitando a gerência e o planejamento.

Em muitos projetos, nem sempre o tamanho de uma estória se encaixa bem no tempo definido para cada iteração. Algumas vezes as estórias são muito pequenas e levam algumas horas, tal como fazer uma mudança em uma funcionalidade existente. Algumas vezes as estórias são difíceis de dividi-las em tamanhos que se encaixem em uma iteração. Algumas vezes as estórias variam muito de tamanho em relação a outras. Tudo isso torna o trabalho de definir a melhor duração das iterações muito difícil.

Em contrapartida a abordagem "*timeboxed*" do XP, o Sistema *Kanban* pode trabalhar sem a estratégia de tempo definido das iterações. O desenvolvimento continua sendo iterativo, mas o Sistema *Kanban* desacopla a priorização (entrada do sistema) da entrega (saída). Uma estória em um Sistema *Kanban* pode levar várias semanas para ser processado. O limite não está na duração do desenvolvimento da estória e sim na quantidade de estórias que estão em desenvolvimento.

Se por exemplo for definida uma cadência de entrega com o cliente para duas semanas, isto não implica que a estória deve ser concluída em duas semanas, apenas os itens na fila de "*Release*" serão entregues. Todas as outras estórias continuarão em processo para ser entregue em uma versão futura.

3.1.4 Métricas

Em qualquer tipo de projeto são necessárias métricas para se conseguir uma correta gestão do processo. Tanto no *XP*, quanto no Sistema *Kanban* as métricas são relacionadas à medida de capacidade. Ou seja, é importante conhecer a capacidade para atender as demandas sem causar sobrecarga no processo. Forçar o processo a trabalhar acima de sua capacidade causará turbulências e as coisas acontecerão mais devagar.

Uma medida de capacidade em sistemas iterativos como o *XP* é a velocidade, isto é, quantas estórias podem ser regularmente entregues durante o tamanho da iteração escolhida. Como os tamanhos das estórias variam, a velocidade é medida em termos de pontos. Cada estória tem atribuído uma quantidade de pontos em função do seu nível de dificuldade. Uma estória deveria variar entre um e três pontos ou seis pontos, mas não mais que isso. Estórias maiores deveriam ser quebradas em estórias menores. A equipe deveria completar o mesmo número de pontos por iteração. Por exemplo, uma equipe pode ter contabilizado 40 pontos por iteração. Essa velocidade define a capacidade desta equipe. É importante ressaltar que velocidade é uma medida de capacidade e não uma medida de desempenho.

A medida de capacidade do Sistema *Kanban* está relacionada à quantidade e a idade dos trabalhos em processo (*WIP*). A taxa de transferência total dos itens

entregues em um dado período de tempo é uma medida de capacidade. Esta taxa de transferência é chamada de *throughput*.

Throughput = *WIP* / Tempo de Ciclo

O tempo de ciclo é a duração para completar um processo. No nosso caso é o tempo gasto pela estória percorrer do estado “Selecionado” até o estado “Release” do Sistema *Kanban*. Este tempo de ciclo algumas vezes é chamado de *lead time*.

É possível reduzir o tempo de ciclo reduzindo a quantidade de trabalhos em processo (*WIP*). O ideal de um sistema enxuto é alcançar o fluxo unitário de peças, ou seja, o tamanho do lote é igual a um. Somente há um único item sendo processado em cada etapa do fluxo de trabalho. Como o tempo de processamento de cada trabalho pode sofrer variações é necessária a utilização de filas, ou estoques de amortecimento (*buffers*) para garantir o fluxo contínuo. Quanto mais estável o processo ficar, menos filas serão necessárias e conseqüentemente o tempo de ciclo diminuirá, tornando o processo mais eficaz.

3.1.5 A Gestão Visual

A Gestão Visual está ligada a utilização de controles visuais. Os controles visuais são quaisquer dispositivos de comunicação utilizados no ambiente de trabalho para informar rapidamente como o trabalho dever ser executado e se há algum desvio de padrão. Auxilia a todos que desejam fazer um bom trabalho a ver imediatamente como o estão executando. Pode mostrar a que categoria os itens pertencem, quantos itens devem constar naquela categoria, qual procedimento padrão para uma determinada tarefa, e muitos outros tipos de informações importantes para o fluxo de atividades de trabalho. O controle visual está ligado à criação de informações *just-in-time* de todos os tipos para garantir a execução rápida e adequada de operações e de processos.

O Sistema Kanban é um bom exemplo de controle visual. O quadro *Kanban* dá um rápido *feedback* a todos sobre o estado do desenvolvimento. A utilização dos cartões para a operação do sistema puxado melhora os aspectos de auto-

gerenciamento das equipes. Problemas podem ser visualizados para que sejam solucionados o mais rápido possível. Todos visualizam a mesma informação, sejam clientes, gerentes ou desenvolvedores, melhorando o processo de comunicação.

As informações no quadro *Kanban* podem ser enriquecidas com a utilização de mais recursos visuais. Por exemplo, os cartões que representam as estórias podem ser classificadas por cores para identificar tipos diferentes de funcionalidades que representam. Cartões verdes poderiam representar estórias que agregam valor diretamente ao sistema (novas funcionalidades). Cartões vermelhos poderiam representar estórias necessárias para a correção de problemas no sistema. E cartões amarelos poderiam representar estórias para melhorias no sistema.

Outras informações importantes para enriquecer o controle visual são os gráficos. Um gráfico normalmente utilizado em metodologias ágeis é o gráfico de *burndown*. O gráfico de *burndown* mostra quanto trabalho ainda resta para concluir a iteração. Sua principal proposta é mostrar facilmente o mais cedo possível se o cronograma está sendo cumprido ou não para que sejam tomadas as ações cabíveis se necessário.

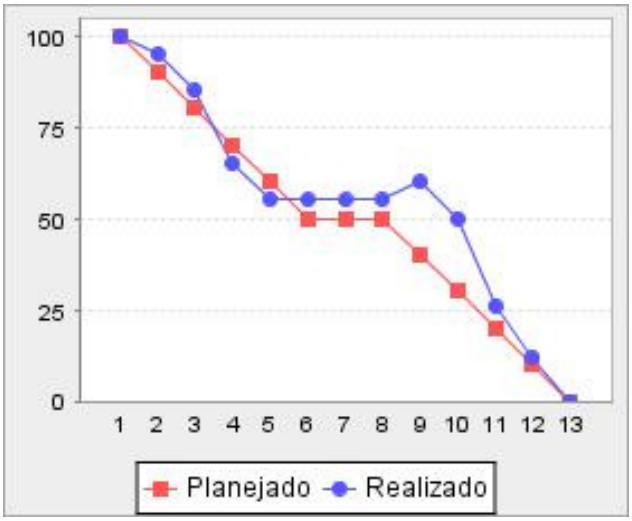

Figura 11 – Gráfico de burndown.

Um gráfico utilizado em sistemas enxutos é o *CFD*, ou *Cumulative Flow Diagram* (Diagrama de Fluxo Acumulado). Este gráfico mostra o quanto o fluxo está nivelado e como o WIP afeta o tempo de ciclo.

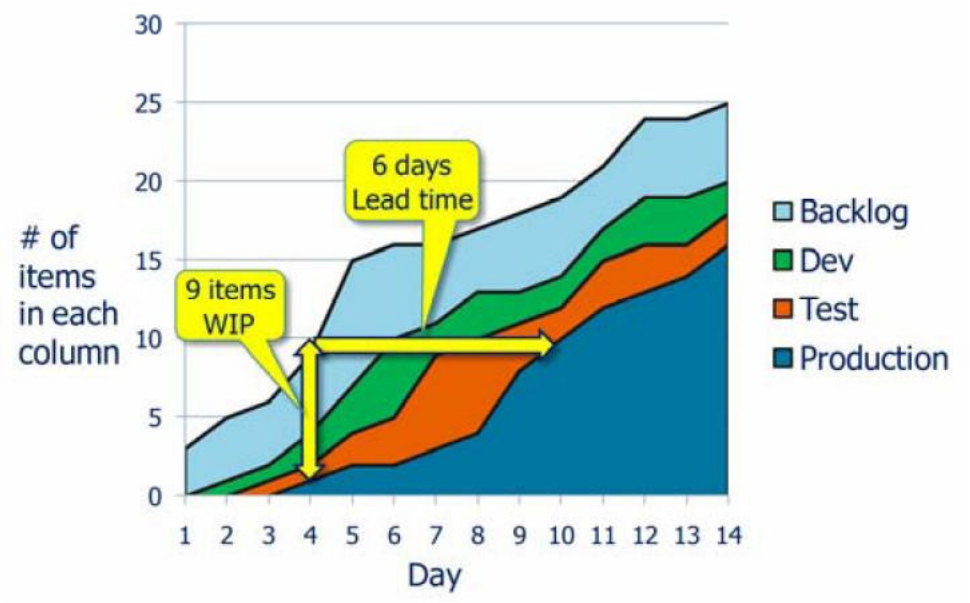

Figura 12 – Gráfico CFD (*Cumulative Flow Diagram*).

Neste exemplo, todo dia o total de número de itens em cada coluna do quadro *Kanban* está empilhado no eixo Y. No dia 4 há 9 itens no quadro. Iniciando da coluna mais a direita há 1 item em Produção, 1 item em Teste, 2 em Desenvolvimento e 5 no Backlog. A seta vertical e horizontal mostra a relação entre o *WIP* e o *lead time* ou tempo de ciclo.

A seta horizontal nos mostra que os itens adicionados ao *Backlog* no dia 4 demoraram, em média 6 dias para chegar a Produção. Cerca de metade desse tempo foi em Teste. Pode-se ver que se fosse limitado o WIP no Teste e no Backlog haveria redução significativa no *lead time*.

O declive da área azul-escura nos mostra a velocidade (ou seja, número de itens implantado por dia). Ao longo do tempo, podemos ver como maior velocidade reduz o *lead time*, enquanto aumentar o *WIP* aumenta o *lead time*.

3.1.6 Operação do Sistema Kanban

No modelo proposto, a operação do Sistema *Kanban* está centrada no quadro *Kanban* e nos cartões que representam as estórias escritas pelo cliente. Os cartões

fluem pelos estados que representam o desenvolvimento baseado no *Extreme Programming*.

As figuras abaixo mostram, em BPMN, o ciclo de desenvolvimento baseado no Extreme Programming utilizando o Sistema *Kanban*.

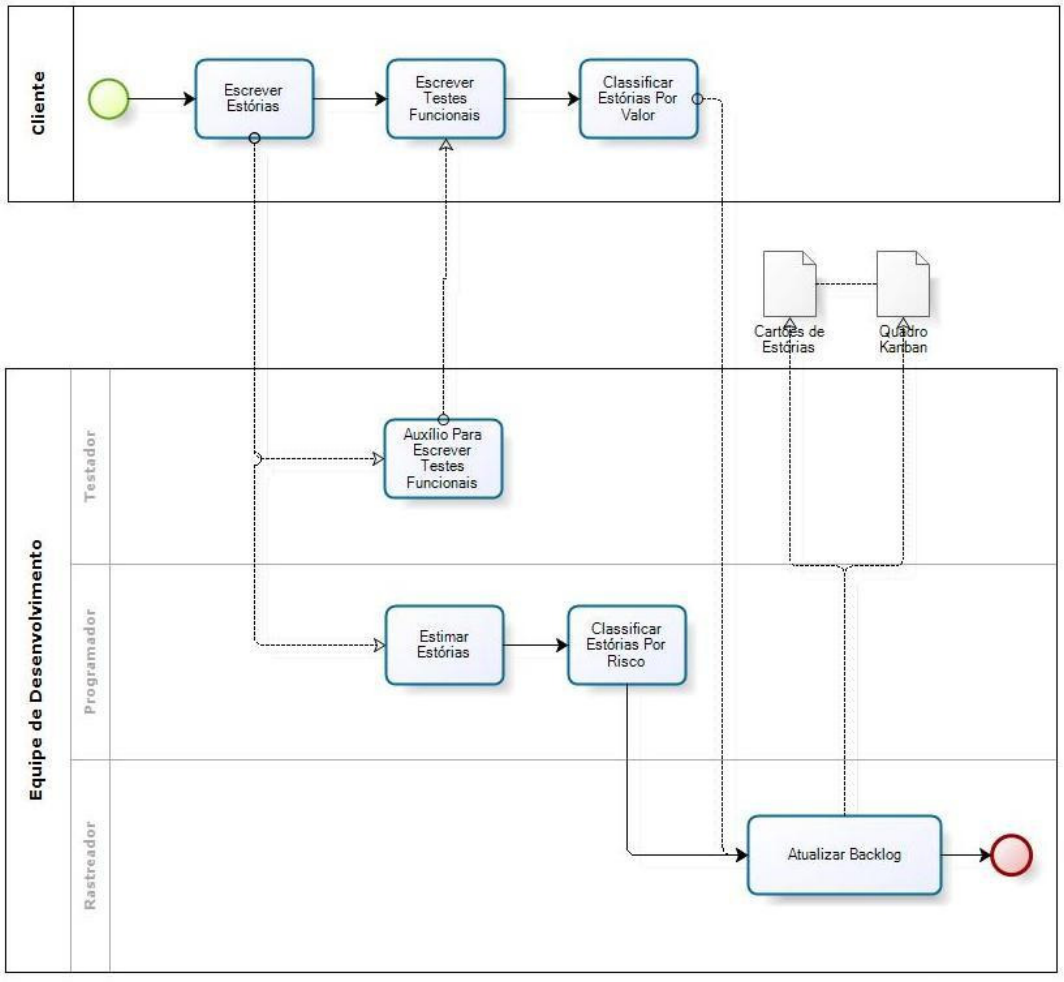

Figura 13 - BPMN do planejamento

A figura 13 mostra em BPMN a fase do planejamento, onde o cliente escreve as estórias, e os testes funcionais e as classifica em função do valor para os negócios. O testador auxilia o cliente para escrever os testes funcionais. Os programadores estimam as estórias e as classificam pelo risco em função da precisão que podem estimar. O rastreador fica responsável por reunir estas informações e atualizar o quadro na coluna “Backlog”.

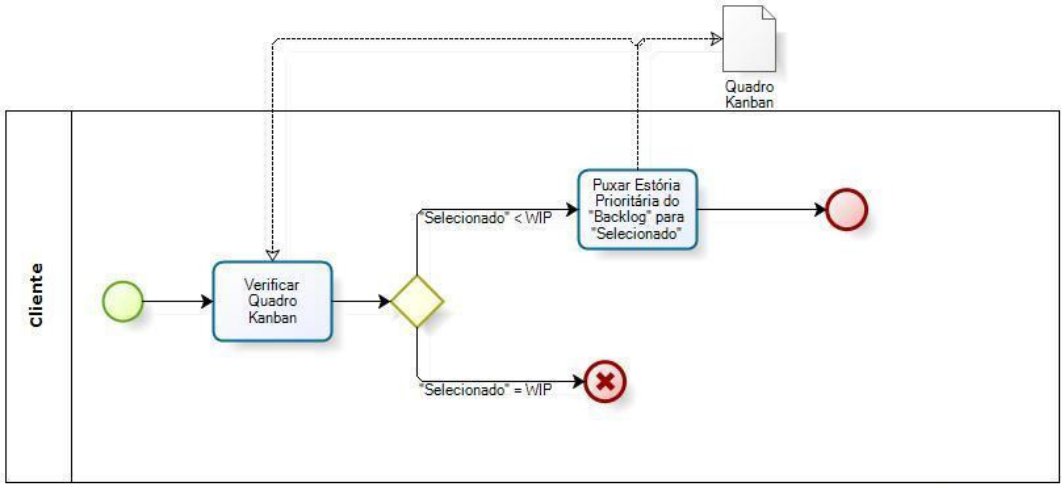

Figura 14 - BPMN da priorização das estórias

A figura 14 mostra o processo de priorização das estórias feita pelo cliente e sua interação com o quadro Kanban, puxando as estórias mais prioritárias para a coluna “Selecionado”.

A figura 15 mostra como as estórias fluem em um sistema puxado do desenvolvimento, respeitando os limites de trabalho em processo (WIP) para que não haja sobrecarga do sistema, fazendo-o trabalhar na sua capacidade real.

Como o sistema é puxado e os limites de trabalho em processo definidos devem ser respeitados, os problemas são expostos imediatamente porque há a interrupção do fluxo, seja por um desequilíbrio na composição da equipe ou problemas que podem ocorrer durante o desenvolvimento. Isso dá o sentido de urgência à equipe para que o problema seja resolvido rapidamente.

A interação com o sistema é feita por todos, clientes e equipe desenvolvimento, promovendo o auto-gerenciamento.

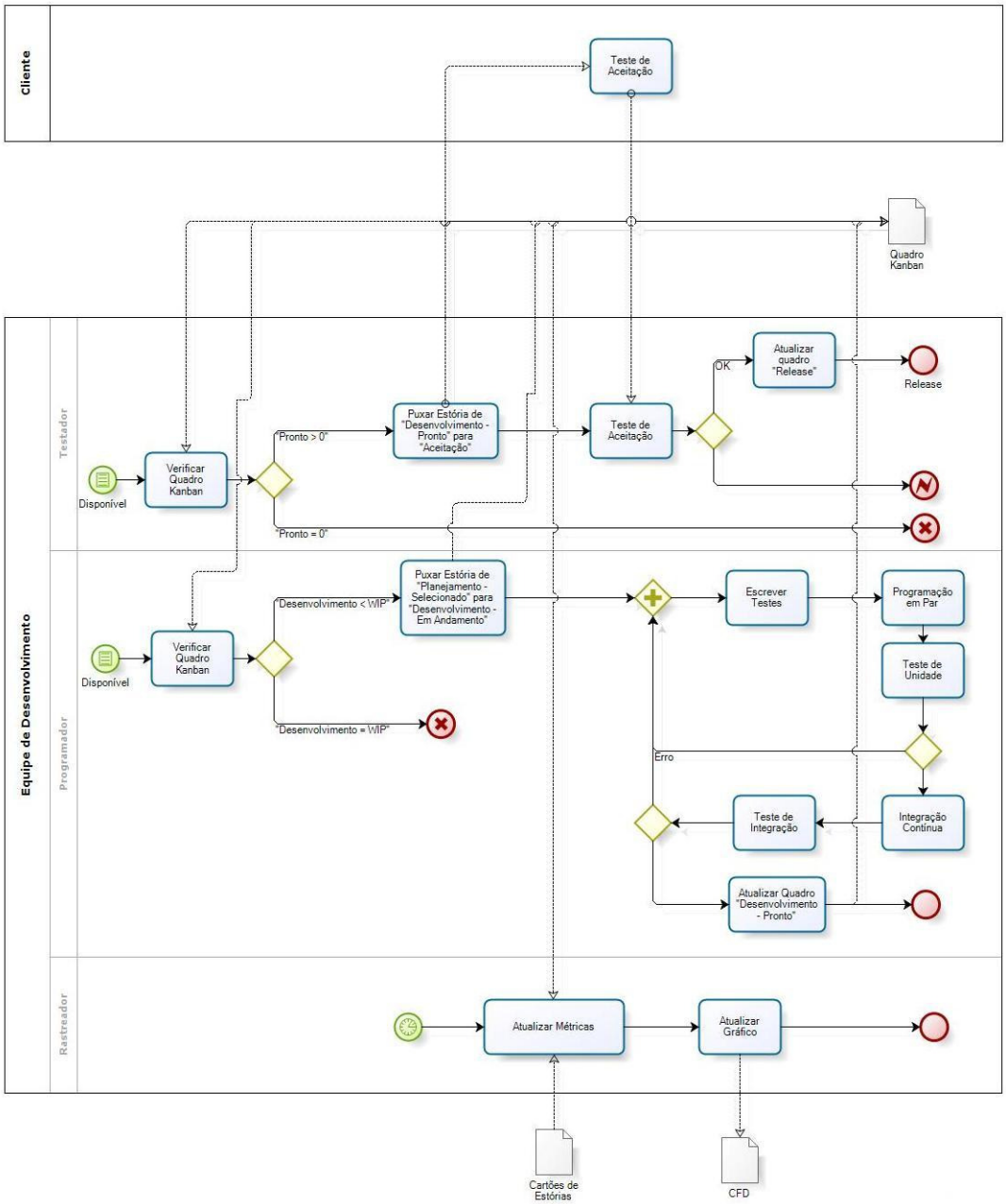

Figura 15 - BPMN do desenvolvimento

4 Conclusões

O Pensamento Enxuto é um tema recente no desenvolvimento de software e tem sido levantado principalmente nos meios das comunidades que utilizam metodologias ágeis.

O estudo para o desenvolvimento deste trabalho mostrou que existem mais de uma abordagem para se aplicar os princípios enxutos ao desenvolvimento de software. Alguns abordam as práticas utilizadas em metodologias ágeis relacionando-os com os princípios enxutos e outros abordam os processos de desenvolvimento de software em termos de fluxo de trabalho aplicando os princípios enxutos.

A abordagem das práticas das metodologias ágeis em termos enxutos geralmente acaba focando apenas nas práticas para a eliminação de desperdício, gerando apenas uma base de compreensão melhor das práticas de desenvolvimento de software que são comuns nas metodologias ágeis.

Já abordagem no fluxo de trabalho em termos enxutos envolve a aplicação mais clara dos princípios enxutos relacionados ao fluxo contínuo e ao sistema puxado, que são essenciais para eliminação de sobrecarga e do desnivelamento das demandas, que são fontes de desperdícios. O Sistema *Kanban* no desenvolvimento de software faz parte desta abordagem.

A implantação de um Sistema *Kanban* no desenvolvimento de software se mostra como um processo que pode levar de forma mais efetiva a implantação de Desenvolvimento Enxuto de Software. Porque com o Sistema *Kanban* expõe todo o fluxo de trabalho do processo de desenvolvimento de software deixando evidente todos os gargalos e problemas do processo para que sejam constantemente tratados em ciclos de melhoria contínua para a efetiva eliminação dos desperdícios. O processo é olhado como um todo e não somente na aplicação de práticas e ferramentas para o desenvolvimento de software.

A utilização do *Extreme Programming* complementa este sistema com suas práticas específicas de desenvolvimento de software, que de uma forma geral tem princípios compatíveis com os princípios enxutos.

REFERÊNCIAS

ANDERSON, D. J. **"The Kanban Primer: A Cultural Evolution in Software"**. Better Software, Jan/Fev 2009. pp 84-90.

ANDERSON, D. J. **Agile Management for Software Enginnering: Applying the Theory of Constraints for Business Results**. Prentice Hall, 2003.

ASTELS, D.; MILLER, G.; NOVAK, M. **Extreme Programming: Guia Prático**. Campus, 2002.

BECK, K. **Programação Extrema Explicada: Acolha as Mudanças**. Bookman, 2004.

HIBBS, C. **The Art of Lean Software Development: A Pratical and Incremental Approach**. O'Reilly Media, 2009.

LADAS, C. **Scrumban – Essays on Kanban Systems for Lean Software Development**. Modus Cooperandi Press, 2009.

LIKER, J. K. **O Modelo Toyota: 14 Princípios de Gestão do Maior Fabricante do Mundo**. Bookman, 2005.

LIKER, J. K.; MEIER, D. **O Modelo Toyota: Manual de Aplicação**. Bookman, 2007.

LIKER, J. K.; HOSEUS, M. **A Cultura Toyota: A Alma do Modelo Toyota**. Bookman, 2009.

OHNO, T. **O Sistema Toyota de Produção: Além da Produção em Larga Escala**. Bookman, 1997.

POPPENDIECK, M.; POPPENDIECK, T. **Leading Lean Software Development: Results Are Not the Point**. Addison-Wesley, 2009.

SOMMERVILLE, I. **Engenharia de Software, 8ª edição**. São Paulo: Pearson Addison-Wesley, 2007.

TELES, V. M. **Extreme Programming**. Novatec, 2004.

WOMACK, J. P.; JONES, D. T. **A Mentalidade Enxuta nas Empresas: Elimine o Desperdício e Crie Riqueza**. Elsevier, 2004.