MAILTON GALDINO DIAS

APLICAÇÃO DA FILOSOFIA LEAN EM UMA EMPRESA DE DESENVOLVIMENTO DE SOFTWARE

LAVRAS - MG

2012

APLICAÇÃO DA FILOSOFIA LEAN EM UMA EMPRESA DE DESENVOLVIMENTO DE SOFTWARE

Monografia apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras como parte das exigências do curso de Sistemas de Informação para a obtenção do título de Bacharel em Sistemas de Informação

Orientador
Dr. Prof. Remulo Maia Alves

LAVRAS - MG
2012

MAILTON GALDINO DIAS

APLICAÇÃO DA FILOSOFIA LEAN EM UMA EMPRESA DE DESENVOLVIMENTO DE SOFTWARE

Monografia de graduação apresentada ao Colegiado do Curso de Sistemas de Informação, para obtenção do título de Bacharel em Sistemas de Informação.

APROVADA em 6 de novembro de 2012.

ANDRÉ LUIZ ZAMBALDE

ANDRÉ VITAL SAÚDE

REMULO MAIA ALVES (Orientador)

LAVRAS-MG

2012

Agradecimentos

Meu agradecimento aos meus pais, Regiane e Veridiano que me deram todo o apoio necessário durante toda minha vida.

As minhas irmãs Ana Maria e Melissa, pelo carinho e força que me passaram.

Agradeço aos professores do Departamento da Ciência da Computação da Universidade Federal de Lavras pelos ensinamentos dos últimos anos, especialmente ao professor Rêmulo Maia Alves pela orientação feita no decorrer do trabalho e ao professor André Luiz Zambalde pelos conselhos dados.

Aos amigos de Lavras, que se tornaram minha nova família e fizeram com que a nova cidade fosse uma memória inesquecível em minha vida.

Aos amigos de Divinópolis que sempre me acolheram quando foi preciso.

A Mitah Technologies que permitiu que o trabalho fosse realizado na organização e na confiança que tiveram em mim.

Resumo

A crescente demanda por sistemas de software e a alta velocidade com que seus requisitos evoluem têm evidenciado que o desenvolvimento deste exige flexibilidade, pois muitas decisões precisam ser tomadas durante o projeto. Além disso, as dificuldades para o desenvolvimento de software vão muito além das questões técnicas. Este trabalho teve como objetivo analisar a possibilidade da implementação da filosofia *lean* em uma empresa de desenvolvimento de software, demonstrando quais processos precisam ser modificados para que a organização fique alinhada a tal ideologia, e também, algumas técnicas que facilitem a adequação a esta filosofia. Para o desenvolvimento deste trabalho, foi observado, através de participação em reuniões de planejamento de Sprint, dentre outras, como a equipe da empresa desenvolve um projeto, após feita a observação chegou-se como resultado propostas de estratégias da filosofia lean que podem ser implementadas na organização. Como conclusão, pode-se verificar que a filosofia *lean* pode ser praticada na organização em pouco tempo, sem grandes gastos financeiros.

Palavras chave: *lean*, software, processos.

Lista de Figuras

Lista de Tabelas

SUMÁRIO

1 INTRODUÇÃO

A crescente demanda por sistemas de software e a alta velocidade com que seus requisitos evoluem têm evidenciado que o desenvolvimento deste exige flexibilidade, pois muitas decisões precisam ser tomadas durante o projeto. Além disso, as dificuldades para o desenvolvimento de software vão muito além das questões técnicas.

Essa necessidade e a questão de como o desenvolvimento de software deve se organizar para entregar um produto com rapidez e qualidade tem se tornado uma das principais discussões na área de engenharia de software (DYBA & DINGSOYR, 2008).

Segundo o Standish Group (2009), 32% dos projetos de software são bem sucedidos, sendo entregues no prazo, cumprindo o orçamento e de acordo com especificações prévias. Por outro lado, 44% destes projetos são entregues atrasados, com orçamento ultrapassado, e/ou com menos funções e funcionalidades requeridas.

Como pode ser verificado por este relatório, nota-se que novas abordagens de desenvolvimento de software devem ser exploradas pelos profissionais desta área para que se tenha um aumento nos projetos de softwares entregues corretamente. Uma das possíveis abordagens é a implantação da filosofia *lean* de desenvolvimento de software.

A filosofia *lean* tem como principal característica identificar o que é valor do ponto de vista de todos os *stakeholders* e desenvolver o produto apenas agregando este valor, tornando o desenvolvimento de software mais ágil e mais preciso com relação ao desejo do cliente. Contudo a filosofia *lean* não prevê nenhuma metodologia ou ferramenta para entrega deste valor. *Lean* é apenas um norte uma filosofia de trabalho, sendo metodologias como scrum e ferramentas como Kaban um dos caminhos a ser seguido para atingir tal norte.

1.1 Objetivos Gerais

O objetivo principal deste trabalho é demonstrar como a filosofia *lean* pode ser implantada em uma empresa de desenvolvimento de software. Para isso foi acompanhado uma parte de um projeto que estava em desenvolvimento, que já apresenta os requisitos principais decididos, tempo de duração, número de funcionários alocados, dentre outros. Durante o desenvolvimento deste trabalho foi analisada como é feita a estimativa para implementação das funcionalidades de um *sprint*, tempo para o desenvolvimento destas funcionalidades e mensuração do número de funcionalidades implementadas por semana.

1.2 Objetivos Específicos

Os objetivos específicos do trabalho são:

- Analisar a possibilidade da implantação da filosofia *lean* na empresa Mitah Technologies.
- Avaliar quais características da empresa não se enquadra com a filosofia *lean*.
- Criar métodos que possam aumentar o desempenho da equipe de desenvolvimento de software, visando entregar mais valor no ponto de vista do cliente.

1.3 Estrutura do trabalho

Este trabalho está estruturado da seguinte forma:

- O Capítulo 1: Introdução: Apresenta uma introdução, a motivação e os objetivos do trabalho.
- O Capítulo 2: Referencial teórico: Apresenta as definições e bases teóricas para o entendimento dos conceitos sobre *lean*, *scrum*, *kanban*, engenharia de software e percepção de valor.

- O Capítulo 3: Metodologia: Mostra a metodologia do trabalho, bem como esta pode ser aplicada.
- O Capítulo 4: Resultados: Apresenta características da empresa, quais práticas devem ser melhoradas e quais novas práticas podem ser implementadas;
- O Capítulo 5: Conclusão: Neste capítulo serão apresentadas as conclusões sobre a monografia e as recomendações para trabalhos futuros.

2 REFERENCIAL TEÓRICO

O presente capítulo irá demonstrar a base teórica que foi utilizada para a realização deste trabalho, nele será abordado sobre engenharia de software e suas características, percepção de valor na visão do cliente, a filosofia lean e seus princípios, metodologia Scrum e seus aspectos e por fim abordará sobre a ferramenta kanban.

2.1 Engenharia de Software

Ao contrário do que muitas pessoas imaginam, o conceito de software abrange não só programas que executam instruções em computadores, mas sim todo conteúdo que o caracteriza, como dados de documentação e configuração (SOMMERVILLE, 2008).

O software é um produto desenvolvido por profissionais da computação da área de Engenharia de Software, ramo da engenharia relacionada ao gerenciamento e desenvolvimento de softwares computacionais a custos baixos e de alta qualidade (PRESSMAN, 2006).

O conceito de Engenharia de Software foi inicialmente proposto em 1968, na tentativa de solucionar a "crise do software". O objetivo era propor soluções mais sistematizadas e controladas para um conjunto de problemas enfrentados no desenvolvimento de sistemas de software, baseados nos conhecimentos de práticas e métodos da computação (PFLEEGER & ATLLE, 2006).

Devido ao crescimento da importância do desenvolvimento de software nos últimos anos, os profissionais e pesquisadores da Engenharia de Software começaram a perceber uma mudança de cenário: o desenvolvimento de software nos dias de hoje, passou a ser um grande esforço coletivo, onde a comunidade de software tem tentado criar esforços para que este desenvolvimento torne-se cada vez mais fácil, rápido, de alta qualidade e menos dispendioso (FUGGETTA, 2000: PRESMAN 2008).

Com isto, metodologias com especificações completas de requisitos, projeto, construção e teste de sistemas, não estão adaptadas ao desenvolvimento rápido, voltado para este cenário com situações de constantes mudanças (SOMMERVILLE, 2008).

Embora existam várias metodologias diferentes, em suma, todas compartilham de características comuns (SOMMERVILLE, 2008):

- **Especificação do Software**: definição da funcionalidade do software e suas restrições;
- **Projeto e implementação**: o produto de software deve atender à especificação;
- **Validação do software**: validação do software para a garantia de que ele faça o que o cliente deseja;
- **Evolução do software**: evolução do software para atender as necessidades do cliente.

2.2 Percepção de Valor

Kötler (2000) define valor como um indicador da contribuição de uma empresa para com o seu cliente baseado na coleção de produtos, serviços intangíveis que a empresa oferece. Para o cliente, um produto representa valor se este responde positivamente aos seus desejos e suas necessidades são atendidas.

Com a crescente complexidade do processo de desenvolvimento de software e as pressões de mercados, as empresas estão buscando a verificação e validação de práticas que asseguram maior valor agregado de um produto de software (ABRAHAMSSON *et al*, 2002; SCHWABER, 2004; BIFFL *et al*., 2006)

Segundo Zeithaml (1988), o valor percebido é a avaliação total do consumidor sobre a utilidade de um produto, baseada em percepções do que é recebido (benefícios) e do que é dado (sacrifícios).

No ponto de vista da Engenharia de Software, a perspectiva de valor fornece uma boa maneira de olhar para o processo de desenvolvimento de um produto, criando estratégia para alcançar um crescimento rentável em longo prazo e uma vantagem competitiva sustentável para as empresas de software (BIFFL *et al*.,2006).

"Qualidade de software é a totalidade das características de uma entidade que lhe confere a capacidade de satisfazer às necessidades explícitas e implícitas" (NBR ISO 8402). Com isso pode-se concluir que valor está diretamente relacionado à qualidade do produto de software.

Entretanto, os desenvolvedores de software precisam considerar quais são os pontos chaves de valor para poder cria-los em seus produtos de software atuais e futuros, assim como entregar este valor para o cliente de forma mais rentável e sustentável possível (BIFFL *et al*., 2006).

O entendimento de valor tanto na visão do cliente quanto na dos desenvolvedores de software faz-se necessário para melhor entendimento sobre a filosofia *lean*.

2.3 Filosofia *lean*

O termo *lean* foi cunhado ao final da década de 80 em um projeto de pesquisa do Massachusetts Institute of Technology (MIT) sobre a indústria automobilística mundial. A pesquisa revelou que a Toyota havia desenvolvido um novo e superior paradigma de gestão nas principais dimensões dos negócios.

De acordo com o Lean Institute Brasil, *lean* é uma estratégia de negócios para aumentar a satisfação dos clientes através da melhor utilização dos recursos. A Gestão *lean* procura fornecer consistentemente valor aos clientes com os custos mais baixos (Propósito) através da identificação de melhoria dos fluxos de valor primários e de suporte (Processos) por meio do envolvimento das pessoas qualificadas, motivadas e com iniciativa

(Pessoas). O foco da implementação deve estar nas reais necessidades dos negócios e não na simples aplicação das ferramentas *lean*.

Para entendermos o conceito *lean* de desenvolvimento de software, primeiro temos que entender o desenvolvimento do produto. Segundo Poppendieck e Poppendieck (2008)

> "Desenvolvimento é o processo de transformar ideias em produtos. Existem duas escolas de pensamento sobre como lidar com esta transformação: poderíamos chamar uma de escola de pensamento determinística e a outra de escola empírica de pensamento. A escola determinística começa criando uma definição completa do produto e então cria uma realização a partir desta definição. A escola empírica começa com um conceito de produto em alto nível e então estabelece ciclos de *feedback* bem definidos que ajustam as atividades para criar uma interpretação ótima do conceito".

O pensamento *lean* trabalha com base na segunda escola, onde o desenvolvimento incremental é necessário para que o produto de software seja desenvolvido de acordo com as necessidades do cliente.

2.3.1 Princípios *lean* de desenvolvimento de software

Princípios são verdades subjacentes que não mudam no tempo ou espaço, enquanto prática é a aplicação dos princípios a uma situação particular. As práticas podem e devem diferir conforme se muda de um ambiente para o próximo, e elas também mudam à medida que uma situação evolui. (POPPENDIECK & POPPENDIECK, 2008).
Poppendieck (2008) define sete princípios *lean* para o desenvolvimento de software.

2.3.1.1 Eliminar o desperdício

Segundo Taiichi Ono (1988) "desperdício é tudo aquilo que não acrescenta valor ao produto na percepção do cliente".

Todas as etapas durante o desenvolvimento de software devem contribuir para gerar um produto final com mais qualidade, com menos tempo e com menos custos.

A Tabela 1 mostra os principais fatores de desperdício na visão de Poppendieck e Poppendieck (2008), estes fatores estão ligados a desperdícios encontrados no desenvolvimento de qualquer produto.

Tabela 1 – Relação desperdício no desenvolvimento de software.

Produção	**Desenvolvimento de Software**
Estoques no Processo	Trabalho inacabado
Superprodução	Funcionalidades extras
Processamento adicional	Reaprendizagem
Transporte	Transferência de controle
Movimentação	Troca de tarefas
Esperas	Atrasos
Defeitos	Defeitos

Fonte: Adaptado de Poppendieck e Poppendieck (2008)

A seguir estes fatores serão explicados.

Trabalho Inacabado:

> "Funcionalidades incompletas são desperdício porque despendem esforços para serem iniciadas e não adicionam valor ao software. Pedaços de código incompletos tendem a se tornar obsoletos, mais difíceis de serem integrados e os programadores lembram menos a respeito inicial do código." (FILHO, 2008).

Funcionalidades extras:

> "Excesso de processos é um desperdício porque eles demandam recursos e aumentam o tempo para conclusão das tarefas. A criação de documentos infla o processo e causa desperdício, pois eles consomem tempo para serem produzidos, sem garantias de que alguém irá lê-los. Documentos ficam desatualizados e podem ser perdidos, tornam a comunicação mais lenta e reduzem o poder comunicativo, pois é um meio de comunicação de via única no qual não é possível que escritor e leitor interajam em tempo real" (FILHO, 2008).

Reaprendizagem:

Reaprendizagem é considerada desperdício, pois o tempo que poderia ser gasto aprendendo coisas novas é utilizado para reaprender algo.

Transferência de controle:

Quando o controle de alguma tarefa é passado para um colega de trabalho, uma vasta quantidade de conhecimento tácito é deixado para trás. Devido ao conhecimento tácito ser difícil de ser repassado, as transferências de controle sempre resultam em perda de conhecimento e desperdício (POPPENDIECK & POPPENDIECK, 2008).

Troca de tarefas:

"Troca de tarefas é uma forma de desperdício porque um número excessivo de mudanças de contexto reduz a produtividade." (FILHO, 2008).

Atrasos:

"Esperar causa atraso e, portanto desperdício. Esperas por requisitos, por teste, por aprovação ou por *feedback* retardam o fluxo do desenvolvimento ou a identificação de problemas" (FILHO, 2008).

Defeitos:

> "Defeitos são desperdício porque o custo para corrigi-los aumenta com o tempo. À medida que o projeto evolui, a complexidade do código aumenta, com isso, a localização e a remoção de um defeito torna-se mais difícil. Algumas vezes corrigir defeitos também é uma forma de desperdício, pois novos defeitos aparecerão e isso pode causar um ciclo vicioso que consome tempo e os recursos do projeto. Portanto, tão importante quanto corrigir defeitos rapidamente é identificar a sua causa para evitar que novos erros sejam introduzidos pela mesma razão, seja uma falha de comunicação ou uma falha no processo" (FILHO,2008).

2.3.1.2 Integrar qualidade

A equipe deve implementar soluções que a deixem segura de que estão construindo um produto de qualidade. Utilizando uma arquitetura adequada, mantendo uma alta cobertura de testes automatizados e preservando a flexibilidade para mudanças (FILHO, 2008).

Shigeo Shingo (1981) descreve dois modos para inspeção da qualidade: inspeção após os defeitos ocorrerem e inspeção para prevenir defeitos. A filosofia *lean* trabalha com o segundo modo, todos os esforços devem ser investidos para que efeitos não ocorram e caso ocorram sua causa deve ser eliminada.

2.3.1.3 Criar conhecimento

Lições devem ser extraídas das experiências vividas pela equipe e incorporadas ao processo, fazendo com que as dificuldades sejam fonte de conhecimento e contribuam para o amadurecimento da equipe e do processo (FILHO, 2008).

Na busca por um padrão de processos, tranca-se os processos em uma documentação que torna difícil que a equipe de desenvolvimento melhore continuamente (POPPENDIECK & POPPENDIECK, 2008).

Uma organização *lean* melhora constantemente seus processos. Ao encontrar uma anormalidade deve-se acionar uma busca pela causa-raiz, acionar experimentos para encontra a melhor maneira de resolver o problema e acionar uma mudança no processo para impedir que ele ressurja (POPPENDIECK & POPPENDIECK, 2008).

Uma forma de auxiliar na melhoria contínua de processos é a utilização do ciclo de Deming, ou ciclo PDCA. O ciclo PDCA consiste de quatro fases. Primeiro, planejamento, que estabelece uma meta ou identifica um problema e elabora um plano de ação. Segundo, execução, que realiza as atividades conforme o plano de ação. Terceiro, verificação, que monitora e avalia periodicamente os resultados, confrontando-os com o planejado. E por fim, ação, que acompanha e padroniza de acordo com o que foi encontrado na verificação, e no fim da ação o ciclo é iniciado novamente. A Tabela 2 mostra o ciclo PDCA adaptado à filosofia *lean*.

Tabela 2. Método de resolução de problemas *lean* e o ciclo PDCA.

Lean	Deming
1. Isole o problema 2. Procure pela causa raiz. 3. Proponha uma contramedida 4. Especifique os resultados esperados.	1. Planejar
5. Execute a contramedida.	2. Execute
6. Verifique os resultados.	3. Verifique
7. Acompanhe e padronize	4. Atue

Fonte: Adaptado de Poppendieck e Poppendieck (2008)

Com estes sete passos é possível criar uma forma metódica para se resolver problemas e assim aumentar a eficiência e a qualidade dos produtos de software desenvolvidos pela organização.

2.3.1.4 Adiar comprometimentos

Ambientes com muita incerteza dificultam previsões. Adiar decisões permite que escolhas sejam apoiadas por mais experiência e conhecimento adquiridos no decorrer do processo (FILHO, 2008).

Isto não quer dizer que todas as decisões devem ser adiadas. Em primeiro lugar, deve-se tentar tornar todas as decisões reversíveis, assim elas podem ser tomadas e facilmente modificadas (POPPENDIECK & POPPENDIECK, 2008).

Para retardar decisões durante a construção de software é importante que a equipe crie a capacidade de absorver mudanças tratando os planejamentos como estratégias para atingir um objetivo e não como comprometimentos (FILHO, 2008).

2.3.1.5 Entregar rápido

Rapidez entre um pedido e uma entrega e entre a entrega e as percepções de quem solicitou permite que o cliente e desenvolvedores aprendam e melhorem através de *feedback* veloz, atualizado e confiável (FILHO, 2008).

Poppendieck e Poppendieck (2008) propõem algumas maneiras de se reduzir o tempo de ciclo de cada projeto.

- Ajuste a chegada de trabalho.
- Minimize o tamanho das coisas no processo.
- Estabeleça uma cadência regular.
- Limite o trabalho pela capacidade.
- Use um cronograma puxado.

2.1.3.6 Respeitar as pessoas

Respeitar as pessoas significa que as equipes recebem planos genéricos e objetivos razoáveis e têm a confiança de se auto organizarem para atingir seus objetivos (POPPENDIECK & POPPENDIECK, 2008).

Não trate as pessoas simplesmente como recursos. Quanto mais a equipe tiver seu trabalho reconhecido, mais motivada e interessada na melhoria de processo ela ficará (FILHO, 2008).

"Uma empresa que respeita pessoas desenvolve bons líderes e garante que a equipe tenha o tipo de liderança que promove pessoas engajadas e pensantes, concentrando seus esforços na criação de um grande produto" (POPPENDIECK & POPPENDIECK, 2008).

2.1.3.7 Otimizar o todo

Uma organização *lean* aperfeiçoa todo o fluxo de valor, do momento em que recebe o pedido visando uma necessidade do cliente até o software seja implantado e a necessidade do cliente seja atendida (POPPENDIECK & POPPENDIECK, 2008).

Lean ainda recomenda a escolha de métricas de alto nível que sejam representativas para identificar a evolução. Essas métricas devem levar em consideração também a qualidade e a satisfação do cliente, pois a partir delas será possível avaliar quais trocas serão vantajosas (FILHO, 2008).

2.4 Fluxo de valor

"O mapa de fluxo de valor é uma linha do tempo contendo os maiores eventos que ocorrem desde o início da contagem do relógio até sua parada." (POPPENDIECK & POPPENDIECK, 2008).

Os mapas de fluxo de valor sempre iniciam e terminam com um cliente. No desenvolvimento de software, o relógio começa a contar em um

mapa de fluxo de valor quando um cliente gera uma demanda. O relógio para quando a solução é satisfatoriamente implantada, ou entregue, resolvendo o problema do cliente (POPPENDIECK & POPPENDIECK, 2008).

A Figura 1 ilustra um mapa de fluxo de valor de solicitação de mudanças de uma funcionalidade de alta prioridade.

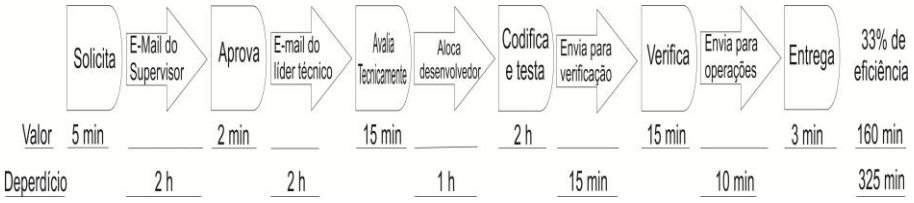


Figura 1. Mapa de fluxo de valor de solicitação de mudança de uma funcionalidade de alta prioridade

Fonte: Adaptado de POPPENDIECK E POPPENDIECK (2008)

Neste mapa pode-se notar que apenas 33% do tempo utilizado para gerar a alteração que o cliente pediu pode ser considerado como valor, uma possível melhora neste mapa seria se o próprio desenvolvedor avaliasse tecnicamente a alteração, com isto acontecendo, 120 minutos de desperdício poderiam ser retirados do processo e a taxa de eficiência passaria a ser de 43%.

A filosofia *lean* não prevê nenhuma metodologia ou ferramenta específica para a implementação dos princípios citados. Logo neste trabalho será utilizado a metodologia Scrum e a ferramenta Kanban como formas de implementação desta filosofia.

2.5 Scrum

Scrum é uma metodologia de desenvolvimento de software que foi criada no início dos anos 1990 por Jeff Sutherland e Ken Schawaber.

O *Scrum* é uma metodologia que aceita que o desenvolvimento de software é imprevisível e formaliza a abstração, sendo aplicável a ambientes voláteis. Ele se destaca dos demais métodos ágeis pela maior ênfase dada ao gerenciamento do projeto. Reúne atividades de monitoramento e *feedback*, em geral, reuniões rápidas e diárias com toda a equipe, visando à identificação e correção de quaisquer deficiências e/ou impedimentos no processo de desenvolvimento (MARÇAL, 2007).

O *Scrum* assume a premissa de que o desenvolvimento de software é muito complexo e imprevisível para ser planejado totalmente inicialmente. Ao invés disso, deve-se usar controle do processo empírico para garantir a visibilidade, inspeção e adaptação (MARÇAL, 2007). Segundo o Scrum, o desenvolvimento de qualquer tipo de produto ou gerenciamento de qualquer tipo de trabalho deve ser um processo iterativo e incremental (CONTROL CHAOS, 2008).

O *Scrum* não define uma técnica específica para o desenvolvimento de software durante a etapa de implementação, a metodologia se concentra em descrever como os membros da equipe devem trabalhar para produzir um sistema flexível, num ambiente de mudanças constantes.

Para que exista este sistema flexível o funcionamento do *Scrum* é estruturado em ciclos chamados *Sprints*. Segundo Henrik Kniger (2012), *sprints* são ciclos de trabalho curtos, que duram geralmente entre duas e quatro semanas, durante este ciclo o time escolhe quais itens do projeto serão desenvolvidos e após o término do Sprint uma reunião é feita chamada *Sprint review* para ver o que foi aprendido e o que deu errado.

O *Scrum* possui um processo bem definido com um ciclo de vida composto por 4 fases (SCHWABER, 1995), como mostrado na Figura 2.

- Planejamento: Estabelecer visão do projeto e expectativas garantindo recursos para sua execução. Nesta fase são criadas as versões iniciais do *Product Backlog* e o plano de release, arquitetura de negócio e técnica em alto nível.
- *Stagging*: Avaliar as várias dimensões do projeto criando itens adicionais ao *Product Backlog* relacionados com o tipo de sistema, time, ambiente de desenvolvimento, tipo de aplicação. Nesta fase os times são formados e são construídos os mecanismos de comunicação e coordenação entre eles.
- Desenvolvimento: Consiste de múltiplos *Sprints* para desenvolvimento dos incrementos de funcionalidade do produto.
- *Releasing*: Realizar a entrega do produto ao cliente.

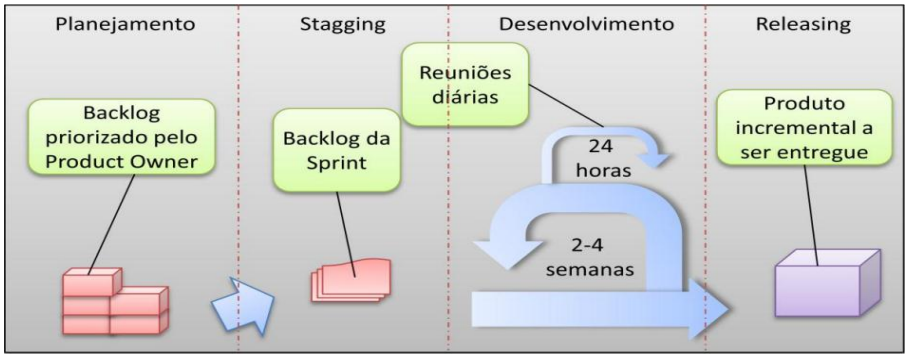


Figura 2. Fluxo Scrum

Fonte: Schwaber (1995)

Santos 2011 elaborou um quadro com os principais papéis delegados aos envolvidos durante o desenvolvimento do projeto. E também descreveu as praticas executadas durante a implementação da metodologia *Scrum*.

Tabela 3. Descrição das prática executadas durante a implementação da metodologia Scrum.

Fonte: Santos (2010)

Práticas	Descrição
Sprint Planning	Reunião em que o *Product Owner* planeja e faz a lista de prioridades que deverão ser cumpridas no projeto por completo.
Daily Scrum	Reunião diária onde cada membro do time responde o que já fez, o que pretende fazer e se há algum impedimento para a conclusão de sua tarefa.
Sprint Review	Reunião de balanço de tudo o que foi feito no *Sprint*.
Sprint Retrospective	Reunião de retrospectiva da *Sprint* onde são avaliados aspectos como o trabalho em equipe, os pontos positivos, negativos e como desenvolver estratégias de crescimento.
Product backlog	Lista de prioridades feita logo no início do projeto, com o objetivo de listar o que deve ser entregue ao cliente.
Sprint Backlog	Produto oriundo do *Sprint Planning Meeting*. É uma lista de tarefas específicas a serem desenvolvidas durante o *Sprint*.
Burndown Chart	Gráfico que estima o tempo gasto no andamento do trabalho dentro do *Sprint*. Ele é monitorado pelo time.
Potentially Shippable	Incremento de produtos potencialmente entregável.

Tabela 4. Descrição dos papéis de cada funcionário em um projeto implementado com *Scrum*.

Fonte: Santos (2010)

Papéis	Descrição
Product Owner	É o moderador entre os interesses do time de desenvolvimento e do cliente. Sua responsabilidade é manter a equipe funcional e produtiva.
Scrum Master	Representante do cliente no projeto. Suas responsabilidades são definir funcionalidades de acordo com o valor de mercado, planejar e fazer a lista de prioridades.
Time	É o time responsável pelo desenvolvimento do projeto. Ele é multidisciplinar e é composto por um grupo de cinco a nove integrantes.
Cliente	Participa das tarefas relacionadas à implementação da lista de funcionalidades.

2.6 Kanban

Kanban é uma palavra japonesa que significa, cartão visual. Na Toyota, Kanban é o termo usado para sistemas visuais e fisícos que junto mostram inteiramente o sistema de produção *Lean*. Diferente dos americanos, os japoneses optaram por implementar um sistema de produção diferente, onde a demanda sinaliza quando deve-se produzir mais (*pull system*).

O uso do método Kanban para o gerenciamento de equipes de desenvolvimento de software registrou um aumento a partir de 2007, quando Rick Garber e David J. Anderson publicaram nas conferências "*Lean New Product Development*" e "*Agile* 2007", os resultados obtidos no uso deste método no desenvolvimento de software.

As principais vantagens do Kanban são:

- **Visualizar o fluxo de trabalho** - Quebrando o trabalho em pedaços, escrevendo cada item em um cartão, e colocando o cartão no Kanban. Nomear cada coluna, para ilustrar o que cada item significa no fluxo de trabalho.

- **Limitar o trabalho em progresso** - Limitar quantos items pode ter cada coluna é uma forma de limitar a quantidade de trabalho. Esse artífcio é um dos pontos chaves do método, uma vez que ele é responsável por definir o Kanban como um *pull system*.

- **Mensurar e gerenciar ciclos de produção –** Como o kanban se trata de um processo mais adaptativo do que prescritivo, acaba se tornando muito empírico. As fases do processo em questão e os valores de itens de trabalho para cada fase devem ser testados pela equipe de forma a encontrarem o valor ideal do fluxo de trabalho. Não há uma fórmula para chegar a esse valor, a equipe deve experimentar e encontrar os números que melhor se encaixem na sua realidade.

A Figura 3 mostra como é um Kanban

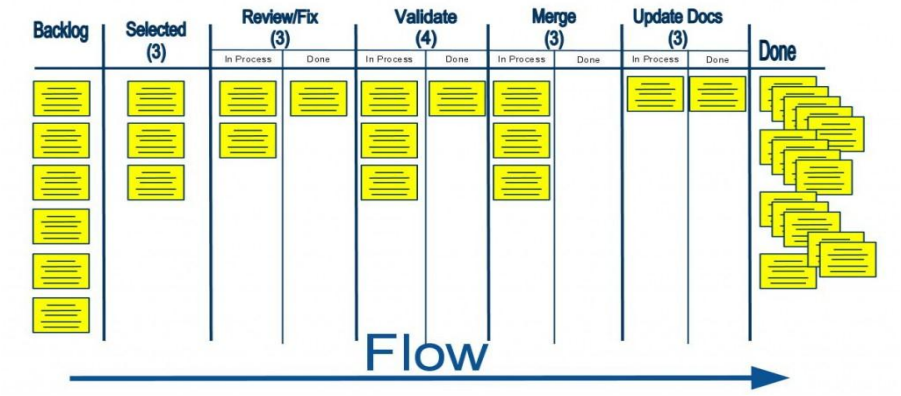


Figura 3. Exemplo Kanban.

Fonte: Adaptado de http://blogs.mulesoft.org/implementing-kanban-for-sustaining-engineering/

2.5 Trabalhos relacionados

Foi feita uma revisão literária para encontrar artigos relacionados ao trabalho. A procura pelos artigos foi feita em várias bases de dados por meio do uso de um portal brasileiro de buscas de periódicos, o Portal Brasileiro de Periódicos da CAPES – Agência Brasileira de Apoio e Fomento a Pesquisa.

A pesquisa foi feita nas seguintes bases de dados eletrônicas:

- Biblioteca Digital da ACM
- Academic Search Premier – ASP (EBSCO)
- Google acadêmicos
- Oxfor Journals (Oxford University Press)
- SciELO.ORG
- SpringerLink

As palavras chaves utilizadas na procura do material foram:

- *Lean* Software *development*.
- *Lean* Software.
- Toyota *Product System*.
- *Lean* software *Project*.
- Software *Engineering*.
- *Lean*.
- *Lean practices*.
- *Lean Principles*.
- Desenvolvimento *Lean* Software.

A partir desta busca, foram encontrados artigos e dissertaçãoes relacionada ao desenvolvimento *lean* de software. Apenas três destes artigos estão diretamente relacionados ao tema abordado no trabalho. Abaixo há uma pequena descrição destes.

Middleton (2001), desenvolveu dois estudos de casos na implementação *lean* na engenharia de software e o método de pesquisa escolhido foi o quantitativo. No estudo a companhia alocou recursos e desenvolvedores em duas diferentes equipes, a primeira equipe contava com desenvolvedores experientes (Equipe A), já a segunda (Equipe B) contava com desenvolvedores com menos experiência.

As respostas dos participantes, inicialmente, foram frustrantes, pois como os erros encontrados eram imediatamente corrigidos, isso tornava o trabalho um pouco diferente do que as equipes estavam acostumadas. Contudo ao longo prazo as taxas de erros caíram drasticamente, com a equipe B se adaptando melhor a nova forma de trabalho. Porém a filosofia *lean* não pode ter sido mantida na empresa em função da forte hierarquia funcional e ao modelo de promoção da empresa.

Karlsson & Ahlströhm (2009) identificaram os principais fatores que auxiliam e dificultam a introdução da filosofia *lean* em uma empresa.

Fatores que dificultam. Primeiro, não é fácil criar um foco de trabalho, quando os empregados não se sentem confortáveis com a função. Segundo, engenharia simultânea é um desafio quando proveniente de um trabalho que tem etapas sequenciais. Terceiro, é difícil coordenar projetos quando as pessoas têm problemas em entender o trabalho de outros da equipe ou do cliente. Quarto, como estimativas de custos são esperadas, a relação com o cliente se torna difícil, pois eles esperam que você tenha todo o custo do projeto na primeira reunião.

Fatores que auxiliam: Primeiro, cooperação entre cliente e empresa durante o desenvolvimento do produto, para que se tenha um maior *feedback* de ambas partes. Segundo, para a implantação *lean* ser bem sucedida é necessário que pessoas com grande experiência nas funções. Terceiro, o apoio da alta administração da empresa é um fator indispensável. Quarto, colaboração entre os departamentos da empresa é crucial.

Parnell-Klabo (2006) em seu trabalho mostra que os maiores obstáculos para organização se tornar ágil são a obtenção de escritório aberto

para alocar toda a equipe, ter apoio da alta administração e treinar as pessoas para que elas se tornem mais suscetíveis a mudança. Depois de ultrapassado esses obstáculos com a ajuda de oficinas de treinamento e utilização de projetos pilotos, os resultados obtidos pela organização mostraram-se satisfatórios, o prazo para entregas de produtos foram reduzidos em 40-50%.

3. METODOLOGIA

Nesta seção será descrita a metodologia de pesquisa que foi utilizada para realização deste trabalho.

Segundo Jung (2009), a metodologia de pesquisa se caracteriza por um conjunto de métodos, técnicas e procedimentos que objetivam tornar viável a execução da pesquisa, que por sua vez resulta em um novo produto, processo ou conhecimento.

Nos próximos tópicos serão descritos o tipo de pesquisa adequado e os procedimentos metodológicos realizados para efetivação deste trabalho.

3.1 Tipos de Pesquisa

O método científico de uma pesquisa corresponde a um caminho para se chegar ao fim de um determinado trabalho (GIL, 1999).

Existem várias maneiras de se classificar uma pesquisa, sendo os pontos mais tradicionais: quanto à natureza da pesquisa, a forma de abordagem do problema, os seus objetivos e os procedimentos técnicos (SILVA E MENEZES, 2000).

Em relação à natureza, este trabalho pode ser classificado como aplicada ou tecnológica, uma vez que o objetivo é gerar conhecimentos para a aplicação prática, dirigidos à solução de um problema específico (JUNG, 2004).

Quanto aos objetivos da pesquisa, este trabalho pode ser considerado como exploratória uma vez que visa à descoberta de teorias e práticas que modificarão a forma de desenvolver software na organização.

Com relação à abordagem do problema, a classificação da pesquisa é considerada como qualitativa. Para Penna (2004) “a pesquisa qualitativa caracteriza-se como uma abordagem interpretativa e compreensiva dos fenômenos, buscando seus significados e finalidades”.

O procedimento para realização do trabalho em questão foi caracterizado como estudo de caso único, uma vez que toda a pesquisa foi realizada apenas em uma empresa.

Em relação ao método para coleta de dados, este trabalho foi do tipo observação, já que o pesquisador participou de reuniões de *Sprint* com a equipe de desenvolvimento sem opinar sobre possíveis melhorias.

A Figura 4 ilustra a classificação que condiz com este trabalho.

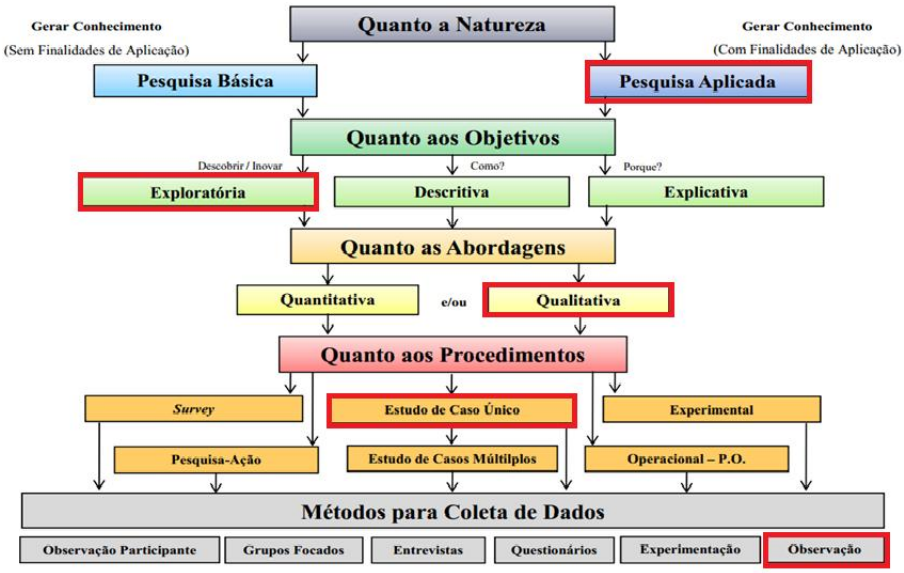


Figura4: Classificação dos tipos de pesquisa.
Fonte: Adaptado de Jung (2009)

3.2 Estratégia da Pesquisa

A primeira fase foi escolher a empresa e o projeto que se enquadrou melhor com tempo máximo para execução deste trabalho, três meses. Após a escolha da empresa e projeto, foi analisado como está o atual cenário de desenvolvimento de software dentro da empresa. A próxima fase foi analisar

quais os principais pontos do roteiro de implantação da filosofia *lean* para desenvolvimento de software, proposto por POPPENDIECK E POPPENDIECK (2008), já estão na cultura da empresa e a partir disso propor pontos do roteiro que não fazem parte da cultura da empresa, priorizando os pontos mais importantes. Abaixo segue o roteiro.

1. **Implementar os princípios *lean* ao longo do fluxo de valor inteiro e do produto completo:** Será nomeado um líder ou uma equipe de liderança do fluxo de valor que aceite a responsabilidade pelo fluxo de valor inteiro, começando e terminando com o cliente. A esquipe ou líder terá que se concentrar no produto todo, não apenas no software. Em seguida será desenhado o mapa de fluxo de valor e então haverá a procura por interrupções de fluxo, retornos e por áreas que não estejam disponíveis quando necessárias, ou que não são capazes de entregar os resultados esperados. Então serão corrigidos os processos falhos e será abastecida as capacidades ausentes no sistema.
2. **Reestruturação das medidas.** É provável que a empresa tenha suas próprias medidas de tempo locais, ou seja, medida de tempo de desenvolvimento, tempo de teste, dentre outros, nesta etapa será proposto que a empresa adote medidas globais, as medidas de fluxo de valor.
3. **Reduzir o custo de passagem de limites:** Caso tenha grandes atrasos no fluxo de valor, eles provavelmente ocorrem nas fronteiras entres os departamentos. Será feita uma analise destes atrasos e propostos melhorias para que se possa melhorar o fluxo de valor entre estes departamentos.
4. **Transferência da responsabilidade e tomada de decisão para os níveis mais baixos da organização:** Uma empresa *lean* deve ser implementada pelas equipes de trabalho que realmente criam valor com o apoio da alta gerência. Nesta etapa de trabalho será proposto

que a(s) equipe(s) projetem seus próprios processos *lean* com a orientação do líder ou da equipe de fluxo de valor.

5. **Trabalhar em lotes pequenos:** Nesta etapa será reduzido o tamanho dos *releases,* estabilizá-los e tentar repeti-los. A partir desta etapa será sempre mantido o tamanho de cada *release*.
6. **Limitar o trabalho conforme a capacidade:** Ter equipes trabalhando com uma cadência repetível de trabalho para poder estabelecer a capacidade da mesma. Nesta etapa será proposto que cada membro da equipe puxe seu trabalho da lista de funcionalidades, após um tempo será possível mensurar a capacidade da equipe, feito isso, será possível determinar quanto a equipe pode produzir em determinado tempo, podendo limitar o quanto de trabalho que a organização pode aceitar.
7. **Acabar com a noção de que é uma boa prática começar o desenvolvimento com uma especificação completa:** Nesta etapa será proposto que a empresa utilize o desenvolvimento concorrente de software. Este modelo permite que funcionalidades e especificações surjam durante o desenvolvimento do projeto, ele tente a economizar tempo, dinheiro e produz melhores resultados, pois permite que a organização só tome decisões com bases nos dados mais atuais.
8. **Manter opções:** Desenvolver múltiplas opções para todas as decisões e não fechar qualquer opção até o cliente dar o alvará que tudo pedido está pronto. Desenvolver códigos suscetíveis a mudança, sempre o mantendo limpo e simples.
9. **Criar equipes construtoras de *design.*** Criar métodos para desenvolver uma arquitetura de sistemas que permita que os produtos sejam quebrados em módulos lógicos que podem ser endereçados a equipes multifuncionais representando os interesses de todos os passos do fluxo de valor. Fornecer a cada equipe a liderança e os incentivos adequados para manter o engajamento, a

transparência e o *feedback* intensivo. Essas equipes serão encorajadas a compartilhar mais cedo e mais frequentemente, a falhar rápido e a aprender constantemente.

10. **Desenvolver a cultura de melhoria constante:** Criar o tempo para que cada equipe examine e melhore seus processos e teste suas suposições. Criar eventos multiequipes e multifuncionais para que todos os membros do projeto saibam como está o andamento do projeto no todo, para identificar acomodações e restrições no fluxo de valor e substituir estas restrições por práticas que melhorarão resultados globais.
11. **Criar métodos de resolução de problemas:** Ensinar o ciclo PDCA adotado ao *lean* para as equipes. Guiar as equipes para estabelecerem hipóteses sobre os problemas, conduzir experimentos rápidos sobre estes problemas e documentar os principais fatores destes experimentos.
12. **Sincronizar:** Reduzir o trabalho inacabado. Será proposto que as equipes escrevam os testes antes de começar a codificar. Os desenvolvedores não deverão colocar os erros em uma lista de erros para corrigi-los depois, eles devem corrigir o erro assim que o encontrarem. Integrar o código de forma contínua e extensiva sempre que possível.
13. **Automatizar:** Automatizar cada processo que seja rotina como teste unitários, *builds*, instalação, para evitar que membros da equipe cometam erros.
14. **Refatorar:** Manter o código base limpo e simples, e no instante que as duplicações aparecerem, refatorar o código, os testes e a documentação para reduzir a complexidade.
15. **Escrever menos código:** Criar a cultura de escrever as funcionalidades de um sistema para apenas aquelas que são realmente necessárias para agregar valor ao cliente.

A pesquisa foi realizada no período de março a abril de 2012, em uma empresa de base tecnológica, situada na Universidade Federal de Lavras – Lavras – MG.

4. RESULTADOS E DISCUSSÃO

Neste capítulo, será apresentado como é o atual cenário de desenvolvimento de software na organização, as propostas feitas para enquadrar o atual cenário a filosofia *lean*, além de discussões sobre estas propostas.

4.1 Descrição da organização e do software

A empresa foco deste trabalho foi a Mitah Technologies, incubadora de base tecnológica da Universidade Federal de Lavras, a companhia foi fundada com o foco no mercado de softwares de rastreabilidade, melhoria de processos e sistemas de gerenciamento integrado.

A empresa conta com vinte colaboradores, incluso neste número estão os desenvolvedores, pessoal do setor administrativo e gestores. O software que foi utilizado como referência para este trabalho denomina-se Smartbiz, sistema de gerenciamento integrado voltado especificamente para o setor de laticínios.

Estão responsáveis para o desenvolvimento deste software oito colaboradores, entre eles encontram-se desenvolvedores, testadores e designers.

4.2 Descrição dos processos da empresa

A metodologia de desenvolvimento de software utilizada pela Mitah Technologies é o Scrum. Como mostrado na Sessão 2.5 o Scrum é uma metodologia de desenvolvimento de software que possibilita que os projetos sejam feitos de forma incremental permitindo assim que a organização se adeque constantemente as necessidades do cliente. O processo de desenvolvimento de software da Mitah pode ser dividido em quatro etapas.

Primeira etapa: Gerente de projetos ou *product owner* conversa com o cliente sobre quais requisitos são necessários para concepção de um

sistema que atenda suas necessidades e a partir disso cria o *product backlog* que será utilizado como referência para o desenvolvimento do projeto. Nesta etapa pode ser aplicado o seguinte ponto do roteiro proposto por Poppendieck (2008), acabar com a noção de que é uma boa prática começar o desenvolvimento com uma especificação completa.

Segunda etapa: Fase de planejamento, onde o gerente de projetos avalia as dimensões do projeto e cria itens adicionais ao *product backlog* relacionados com o tipo de sistema, time. Nesta etapa, a filosofia lean pode ser implementada aplicando-se o princípio, integrar qualidade, garantindo que as dimensões do projeto e os itens adicionais tenham algum valor para o cliente.

Terceira etapa: Desenvolvimento, consiste de múltiplos *sprints*, esta fase é detalhada abaixo.

1 – *Product owner* convoca uma reunião de planejamento com o time para demonstrar quais funcionalidades serão implementadas.

2 – Time mensura quanto tempo levará para implementar cada funcionalidade através da estratégia de *planning poker*. O *planning poker* é uma forma de todos do time que estão ligados ao desenvolvimento do projeto dar sua opinião durante o *Sprint planning*. Abaixo segue um pequeno exemplo de como funciona este método.

Cada pessoa do time possui vários cartões com uma quantidade de horas que eles estimam que irá demorar para desenvolver cada funcionalidade. No caso da Mitah este cartões possuem os valores, 1, 3, 5, 8, 13, 21, todos estes números são relacionado a horas para desenvolver uma funcionalidade, além destes cartões há um com símbolo de infinito que demonstra que alguém não entendeu a funcionalidade e um com o símbolo de uma xícara de café que quer dizer que os presentes se cansaram da reunião e está na hora de fazer uma pausa.

No primeiro momento o *Scrum* Master faz uma explicação do que se trata a funcionalidade que deve ser desenvolvida, neste momento todas as pessoas podem fazer alguma pergunta sobre esta funcionalidade. Após

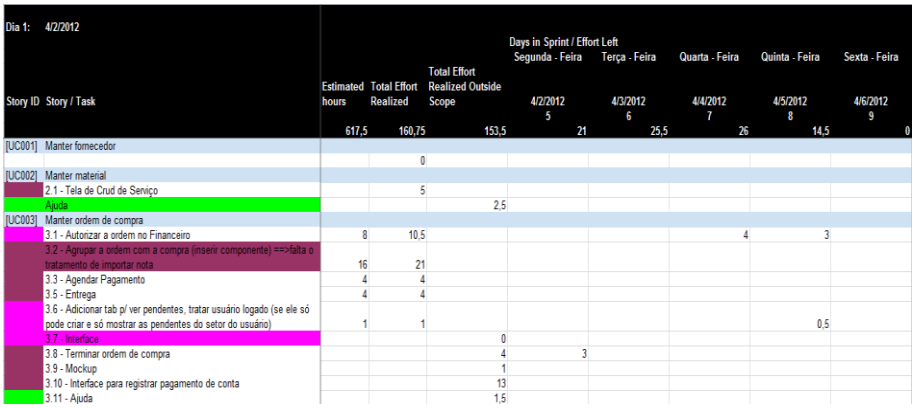
entendida a funcionalidade, todos devem mostrar, ao mesmo tempo, um cartão com um número que a pessoa acredita que é o tempo necessário para implementar a funcionalidade, caso haja um número próximo a totalidade de votos, este valor é assumido como tempo para o desenvolvimento desta funcionalidade, mas caso haja uma discrepância muito grande entre os números mostrados, significa que a funcionalidade não foi bem entendida. Com isto uma nova explicação do requerimento se torna necessário.

O *plannig poker* é um método útil, pois faz com que todos os envolvidos no desenvolvimento tenham uma ideia geral de todas as funcionalidades do projeto e também faz com que todos participem das estimativas. Nesta etapa o princípios da filosofia lean que pode ser implementados é, entregar rápido. Este princípio pode ser implementado criando uma forma de estabelecer uma cadência regular de trabalho e usando um cronograma puxado, exemplo, um kanbam.

3 – Início do Sprint, e durante o Sprint há reuniões diárias, *daily scrum* para todo time entender como está o desenvolvimento completo do Sprint.

Quarta etapa: Entrega do Sprint, onde o cliente avalia se foi entregue o que ele realmente queria, ou seja, aquilo que ele acredita que tenha valor.

Para controle do andamento da implementação das funcionalidades, bem como para definição de responsabilidades, a empresa utiliza de um kanban virtual. A Figura 5 mostra como é tal kanban.



Dia 1: 4/2/2012					Days in Sprint / Effort Left					
					Segunda - Feira	Terça - Feira	Quarta - Feira	Quinta - Feira	Sexta - Feira	
Story ID	Story / Task	Estimated hours	Total Effort Realized	Total Effort Realized Outside Scope	4/2/2012 5	4/3/2012 6	4/4/2012 7	4/5/2012 8	4/6/2012 9	
		617,5	160,75	153,5	21	25,5	26	14,5	0	
[UC001]	Manter fornecedor									
			0							
[UC002]	Manter material									
	2.1 - Tela de Crud de Seniço		5							
	Ajuda			2,5						
[UC003]	Manter ordem de compra									
	3.1 - Autorizar a ordem no Financeiro	8	10,5				4	3		
	3.2 - Agrupar a ordem com a compra (inserir componente) ==>falta o tratamento de importar nota	16	21							
	3.3 - Agendar Pagamento	4	4							
	3.5 - Entrega	4	4							
	3.6 - Adicionar tab p/ ver pendentes, tratar usuário logado (se ele só pode criar e só mostrar as pendentes do setor do usuário)	1	1					0,5		
	3.7 - Interface			0						
	3.8 - Terminar ordem de compra			4	3					
	3.9 - Mockup			1						
	3.10 - Interface para registrar pagamento de conta			13						
	3.11 - Ajuda			1,5						

Figura 5. Demonstração Kanban da empresa

Fonte: Autor

Os tons mais escuros correspondem a funcionalidades em desenvolvimento, as funcionalidades que estão com fundo branco já foram implementadas e testadas.

4.3 Proposta da integração da filosofia lean

Como já apresentado, a proposta deste trabalho é demonstrar quais melhores práticas da filosofia *lean* a organização pode incorporar no desenvolvimento dos seus projetos, a fim de agilizar a execução das tarefas e criar uma cultura de melhoria continua na organização, deste modo entregando produtos que agregam maior valor na visão do cliente. Todas as propostas feitas da implementação da filosofia lean forão para a terceira etapa do ciclo Scrum, desenvolvimento, pois nesta etapa o time de desenvolvimento está mais ativo no projeto, e é este time que entrega valor para o cliente.

Como já citado, a organização utiliza o *Scrum* como metodologia para desenvolvimento de projetos, em função disso vários aspectos da filosofia *lean* que são propostos por Poppendieck e Poppendieck (2008), já fazem parte da organização. Tais aspectos são descritos abaixo.

1. **Transferência da responsabilidade e tomada de decisão para os níveis mais baixos da organização:** Como todos os membros da equipe ajudam a estimar o tempo de desenvolvimento dos *Sprints*, então todos tem responsabilidade com relação aos prazos de entregas. Deste modo a responsabilidade e tomada de decisões está também nos níveis mais baixos.
2. **Trabalhar em lotes pequenos:** Como os projetos são desenvolvidos em *Sprints* de um mês, isto implica que a organização já trabalha com *realises* de um tamanho suficientemente pequeno.
3. **Limitar o trabalho conforme a capacidade:** Durante o *Sprint Planning* cada pessoa da equipe já faz uma estimativa do que irá conseguir produzir durante o *Sprint*, desta forma limitando o trabalho conforme a capacidade de cada um.
4. **Acabar com a noção de que é uma boa prática começar o desenvolvimento com uma especificação completa:** Como a metodologia *Scrum* é composta de vários ciclos, isto permite que as especificações surjam durante o projeto, não sendo necessário desta forma predizer todas as funcionalidades no início deste.
5. **Reduzir o custo de passagem de limites:** Como a Mitah ainda é uma empresa pequena, apenas com 20 colaboradores, problemas de comunicação praticamente não ocorrem. Com isto pode-se afirmar que a organização ainda não tem custos com passagem de limites.

4.3.1 Proposta de características a serem implementadas

Como já mostrado, a Mitah conta com algumas características que são inerentes ao desenvolvimento *lean* de software, desta forma abaixo serão propostas apenas as características que são interessantes de serem incorporadas na organização no primeiro momento.

4.3.1.1 Utilização do Kanban Físico

A primeira proposta é a de utilizar um kanban físico ao invés de um kanban eletrônico como a organização utiliza atualmente. Dois aspectos foram considerados para se fazer essa proposta. Primeiro, com um kanban físico a evolução do mesmo se torna fácil, pois é possível retirar e adicionar colunas e modificar o *layout* dentre outras coisas, abaixo segue uma lista de possíveis modificações.

- Adicionar ou remover colunas.
- Adicionar novos tipos de item no kanban.
- Escrever algumas políticas, tais como definição de *done*.
- Escrever algumas métricas ao longo do quadro, como velocidade.
- Adicionar cores para itens, por exemplo, função em um post-it vermelho significa defeito.

Tais modificações podem ser úteis dependendo do andamento do projeto, e com um kanban físico estas mudanças podem ser rapidamente feitas e todos na equipe estão habilitados a fazê-la.

O segundo fator considerado para se utilizar o kanban físico é a colaboração. Como há uma reunião diária para analisar como está o andamento do projeto, essa se tornar mais produtiva com a utilização desta ferramenta, pois, todos estão vendo como anda o desenvolvimento do trabalho sem ter que perguntar especificamente, e caso haja algum problema para o desenvolvimento de alguma tarefa, basta marcar esta tarefa no kanban e todos ficarão cientes do problema.

4.3.1.2 Utilização de novas métricas

Métricas são uteis para descobrir se o atual processo de desenvolvimento precisa ser melhorado e se mudanças que são feitas nesse processo resultam em algo melhor. Então, é proposto que seja utilizado dois tipos de métricas.

A primeira, funcionalidades por semana. Para que isto seja implementado basta ao final de cada semana contar quantas funcionalidades foram implementados e testados. Esta informação pode ser escrita em alguma parte do kanban para que todos possam acompanhar quantas funcionalidades foram implementadas nas últimas semanas e deste modo deixando todos do time ciente se há aumento ou não de produtividade.

A Figura – 6 – demonstra como isto ficaria no kanban.

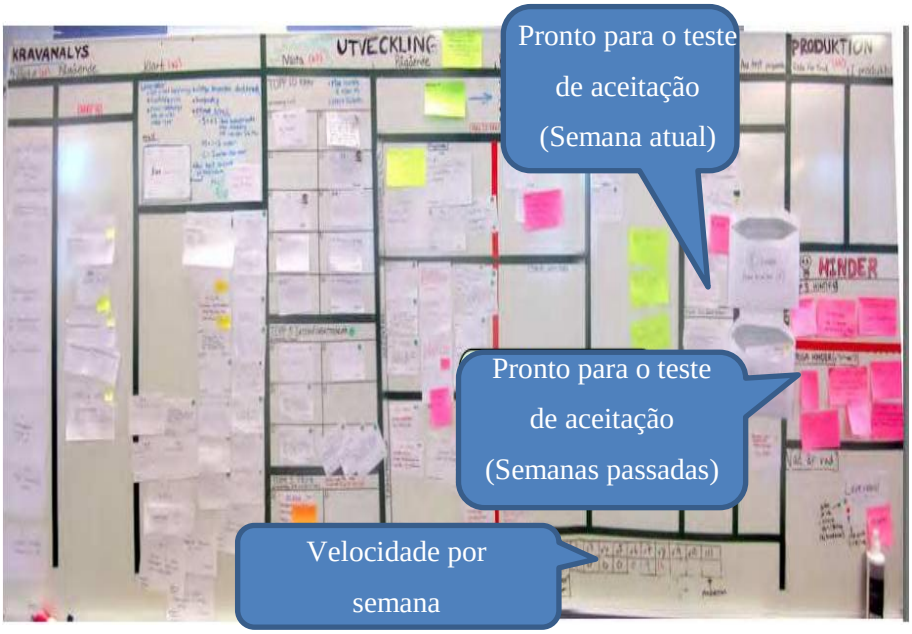


Figura 6. Demonstração Kanban com velocidade por semana

Fonte: Adaptado de Kniberg, H. (2012)

Usando a informação de quantas funcionalidades são feitos por semana, é possível gerar um gráfico burn-up. A Figura 7 mostra o total de funcionalidades que foram realizados em um projeto.

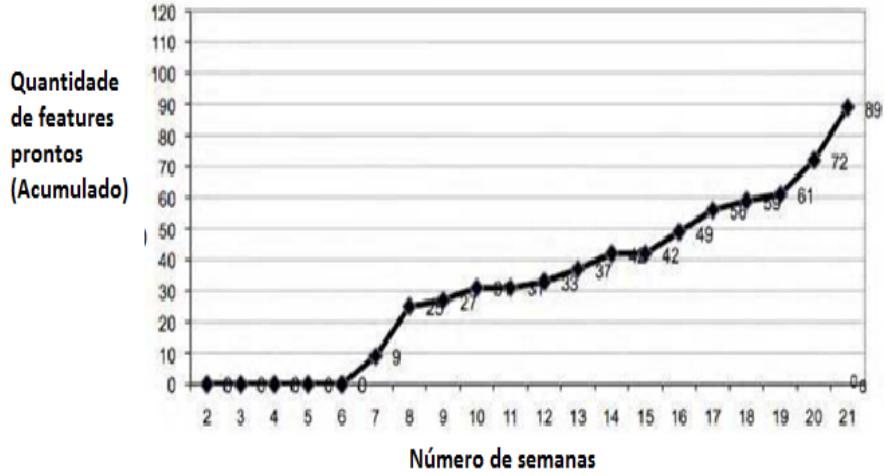


Figura 7. burn-up.

Fonte: Adaptado de Kniberg, H. (2012

A informação provida do burn-up é útil de várias maneiras. Primeiro, ela pode ser usada para confirmar se a estimativa inicial de funcionalidades por semana está correta. Segundo é possível fazer um melhor levantamento da quantidade de funcionalidades que uma equipe pode desenvolver por semana.

O *burn-up* também é útil para demonstrar problemas maiores, por exemplo, nas primeiras semanas a velocidade demostrada na Figura 7 foi igual à zero. Mesmo que time tenha se esforçado para realizar o trabalho possivelmente algum gargalo estava impedindo que a equipe avançasse com a finalização dos requisitos. Com isto um senso de urgência é criado em todo o time, fazendo com que todos tentem resolver este gargalo.

E por último o *burn-up* é útil para visualizar a melhoria no processo. A Figura 5 demonstra isso claramente quando se nota a inclinação da curva em vários momentos. Este tipo de visualização dos resultados ajuda a motivar todos os envolvidos no projeto.

Outra métrica proposta é o tempo de ciclo, que significa quanto tempo uma funcionalidade demora para ficar pronta. Isto é fácil de se mensurar, basta anotar quando a funcionalidade entrou para a lista de funcionalidades a

serem implementadas, data de ínicio, e anotar a data em que esta funcionalidade ficou pronta, data de fim. Com estes dados é possível construir um quadro que mostra quantos dias uma funcionalidade demora para ficar pronta. A partir deste quadro é possível comparar com a estimativa inicial de quanto tempo se demoraria para desenvolver uma funcionalidade e verificar se a estimativa estava correta ou não.

A Figura 8 mostra um exemplo.

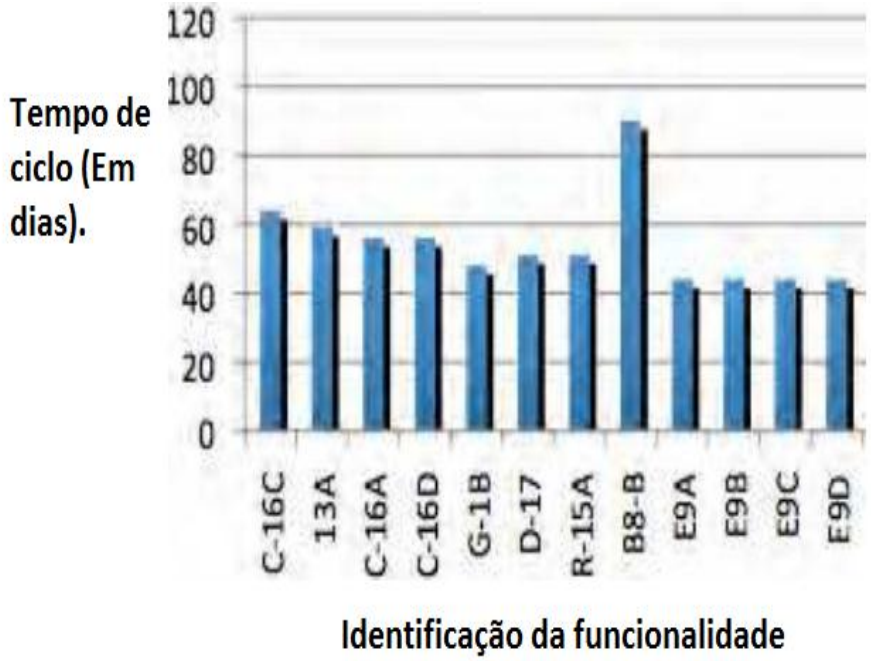


Figura 8. Tempo de Ciclo.

Fonte: Kniberg, H. (2012)

4.3.1.3 Objetivo principal

Pessoas são mais inclinadas a acreditarem que um objetivo é possível, se estas sabem o que realmente é este objetivo. Para que isto seja feito é aconselhado que o objetivo principal do *Sprint* esteja sempre visível

para todos da equipe, isto pode ser feito colocando um post-it no topo do kanban com o objetivo, como na Figura 9.

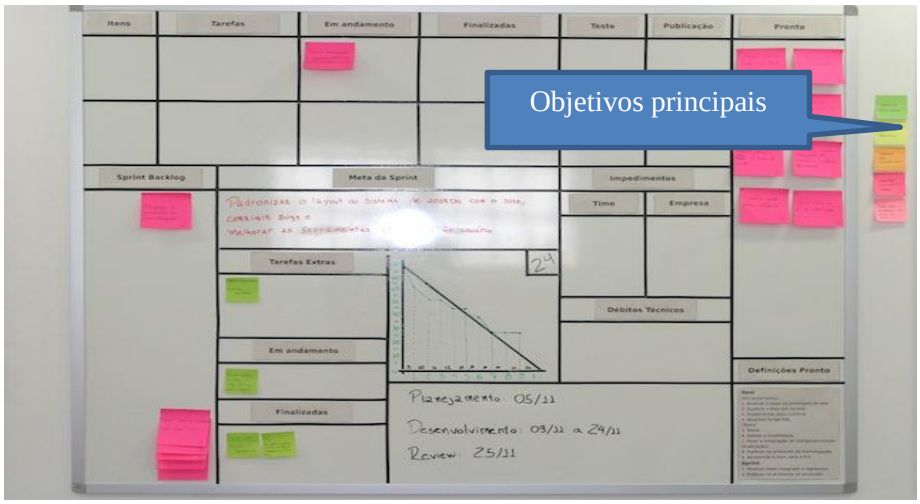


Figura 9. Kanban com objetivos principais.

Fonte: Kniberg, H. (2012)

A cada uma semana ou duas é recomendado que se faça uma revisão deste objetivo, para ver se a equipe está confiante que o trabalho será concluído. Isto pode ser feito da seguinte maneira, o gerente de projeto faz a seguinte pergunta: “Vocês acreditam que conseguiremos cumprir o objetivo?”. Todos da equipem escrevem um número de um a cinco que significa.

- 5 = Com certeza.
- 4 = Provavelmente.
- 3 = Talvez sim, mas com muito esforço (horas extras).
- 2 = Provavelmente não.
- 1 = Sem chances.

Este dado deve ser anotado em uma planilha para se acompanhar como está a confiança do time em relação ao término do trabalho.

Tabela 3. Confiança que o objetivo será alcançado.

	1° semana	2° semana	3° semana
5 = Com certeza	III	I	
4 = Provavelmente	III	II	II
3 = Talvez sim.	II	II	III
2 = Provavelmente não.		III	I
1 = Sem chance.			III

Como mostrado na Tabela 5, a confiança da equipe caiu, isto pode ser sinal de vários problemas. Tais como.

- Algum tipo de impedimento, ex: Algum servidor off-line.
- Muitas funcionalidades no *Sprint*.
- Objetivo não realístico.
- Um gargalo, ex: Alto número de funcionalidades que ainda não foram testadas.

Lembrando que esta informação sobre o término do trabalho é baseado apenas no que o time está sentido sobre o desenvolvimento deste. Não levando em conta métricas mais técnicas, como número de funcionalidades implementadas, contudo, ninguém melhor que a própria equipe para saber se o objetivo vai ser completado ou não.

4.3.1.4 Criando diagramas de causa-efeito

As últimas duas subseções demostraram como mensurar o tempo para implementação de funcionalidades, e como medir a confiança da equipe acerca da entrega dos objetivos dos *sprints*. A presente subseção tem o objetivo de mostrar como utilizar uma forma metódica para a resolução de possíveis problemas que podem aparecer durante o projeto, através de diagramas de causa-efeito.

Diagrama de causa-efeito é uma maneira simples e pragmática de se fazer uma analise causa-raiz. A chave para se resolver um problema é primeiro ter certeza de que se entende o que realmente é este, por que este problema precisa ser solucionado, como você saberá que o problema foi solucionado, e o que causou este problema. Abaixo segue um pequeno exemplo de como solucionar um problema utilizando um ciclo PDCA adaptado.

1. **Primeiro definir o problema:** Entrega de projetos atrasados.
2. **Procurar pela causa raiz do problema**: O que está gerando a entrega de projetos atrasados. Má estimativa do tempo de desenvolvimento de cada funcionalidade, número de funcionalidades em excesso no Sprint, poucas pessoas na equipe, algum tipo de gargalo, algum tipo de impedimento. A organização deve entender a situação para que se possa entender a causa raiz do problema e desta forma possa ir para próxima etapa.
3. **Propor uma contramedida:** Vamos supor que o causa raiz do problema seja má estimativa do tempo de desenvolvimento de cada funcionalidade. Neste caso uma contramedida possível seria utilizar um dos métodos citados na Sessão 4.2.2.2 como construção do tempo de ciclo, para que desta forma a equipe possa ter uma maior consciência de quanto tempo se demora em média para desenvolver uma funcionalidade e adotar este tempo médio como um valor para o *poker planning.*
4. **Especificar os resultados esperados:** O próximo passo é especificar quais melhorias se espera com a contramedida adotada, no nosso caso especificar qual o tempo real para se desenvolver um certo número de funcionalidades e estimar melhor o tempo de entrega do projeto.
5. **Executar a contramedida:** Neste passo será executada a contramedida proposta.

6. **Verificar os resultados:** Comprovar se os resultados especificados estão de acordo com os estimados, caso a resposta seja positiva, pode-se passar para o Passo 7, caso contrário, deve-se voltar ao Passo 4.
7. **Acompanhar e padronizar.** A contramedida deve ser padronizada para que possa se tornar um processo padrão na organização, e também deve ser acompanhada para que esta seja sempre melhorada.

Com estes sete passos é possível criar um diagrama de causa-efeito de um problema com intuído de se criar uma forma metódica de resolução de problemas.

5 CONCLUSÕES E TRABALHOS FUTUROS

Como mostrado no início deste trabalho, apenas 32% dos projetos de software são bem sucedidos, com isto novas propostas para implementação de software devem ser exploradas pelas organizações.

Este trabalho propôs como uma empresa pode adotar a filosofia *lean* de desenvolvimento de software, mostrou como a ferramenta kanban pode ser melhor explorada pela empresa, bem como apresentou formas de como mensurar o tempo de implementação de funcionalidades e evidenciou uma forma metódica de resolução de possíveis problemas que venha a surgir na organização através do PDCA adaptado. Todas estas propostas foram feitas a fim de entregar softwares mais rápidos e com maior qualidade, a fim de atender o que é considerado valor pelo cliente.

Como resultado deste estudo, verificou-se que a adoção da filosofia *lean* é possível na empresa sem grandes investimentos financeiros, apenas mudando a formar de executar alguns processos e a mensuração de outros, como mostrado na sessão 4.2.

Como trabalhos futuros ainda podem ainda ser analisadas algumas etapas do roteiro de Poppendieck e Poppendieck que não foram analisadas neste trabalho, uma vez que tais etapas só podem ser implementadas mediante a implantação dos métodos sugeridos neste trabalho. Abaixo seguem quais são estas etapas.

1. **Sincronizar:** Reduzir o trabalho inacabado. Será proposto que as equipes escrevam os testes antes de começar a codificar. Os desenvolvedores não deverão colocar os erros em uma lista de erros para corrigi-los depois, eles devem corrigir o erro assim que o encontrarem. Integrar o código de forma contínua e extensiva sempre que possível.
2. **Automatizar:** Automatizar cada processo que seja rotina como teste unitários, *builds*, instalação, para evitar que membros da equipe cometam erros.

3. **Mapear fluxo de valor:** Criar a cultura de sempre mapear o fluxo de valor de uma determinada função, para que com isto a organização possa saber qual tempo é gasto realmente agregando valor para o cliente.

REFERÊNCIAS BIBLIOGRÁFICAS

ASSOCIAÇÃO BRASILEIRA DE NORMA TÉCNICAS. NBR ISO 8402. Gestão da Qualidade e Garantia da Qualidade. Rio de Janeiro 2001.

ABRAHAMSOON, P.; O.; RONKAINEN, J.; WARSTA, J. **Agile software development methods: review and analysis**, VTT Technical report, 2002.

BIFFL, S., AURUM, A., BOEHM, B., ERDOGMUS, H.; GRÜNBACHER, P. **Value Based Software Engineering**. Springer-verlag, 388p., 2006.

DYBA, T.; DINGSOYR, T. **Empirical studies of agile software development: A systematic review**. Informatics. Software technology. 50, 9-10 (Aug. 2008), 833-859.

FILHO, D. L.: **Experiências com desenvolvimento ágil**. Instituto de Matemática e Estatística da Universidade de São Paulo. 2008.

FUGGETA, A. **Software Process. A Roadmap**. In: 22nd Internacional Conference on the Future of Software Engineering, Limerick, Ireland: ACM – Association for Computing Machinery, p. 25-34, 2000.

Gil, Antonio Carlos. Métodos e técnicas de pesquisa social. 5. ed. São Paulo: Atlas, 1999. **IMPLEMETING KANBAN FOR SUSTAINING ENGINEERING**. Disponível em http://blogs.mulesoft.org/implementing-kanban-for-sustaining-engineering/. Acessado em Maio de 2012.

JUNG. C. F. **Metodologia para pesquisa & desenvolvimento aplicada a novas tecnologias, produtos e processo**. Rio de Janeiro: Axcel Books, xvi. 321 p., 2004.

KNIBERG. H., **Lean from the trenches: Managing Large-Scale Projects With Kanban**. ed The Pragmatic Programmers, 2012.

Karlsson, C., AHLSTRÖHM, P., 2009. **The difficult path to lean product development**. Journal of Product Innovation Management 13 (4), 283,295.

KÖTLER, P. Administração de marketing. 10. Ed. São Paulo: Prentice Hall, 2000 764 p.

LEAN INSTITUTE BRASIL. Disponível em http://www.lean.org.br. Acessado em Agosto de 2011.

MARÇAL, A. S. et al. **Estendendo o SCRUM segundo as Áreas de Processo de Gerenciamento de Projetos do CMMI**. In Proceedings of CLEI 2007: XXXIII Conferência Latinoamericana de Informática. Costa Rica, 2007.

MIDDLETON, P., 2001. **Lean software development: two case studies**. Software Quality Journal 9 (4), 241-252.

MUNDIM, A. P. F.; ROZENFEL, H; AMARAL, D. C; SILVA, S. L.; GUERRERO, V.; HORTA, L. C. **Aplicando o cenário de desenvolvimento de produtos em um caso prático de capacitação profissional**. Gestão e Produção, V.9, N.1, abr. 2002.

PARNELL-KLABO, E. 2006. **Introducing lean principles with agile practice at a fortune 500 company**. In: Proceedings of the AGILE Conference, pp, 232-242.

PENNA E. M. D. **O Paradigma Junguiano no contexto da metodologia qualitativa de pesquisa**. Psicologia USP, 2004.

PFLEEGER, S. L.; ATLL, J. M. **Software engineering: theory and practice**. 3rd ed. Upper Saddle River, NJ: Pearson, 2006.

PRESSMAN, R. S. **Engenharia de software**. 6. Ed. São Paulo: McGraw-Hill, 2006 xxxi, 720p.

POPPENDIECK, M., POPPENDIECK, T,. **Implementando o desenvolvimento lean de software: Do conceito ao dinheiro.** ED. ARTMED: Pearson Addison Wesley, 2011.

SANTOS, Mariana de Azevedo. **AGILE UBPM FOR SCRUM: Modelo de Aprimoramento do Gerenciamento e Desenvolvimento Ágil Baseado na Percepção de Valor do Usuário**. 2011. 145 f. Trabalho de conclusão de curso (TCC) - Bacharelado em Sistemas de Informação, Universidade Federal de Lavras. Lavras-MG, 2011.

SCHWABER, K. **Agile Project Management with Scrum**. Microsoft Press, 2004;

SHINGO, S. 1981. **Study of Toyota Production System from the Industrial Engineering Viewpoint.** Japanese Management Association.

SILVA, Edna Lúcia; MENEZES, Estera Muszkat. **Metodologia da pesquisa e elaboração de dissertação**. Florianópolis: [S. n.], 2000.

STANDISH GROUP. **New Standish Group report shows more Project failing and les successful projects.** Disponível em

http://www.standishgroup.com/newsroom/chaos_2009.php. Acesso em Outubro de 2010

SOMMERVILL, I. **Engenharia de software**. 8. ed. Rio de Janeiro: Pearson Addison Wesley, c 2007.

TAIICHI OHNO. **Toyota Production System: Beyond Large-Scale Production**. Productivy Press, 1988.

ZEITHAML, **Valerie.Consume perceptions of price, quality and value: a means-end model and synthesis of evidence**. Journal of Marketing, New York, Jul. 1988 v. 52, n. 3, p 2-22