UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE ESPECIALIZAÇÃO EM ENGENHARIA DE SOFTWARE

FRANCISCO JOSÉ RÊGO LOPES

AVALIAÇÃO DA APLICAÇÃO DAS TÉCNICAS *LEAN SOFTWARE DEVELOPMENT* E *KANBAN* EM ATIVIDADES DE MANUTENÇÃO DE SOFTWARE

Trabalho de Conclusão apresentado como requisito parcial para a obtenção do grau de Especialista em Engenharia de Software

Prof. Me. Fábio Petrillo
Orientador

Prof. Dr. Marcelo Soares Pimenta
Profa. Dra. Carla Maria Dal Sasso Freitas
Coordenadores do Curso

Fortaleza, junho de 2012.

AGRADECIMENTOS

Agradeço a Deus, ao Senhor Jesus, que tem me abençoado e fortalecido todos os dias.

A minha esposa Viviane e a nossas filhas, Cecília e Júlia, pelo apoio e companheirismo sempre presentes.

Ao Petrillo, por sua constante disponibilidade, pelas suas ideias, contribuições e incentivo, cruciais para execução deste trabalho.

Ao Régis, Frandberto, CH, Misael e Fred pela ajuda ímpar na realização do experimento, discussões, *insights*.

Ao Renato e ao Neiva pela força nos momentos iniciais do Curso de Especialização em Engenharia de Software.

E ao SERPRO por proporcionar esta oportunidade.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

DFC	Diagrama de Fluxo Cumulativo
GQM	*Goal Question Metric* – Objetivo Questão Métrica
SWEBOK	*Software Engineering Body of Knowledge* – Conjunto de Conhecimentos em Engenharia de Software
TPS	*Toyota Production System* – Sistema Toyota de Produção
WIP	*Work in Progress* - Trabalho em progresso

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

Manutenção é um assunto corrente na literatura de engenharia de software. Recentemente, métodos ágeis de desenvolvimento tem ganho força e espaço, tanto na academia quanto na indústria.

A vivência e observação de equipes de manutenção de software com seus respectivos problemas associados, leva a uma reflexão sobre a aplicabilidade de princípios e práticas de *Lean* e *Kanban* em projetos desse gênero, com o intuito de resolver ou minimizar esses problemas de forma eficaz.

Assim, o objetivo deste trabalho é avaliar a utilização de *Lean* e *Kanban* em projetos de manutenção de software, considerando o levantamento de um conjunto de problemas comuns, citados pela literatura referente ao tema, o estudo e o relacionamento das práticas aplicáveis a esses projetos e como estas práticas podem resolver as questões problemáticas suscitadas.

A condução de estudo teórico, associada à execução de um experimento prático com um time de manutenção, direcionou ao levantamento de um conjunto de dados e informações que subsidiaram a conclusão de que as práticas propostas são aplicáveis a contextos de manutenção, sendo uma proposta de solução provavelmente viável aos problemas apresentados.

Palavras-chave: métodos ágeis, engenharia de software, *kanban*, manutenção.

1 INTRODUÇÃO

1.1 Motivação

A motivação para este trabalho vem da vivência e observação de equipes de manutenção em empresa de grande porte na área de desenvolvimento de software. Parte dessas equipes se depara com problemas comuns, tais como volume significativo de demandas, não raro maior que a capacidade de atendimento, mudanças de prioridades, funcionalidades com necessidade de entrega urgente, escassez de recursos humanos, dificuldade de comunicação, baixa motivação da equipe, dentre outros mais.

A comunidade de desenvolvimento ágil vem centrando muito de sua atenção para a utilização de *Lean* e *Kanban* como instrumentos de um processo de melhoria contínua do desenvolvimento de software. O conjunto de princípios e práticas elencados nessas técnicas leva à hipótese de que elas podem ser benéficas para as atividades de manutenção, contribuindo para mitigar os problemas citados acima.

1.2 Objetivo Geral

O objetivo deste trabalho é avaliar a aplicação de técnicas de *Kanban* e *Lean Software Development* nas tarefas de manutenção – evolutivas, adaptativas e corretivas – de um software já executando em ambiente produtivo.

Nessa avaliação pretende-se elencar quais técnicas efetivamente são adequadas a estes tipos de tarefas, melhorando o controle das atividades, a comunicação com as partes interessadas, a alocação da equipe, a previsibilidade de entregas, provendo resposta a mudanças de prioridades.

1.3 Objetivos Específicos

Os objetivos específicos do trabalho são:

- levantar, dentro das técnicas *Lean* e *Kanban,* os princípios e práticas aplicáveis às tarefas de manutenção;
- avaliar como esses princípios e práticas podem ser aplicados em projetos de manutenção;

- detectar as vantagens e desvantagens da aplicação das técnicas, considerando a resolução de problemas conhecidos relacionados à manutenção de software.

2 MANUTENÇÃO DE SOFTWARE, *LEAN* e *KANBAN*

2.1 Manutenção de Software

2.1.1 O Que é Manutenção de Software

De acordo com SWEBOK (2004), a manutenção de software é definida como o conjunto total de atividades necessárias para prover um suporte ao software dentro de uma relação custo/benefício adequada.

A manutenção compreende todas as modificações feitas em um sistema após a sua implantação IEEE1219 (1998 apud SWEBOK, 2004). Essas modificações podem ser de várias ordens, compreendendo correção de falhas, evolução de design, implementação de melhorias, provimento de interface com outros sistemas, adaptação para plataformas distintas, migração de software legado.

2.1.2 Dinâmica da Evolução de Programas

Diversos estudos tem sido realizados a respeito da manutenção de software. Um dos mais citados na literatura trata da dinâmica de evolução de programas, que é o estudo de mudanças do sistema. Eles foram em sua maioria conduzidos por Lehman e Belady (SOMMERVILLE, 2007). Esses estudos resultaram na proposição das Leis de Lehman, que dizem respeito e estão intrinsecamente ligadas à manutenção de software:

Tabela 2.1: Leis de Lehman

Lei	**Descrição**
Mudança contínua	Um programa usado em um ambiente real deve mudar necessariamente ou se tornar progressivamente menos útil.
Complexidade crescente	À medida que um programa muda, sua estrutura tende a se tornar mais complexa. Recursos extras devem ser dedicados para preservar e simplificar a estrutura.
Evolução de programa de grande	A evolução de um programa é um processo auto-

Lei	Descrição
porte	regulável. Atributos de sistemas como tamanho, tempo entre versões e número de erros reportados é quase invariável em cada versão de sistema.
Estabilidade organizacional	Durante o ciclo de vida de um programa, sua taxa de desenvolvimento é quase constante e independente de recursos dedicados ao desenvolvimento do sistema.
Conservação de familiaridade	Durante o ciclo de vida de um sistema, mudanças incrementais em cada versão são quase constantes.
Crescimento contínuo	A funcionalidade oferecida pelos sistemas deve aumentar continuamente para manter a satisfação do usuário.
Qualidade em declínio	A qualidade dos sistemas entrará em declínio a menos que eles sejam adaptados a mudanças em seus ambientes operacionais.
Sistemas de *feedback*	Os processos de evolução incorporam sistemas de *feedback* com vários agentes e *loops* e você deve tratá-los como sistemas de *feedback* para conseguir aprimoramentos significativos de produto.

Fonte: SOMMERVILLE, 2007. p. 325.

Essas leis demonstram claramente que a manutenção é uma atividade comum e presente de forma constante nos ambientes de engenharia de software. Conforme Pressman (2010), na medida em que o tempo passa, uma empresa descobre que gasta mais dinheiro e tempo mantendo programas existentes do que desenvolvendo novas aplicações. Cita ainda que não é incomum que uma empresa de software gaste entre 60% e 70% de todos os seus recursos com manutenção de software.

De acordo com Erlikh (2000 apud SOMMERVILLE, 2007, p. 323), 90% dos custos de software estão na sua evolução. Assim, depreende-se que empregar esforço no estudo dessa área da engenharia de software é devidamente justificável.

2.1.3 Tipos de Manutenção

Autores e padrões normativos tem feito classificações dos tipos de manutenção. ISO/IEC 14764 (1999 apud SWEBOK, 2004) classifica-os da seguinte forma:

- **manutenção corretiva**: modificações reativas de um produto de software, realizadas para corrigir problemas que foram descobertos;
- **manutenção adaptativa**: modificações de um produto de software feitas após a sua entrega, para manter este software como um produto utilizável em um ambiente modificado ou em modificação;

- **manutenção perfectiva**: modificações de um produto de software feitas após a sua entrega, para melhorar performance ou manutenibilidade;
- **manutenção preventiva**: modificações de um produto de software feitas após a sua entrega, para detectar e corrigir falhas latentes, antes que se elas se tornem falhas efetivas.

Também faz uma categorização em dois itens, conforme a tabela abaixo:

Tabela 2.2: Categorias de manutenção de software

	Correções	**Melhorias**
Proativas	Preventiva	Perfectiva
Reativas	Corretiva	Adaptativa

Fonte: SWEBOK, 2004.

Já Sommerville (2007) os resume em três categorias, a saber:

- **manutenção para reparos de defeitos de software**: aqui ressalta uma diferenciação entre o tipo de erro e o seu custo, considerando que erros de codificação são normalmente associados a baixo custo; erros de projeto são mais caros, pois podem envolver a escrita de vários componentes dos programas; e erros de requisitos são os mais onerosos, pois podem requerer o reprojeto do sistema existente;
- **manutenção para adaptar o software a um ambiente operacional diferente**: necessária quando algum aspecto do ambiente do sistema – hardware, plataforma, ou algum outro software de apoio – muda e o sistema precisa ser modificado para se adaptar a esta nova situação;
- **manutenção para adicionar funcionalidade ao sistema ou modificá-la**: necessária quando os requisitos do sistema mudam em resposta às mudanças organizacionais ou de negócios. A escala desse tipo de manutenção é muito maior que a dos outros dois.

Como os nomes dos tipos de manutenção variam de autor para autor, serão considerados nesse texto os termos resumidos por Sommerville (2007), respectivamente: manutenção corretiva, manutenção adaptativa e manutenção evolutiva.

2.1.4 Questões Importantes na Manutenção de Software

Atividades de manutenção tem particularidades quando comparadas às atividades de desenvolvimento de um novo sistema. Elas exigem dos engenheiros de software atenção a desafios diferenciados, tanto de ordem técnica quanto de gestão.

2.1.4.1 Conhecimento Limitado do Software

Quando um software é implantado, o momento de manutenção se inicia. Não existe certeza de que a equipe que irá manter o sistema será a mesma que o desenvolveu. Com isso, surge uma possível necessidade de repasse de conhecimento entre equipes ou ainda entre empresas, pois

outsourcing pode ser uma possibilidade a ser considerada.

Uma outra vertente a ser considerada é o caso em que não há possibilidade de repasse técnico, como ocorre em sistemas legados. Nessa situação, a equipe de manutenção terá que empreender tempo considerável na busca do entendimento do funcionamento do sistema. SWEBOK (2004) cita pesquisas que indicam que cerca de 40% a 60% do esforço de manutenção é dedicado ao entendimento do software a ser modificado.

2.1.4.2 Testes

À medida que o software evolui através de manutenção, o volume de testes necessários aumenta. Cada manutenção realizada pode implicar na necessidade de execução de testes de regressão.

SWEBOK (2004) registra que o custo de repetição de testes em uma parte maior de um sistema pode ser significativo em termos de tempo e dinheiro. Esse custo pode ser reduzido com a utilização de testes de regressão automatizados. Com isso, teste também passa a ser um ponto importante a ser considerado nas manutenções.

2.1.4.3 Análises de Impacto

Análise de impacto lida com o claro entendimento dos pontos de um sistema que devem ser alterados por conta de uma determinada manutenção e esses impactos nem sempre envolverão apenas o software sendo mantido. A manutenção pode implicar na alteração de mecanismos de integração com outros sistemas.

As análises precisam ser conduzidas com cuidado, para minimizar a ocorrência de efeitos colaterais indesejáveis em virtude da manutenção.

2.1.4.4 Manutenibilidade

IEEE610.12 (1990 apud SWEBOK, 2004) define manutenibilidade como a facilidade como um software pode ser mantido, melhorado, adaptado ou corrigido para satisfazer requisitos específicos. Esta característica precisa ser levada em conta em todos os momentos do ciclo de vida do software, desde seu desenvolvimento até suas atividades de manutenção para que as próprias manutenções tenham custos menores.

2.2 *Lean*: Desenvolvimento Enxuto

2.2.1 História

A produção enxuta (*Lean*) surgiu na década de 40 no Japão, motivada por um problema encontrado na empresa de manufatura de veículos Toyota: como as pessoas não tinham muito dinheiro para comprar carros, estes tinham que ser baratos para poderem ser viáveis. A forma mais simples de se ter carros baratos era a produção em massa. Porém, produção em massa significava

ter milhares de carros produzidos e o mercado japonês não tinha vazão para tamanha produção (POPPENDIECK, 2003).

A busca para a solução desse dilema acabou resultando no *TPS (Toyota Production System)*, que compreende o conjunto de práticas que definem um sistema de produção *Lean*, dentre as quais a mais importante é a eliminação do desperdício.

Nesse tempo ainda não existia o termo *Lean*. Ele só foi cunhado no final da década de 80, quando foi fundado o *IMVP* (*International Motor Vehicle Program*), um programa de pesquisas ligado ao *MIT* (*Massachusetts Institute of Technology*), destinado a avaliar os rumos da indústria automobilística mundial. A "produção enxuta" (do original em inglês *Lean*) foi um termo criado pelos pesquisadores do *IMVP* para definir um processo de produção mais flexível, ágil e inovador do que a produção em massa (LEANWAY, 2012).

2.2.2 Como *Lean* Foi Adaptado Para Desenvolvimento de Software

O termo *Lean Software Development* foi originariamente definido por Mary e Tom Poppendieck, em seu livro *Lean Software Development – An Agile Toolkit*. Neste livro, os autores definem como aplicar em desenvolvimento de software o conjunto de princípios *lean* definidos para manufatura.

Gomes (2010) explica uma classificação de *Lean* dentro dos chamados métodos ágeis de desenvolvimento de software:

> Mas seria *Lean* uma nova metodologia, ou apenas mais um método ágil?
> Segundo Jeff Sutherland, criador do método ágil Scrum, todos os métodos ágeis são aplicações do pensamento *Lean* para software. Entretanto, *Lean* vai além do desenvolvimento ágil, oferecendo uma perspectiva mais abrangente que permite resultados ainda melhores. Kent Beck, criador do método ágil XP, afirma que muitas das preocupações das fábricas também são comuns ao desenvolvimento de software, por exemplo: lidar com incertezas e mudanças, melhorar processos continuamente e entregar produtos que agreguem valor aos clientes. (2010)

2.2.3 Princípios

Lean está calcado em sete princípios que definem o conjunto de ideias que dá direcionamento à produção enxuta.

2.2.3.1 Eliminar Desperdícios

Desperdício é tudo aquilo que não agrega valor ao produto final. Essa ótica de valor é a que é percebida pelo cliente.

Traduzindo esse conceito para desenvolvimento de software, ele compreende vários itens: documentação desnecessária, tempo de espera, funcionalidades a mais, alternância entre tarefas.

Os criadores do *TPS* identificaram sete tipos de desperdício na manufatura. Poppendieck (2003) elaborou um mapeamento desses desperdícios para desenvolvimento de software, o qual

consta na tabela abaixo:

Tabela 2.3: Mapeamento dos tipos desperdícios da manufatura para desenvolvimento de software (Poppendieck, 2010)

Manufatura	**Desenvolvimento de Software**
Inventário	Trabalho parcialmente concluído
Processamento extra	Processos extras
Superprodução	Funcionalidades extras
Transporte	Alternância de tarefas
Espera	Espera
Movimentação	Movimentação
Defeitos	Defeitos

Com essa definição, equipes de desenvolvimento interessadas em eliminação do desperdício devem buscar atacar as fontes citadas.

2.2.3.2 Construir com Integridade

Clark e Fujimoto (1980 apud POPPENDIECK 2003) definem que a integridade de um produto tem duas dimensões: externa e interna. Poppendieck (2003) deu novos nomes a estas duas dimensões, chamando-as de integridade percebida e integridade conceitual e explicando-as assim:

> Integridade percebida significa que o produto como um todo alcança um balanceamento entre funções, usabilidade, confiabilidade e economia que satisfaz os clientes. Integridade conceitual significa que os conceitos centrais do sistema trabalham juntos como um todo, de uma forma coesa e harmônica. (2003, p. 125).

Um software deve ser construído considerando alcançar essas duas dimensões de integridade.

Para isso algumas práticas de engenharia são sugeridas:

- Desenvolvimento Orientado a Testes (*TDD – Test Driven Development*);
- Desenvolvimento iterativo;
- Refatoração;
- Integração contínua;
- Revisão e inspeção de código;
- Padrões;
- Testes contínuos e automatizados.

2.2.3.3 Criar Conhecimento

Durante o processo de desenvolvimento, a equipe deve ter a capacidade de aprender com seus

erros, de modo a não repeti-los.

Desenvolvimento de software é um processo criativo, um exercício de descoberta. A manufatura de produtos não é assim, ela não tem foco em criação, mas em redução da variabilidade. Uma analogia interessante ao desenvolvimento de software é uma receita culinária. Software assemelha-se à criação da receita – que ocorre em ciclos de experimentação e aprendizado – enquanto que a produção é comparada a várias execuções da receita que foi criada (POPPENDIECK, 2003).

Sendo assim, o processo de aprendizado deve ser maximizado e, para isso, pode ser considerada a realização de:

- Desenvolvimento iterativo;
- Ciclos de *feedback*, inspeção e adaptação;
- Equipes pequenas e multidisciplinares;
- Treinamentos e mentorias;
- Criação e utilização de padrões, *templates*;
- Formas de compartilhamento de informações: quadros, painéis, páginas *wiki*, *blogs*.

2.2.3.4 Adiar Decisões e Compromissos

A tomada de decisões deve sempre ser realizada com base em conhecimentos, fatos e informações, e não em suposições. Para isso é necessário que elas sejam tomadas o mais tarde possível. Decisões tomadas no início de um projeto nem sempre são as melhores e trazem grandes impactos futuros.

2.2.3.5 Entregar o Quanto Antes

Para obter melhoria contínua é necessário *feedback* constante. Entregas rápidas proporcionam essa constância. Com o *feedback* constante, a equipe aprende com seus erros e, com eles, promove os ajustes necessários, entrando no ciclo de melhoria contínua.

Além disso, quanto mais cedo o cliente receber o que solicitou, mais rápido será o retorno a respeito do atendimento das suas necessidades.

2.2.3.6 Dar Poder ao Time

As pessoas são o componente mais importante no processo de desenvolvimento de software e as que estão desenvolvendo um certo trabalho são as que melhor conhecem os detalhes necessários para a tomada de decisões técnicas. Com base nesse argumento, a cadeia deve ser modificada para que as pessoas da linha frente passem a tomar decisões ao invés de serem conduzidas por um gerente ou líder de projeto.

2.2.3.7 Otimizar o Todo

Em um sistema complexo, a integridade é obtida através do emprego da combinação de conhecimentos de diversos profissionais com habilidades diferentes. Quando há a necessidade de otimização, um técnico irá atuar naquela área que é do seu conhecimento, ao invés de focar no todo. Isso é chamado de subotimização (POPPENDIECK, 2003).

A subotimização deve ser evitada pois não leva à melhoria do sistema como um todo, mas apenas de suas partes. O produto deve ser sempre visto pela ótica do todo e nunca por uma única parte.

2.3 *Kanban*

Kanban é uma palavra japonesa que, traduzida, significa "cartão de sinal" (ANDERSON, 2011). Em manufatura, *kanbans* são utilizados para informar a um determinado estágio de uma cadeia produtiva que o próximo estágio precisa de mais insumos e que estes já podem ser produzidos.

O responsável por um estágio do processo produtivo só pode trabalhar se o estágio subsequente o acionar através de um *kanban*. Esse encadeamento de estágios em que o próximo solicita o trabalho do anterior acaba por constituir o que se chama de "sistema puxado": um estágio "puxa" o trabalho do anterior.

Anderson (2011) descreve o *Kanban* (com K maiúsculo) como um método de mudança evolutiva que utiliza um sistema puxado com quadro *kanban* (com k minúsculo) de visualização e outras ferramentas para catalisar a introdução de ideias *Lean* no desenvolvimento de tecnologia e operações de TI.

A literatura sobre *Kanban* recomenda práticas que podem ser seguidas para uma implementação da técnica. Entretanto, as recomendações também giram em torno de um processo de melhoria contínua, obtido através de inspeção e adaptação.

2.3.1 Visualizar o Fluxo de Trabalho

Ohno (1988 apud POPPENDIECK, 2003) define que uma das características de um sistema puxado é o controle visual ou gestão visual. Esse controle visual deve ser capaz de mostrar claramente as tarefas em andamento, o que precisa ser feito, os problemas, qual progresso está sendo atingido. Enquanto não houver um controle visual apropriado do trabalho, o time não se tornará autogerenciado, a cadeia comando-controle não será rompida e não será possível o estabelecimento de um sistema puxado.

A principal ferramenta para visualização do fluxo de trabalho é o quadro *kanban*. Para sua montagem, entretanto, é necessário primeiramente o mapeamento da cadeia de valor, que define o processo pelo qual o trabalho evolui de um material cru (como ideias de negócio, por exemplo) para um produto final com valor de negócio (funcionalidade em produção).

Para fazer o mapeamento da cadeia de valor, todos os estágios do processo de manutenção

devem ser levados em conta. Importante considerar a recomendação de Anderson (2011), que assevera não deva ser mapeada a cadeia do processo padrão existente, nem do processo desejado, mas sim do processo que a equipe de desenvolvimento utiliza na prática, mesmo que este não esteja aderente ao processo padrão da organização.

Outra questão significativa é o estabelecimento dos pontos inicial e final de controle do sistema *kanban*. A cadeia de valor a ser mapeada é aquela sobre a qual o time tem gestão. Se o processo de análise de negócio, por exemplo, está sob gestão de outra equipe, ele não deve fazer parte da cadeia de valor mapeada, mas colocado como um ponto externo, com o qual o sistema *kanban* faz interface.

2.3.1.1 Quadro *Kanban*

Com a cadeia de valor mapeada, o quadro *kanban* deverá ser montado de modo a refleti-la visualmente. Na Figura 2.1 está um exemplo de um quadro kanban que exibe uma cadeia de valor aplicável à manutenção de software:

Backlog	Especificação	Desenvolvimento				Revisão por pares	Testes integrados	Homologação	Pronto para implantação	Implantação
		Pronto para desenvolvimento	Análise e projeto	Implementação	Pronto					

Figura 2.1: Exemplo de quadro kanban com cadeia de valor para manutenção de software

2.3.1.2 Itens de Trabalho

Com a cadeia de valor mapeada e representada no quadro *kanban*, as tarefas, funcionalidades, itens de backlog - aqui chamados genericamente de itens de trabalho - que vão dar andamento ao fluxo devem ser descritas em cartões de trabalho. A anatomia de desenho desse cartão de trabalho deve ser variável e adaptada a cada tipo de trabalho. Anderson (2011) cita alguns itens que devem fazer parte do cartão de trabalho, mas também reforça que este deve conter as informações necessárias para que o time possa puxar o trabalho, ao invés de ter o gerente ou líder de projeto "empurrando" o trabalho para o time, dizendo o que cada pessoa tem que fazer.

A Figura 2.2 mostra um exemplo de cartão de trabalho contendo algumas informações importantes para uma cadeia de valor de manutenção de software. É interessante que o nome do responsável pelo item seja uma etiqueta que possa ser modificada, vez que este responsável deverá mudar ao longo da cadeia de valor.

Anderson (2011) também recomenda que atrasos e bloqueios sejam identificados visualmente. Na Figura 2.2 o atraso de um item está representado através da aposição de uma etiqueta vermelha em seu canto superior direito. Bloqueios podem ser representados através da ligação de um outro cartão de trabalho que identifique a questão e permita que ela seja tratada.

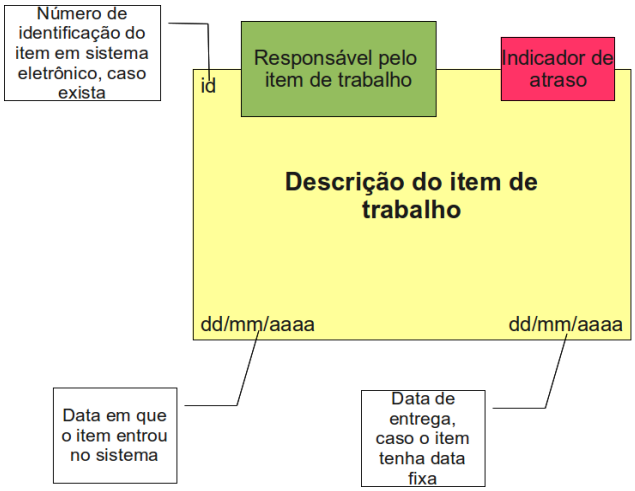


Figura 2.2: Exemplo de um cartão de trabalho

2.3.1.3 Reuniões Diárias

A implantação do quadro *kanban* pode levar à conclusão de que não há necessidades de reuniões diárias, comuns em processos ágeis, pois as típicas perguntas sobre o que está em andamento, o que será feito a seguir e impedimentos, são respondidas pelo próprio quadro.

Anderson (2011) ensina que reuniões diárias em *Kanban* acontecem de maneira diferente: o foco é no fluxo de trabalho. O facilitador da reunião – líder de projeto, por exemplo, percorre o quadro juntamente com a equipe, buscando identificar itens bloqueados, itens que parecem estar parados e não tem se movimentado por alguns dias, itens atrasados devido a defeitos.

2.3.2 Limitar o Trabalho em Progresso[1]

Anderson (2011) descreve *Kanban* como uma abordagem que conduz mudança no processo existente pelo aperfeiçoamento. A partir dessa descrição, propõe introdução paulatina de *Kanban* e considera que o alvo principal da mudança envolve a limitação da quantidade de trabalho em progresso e a interface e a iteração com as partes do negócio da cadeia de valor. Limitar o trabalho em progresso significa não permitir que o sistema *Kanban* receba mais itens do que ele pode suportar.

Anderson (2011) associa a redução do trabalho em progresso ao aumento da qualidade. Daí, deduz que é necessário investir na gestão do trabalho em progresso, introduzindo políticas explícitas para limitá-lo.

A limitação do trabalho em progresso deve ser demonstrada visualmente no quadro *kanban*. Há várias formas de se fazer isso. Uma das mais comuns consiste em escrever a quantidade de itens

1 Tradução de *WIP - Work in progress*

de trabalho suportada em cada um dos estágios da cadeia de valor representados no quadro.

De uma maneira geral, estabelecer os valores para esses limites não é fácil. Boeg (2011) cita formas de se fazer isso: colocar o sistema em execução e observar como se comporta e, só então, estabelecer os limites; uma segunda abordagem, porém mais radical, é limitar a quantidade de trabalho para o valor que se espera do sistema e adicionar colunas de *buffer* após cada um dos estágios da cadeia de valor.

A despeito dessa dificuldade de estabelecer os valores de limite para o trabalho em progresso, Anderson (2011) considera que a não existência de limites é, claramente, um erro. Assim, os limites devem ser estabelecidos e ajustados ao longo do tempo, de forma que o trabalho flua de suavemente ao longo da cadeia de valor.

Anderson (2011) também explica que a fila de entrada (na Figura 2.3 representada pela coluna "*Backlog*") também deve ter um limite estabelecido. Os demandantes devem ter a compreensão de que a capacidade de atendimento da equipe é limitada e de que eles não podem solicitar itens de trabalho de uma maneira ilimitada.

Backlog (5)	Especificação (3)	Pronto para desenvolvimento	Desenvolvimento (3)			Revisão por pares (2)	Testes integrados(4)	Homologação (5)	Pronto para implantação
			Análise e projeto	Implementação	Pronto				

Figura 2.3: Exemplo de quadro *kanban* com limitação de trabalho e progresso para os estágios da cadeia de valor

Embora os limites possam ser estabelecidos unilateralmente pelo time, Anderson (2011) ressalta a importância fazer isso de forma consensual entre *stakeholders* em todos os níveis, e gerência sênior. Esse acordo se fará valer quando o sistema *Kanban* for colocado sob estresse e os *stakeholders* que o aceitaram tiverem que honrar o compromisso estabelecido, não introduzindo mais itens do que os que foram combinados.

2.3.3 Ajustar Cadências e Priorizar

Anderson (2011) define a cadência como um conceito em *Kanban* que determina o ritmo de um tipo de evento. Priorização, entregas, retrospectivas, e qualquer outro evento recorrente podem possuir sua própria cadência.

Boeg (2011) cita que, embora existam vários tipos de atividade em um ambiente típico de desenvolvimento de software que possam se beneficiar de cadências regulares, as mais típicas e que precisam ser consideradas são as de entrada e saída. Anderson (2011) chama de cadências de priorização e entrega, respectivamente.

2.3.3.1 Cadência de Entrada

O estabelecimento da cadência de entrada implica na definição de um evento periódico de priorização pelos envolvidos na tomada de decisão. Essa regularidade precisa ser definida com base nos custos de transação e coordenação envolvidos. A situação ideal de priorização envolve intervalos menores de eventos. Porém, deve ser buscado um balanceamento entre frequência e custo associado a cada reunião.

Anderson (2011) destaca que eficiência e cadência de priorização podem ser incrementadas focando-se em reduzir seus custos de transação e coordenação, e que esses custos podem ser reduzidos com reuniões de priorização regulares.

2.3.3.2 Cadência de Entrega

O estabelecimento dessa cadência implica em um acordo que define qual a periodicidade de entrega de software funcional.

É importante ressaltar que esta cadência é dissociada do conceito de *time-box* dos métodos de desenvolvimento ágil. Anderson (2011) explica que *Kanban* dispensa o uso de iterações *time-boxed*, desacoplando as atividades de priorização, desenvolvimento e entrega. É permitido o ajuste da cadência de cada uma dessas atividades naturalmente. Contudo, *Kanban* não dispensa uma cadência de entrega regular.

Para definição dessa cadência devem ser levados em consideração os custos de entrega. Quanto se gasta para publicar uma *build* em ambiente produtivo? Quais as pessoas envolvidas, quais os recursos necessários? Como realizar treinamento, empacotamento, *marketing*, etc.? Entretanto, deve se buscar cadências de entrega pequenas.

É importante e benéfico se fazer entregas regulares. O comprometimento com datas específicas como, por exemplo no dia 10 de cada mês, permite que as pessoas envolvidas se programem. Além disso, reduz custos, pois não há a sobrecarga de reuniões para decidir quando o *release* deve "subir" para o ambiente produtivo. No final, essa pulsação regular de entregas acaba por ajudar a construir confiança.

Entretanto, existem circunstâncias em que se faz necessária uma entrega que fuja da cadência regular como, por exemplo um marco legal. Nesses casos, esse *release* especial deve ser planejado e os *releases* periódicos retomados logo que possível.

O conceito de *lead time* deve ser levado em consideração para a entrega, pois, caso tenha sido decidido por uma cadência de entrega semanal, por exemplo, é possível que nem todos os itens que entraram no sistema *Kanban* estejam prontos na próxima entrega. Nesse sentido, a equipe deve buscar definir um balanceamento entre a cadência de entrega e o seu *lead time* médio.

2.3.4 Identificar Classes de Serviço

No desenvolvimento de software de uma forma geral, é facilmente perceptível que os itens de

trabalho são variáveis. Nem todas as demandas são de *bugs,* nem todos os casos de uso tem os mesmos tamanho e forma, nem todas as funcionalidades tem a mesma complexidade, e assim por diante.

Não sendo então todas as demandas iguais, se impõe a situação de que alguns tipos de itens de trabalho serão mais prioritários do que outros e devem ser tratados de formas diferentes dentro do sistema *Kanban*.

Anderson (2011) descreve os diversos tipos de itens de trabalho, classificando-os em "classes de serviço" e generaliza sua descrição indicando que a definição dessas classes deve considerar a maior oferta de vantagens para a agilidade do negócio e gerenciamento de riscos. Ainda considera que alguns itens de trabalho são mais necessários que outros, enquanto alguns são mais valiosos que outros.

Boeg (2011) exemplifica classificação semelhante à de Anderson (2011), em quatro classes: padrão, prioritária, data fixa e expedição. O que deve ser considerado são os tipos de itens de trabalho que se enquadram em cada categoria e qual o tratamento especial que deve ser dado a eles. Se for possível associar custo financeiro para cada classe, é interessante fazê-lo. A tabela 2.4 mostra um exemplo de classes de serviço.

Tabela 2.4: Exemplificação de classes de serviço (Boeg. 2011. p. 60.)

Classe	**Tipos de itens de trabalho**	**Tratamento especial**
Padrão	*Bugs* cosméticos *User stories*	Nenhum
Prioritária	*Bugs* críticos *User stories* de alta prioridade	Tem prioridade em cada estágio da cadeia de valor
Data fixa	*User stories*	Tem prioridade em cada estágio da cadeia de valor, caso não exista segurança de que a data fixada será alcançada. Caso contrário será tratado como classe padrão. Devem ser realizadas estimativas de esforço e tamanho para avaliar o tempo de fluxo. Se for necessário, um *deploy* emergencial pode ser realizado.
Expedição	*Bugs* que impedem o trabalho em ambiente produtivo	Quebra de limites do trabalho em progresso. Todo o trabalho que estiver em andamento pode ser parado. *Deploy* emergencial pode ser realizado.

O estabelecimento de classes de serviço requer ainda que sejam estabelecidos limites para o número de itens de cada classe. Por exemplo, a classe de expedição só pode ter um item de cada

vez. Se não houver limite para isso, é possível que o sistema chegue a uma situação em que todos os itens serão da classe de expedição e então o mecanismo de classificação perde o sentido.

Os tipos de classe de serviço, sua política associada e limites devem ser claramente conhecidos por todos os envolvidos, de modo a facilitar a comunicação e entendimento do que pode ou não ser feito no sistema *Kanban*.

A classificação da classe de serviço de um item de trabalho deve ser feita no momento em que ele entra no backlog do sistema.

O quadro *kanban* também deve refletir visualmente o enquadramento dos itens de trabalho dentro de cada classe. Isso pode ser feito, por exemplo, através de identificação de cores nos cartões de trabalho ou ainda pela separação de raias específicas para as classes não padrão.

Backlog (5)	Especificação (3)	Desenvolvimento (3)			Revisão por pares (2)	Testes integrados(4)	Homologação (5)	Pronto para implantação (*)
		Pronto para desenvolvimento	Análise e projeto	Implementação				
Classe de expedição								
Classe de data fixa								

Figura 2.4: Exemplo de quadro *kanban* com raias diferenciadas para classes de serviço.

2.3.5 Estabelecer Acordo de Nível de Serviço

Na medida em que o sistema *Kanban* está estabelecido e medições começam a ser feitas, é possível obter um histórico do nível de entrega que o sistema pode oferecer para cada uma das classes de serviço.

Com isso, é possível estabelecer um acordo com os *stakeholders*, definindo, para algumas classes, o que eles podem esperar em termos de tempo de entrega.

Tabela 2.5: Exemplificação de acordo de nível de serviço para a classe de serviço padrão.

Prioridade	**Acordo de Nível de Serviço**
Baixa	Média de entrega: 21 dias 90% das entregas: 25 dias 100% das entregas: 30 dias
Média	Média de entrega: 13 dias 90% das entregas: 18 dias 100% das entregas: 25 dias
Alta	Média de entrega: 10 dias

Prioridade	Acordo de Nível de Serviço
	90% das entregas: 14 dias 100% das entregas: 18 dias

Fonte: Adaptado de Boeg. 2011. p. 75.

2.3.6 Medir o Fluxo

Pressman (2010) define que medição é um elemento chave no processo de engenharia.

Sommerville (2007) define uma métrica de software como qualquer tipo de medição que se refira a um processo ou documentação relacionada.

Conforme ensina Anderson (2011), o fluxo contínuo do sistema *Kanban* significa que, em termos de relato da situação das atividades, o interesse em reportar se um projeto está "*on-time"* ou se um plano específico está sendo seguido, está em segundo plano. O que passa a ser importante é mostrar que o sistema *Kanban* é previsível e está funcionando como projetado.

Existem várias medições que podem ser feitas. Entretanto, há que se tomar cuidado com a quantidade de medições a serem consideradas. A escolha de um número excessivo delas poderá fazer com que percam o sentido. Boeg (2011) afirma que até quatro diagramas de medições é muitas vezes o limite de informações que as pessoas conseguem processar antes de "se afogar" nelas. Abaixo seguem descrições de medições através do Digrama de Fluxo Cumulativo e *Lead Time*.

2.3.6.1 Diagrama de Fluxo Cumulativo (DFC)

Hellesøy (2010) define Diagrama de Fluxo Cumulativo (DFC) como uma ferramenta visual que comunica a capacidade do time em entregar software funcionando em tempo hábil. Um DFC mostra a quantidade atual de trabalho no sistema para cada estágio da cadeia de valor ao longo do tempo.

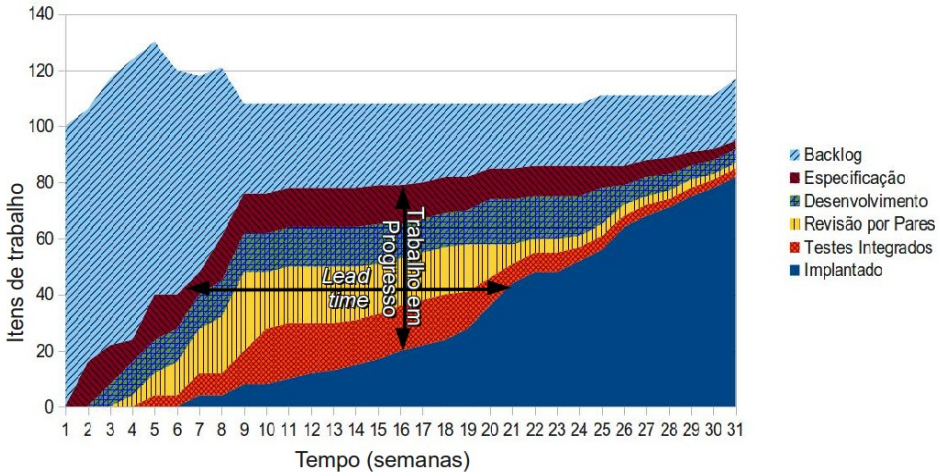


Figura 2.5: Exemplo de diagrama de fluxo cumulativo.
Fonte: Adaptado de Hellesøy. 2010.

A Figura 2.5 mostra um exemplo no qual está destacada em um momento do tempo a quantidade de trabalho em progresso, representada nos estágios da cadeia de valor entre "Backlog"

e “Implantado”. O *lead time* é exibido em um determinado momento da escala temporal, mostrando o tempo que um item de trabalho leva para percorrer toda a cadeia de valor. Lido de cima para baixo e da esquerda para direita, é possível perceber como o backlog percorre a cadeia de valor e se torna trabalho implantado.

Petersen (2010) cita exemplo de DFC que exibe a análise do fluxo contínuo.

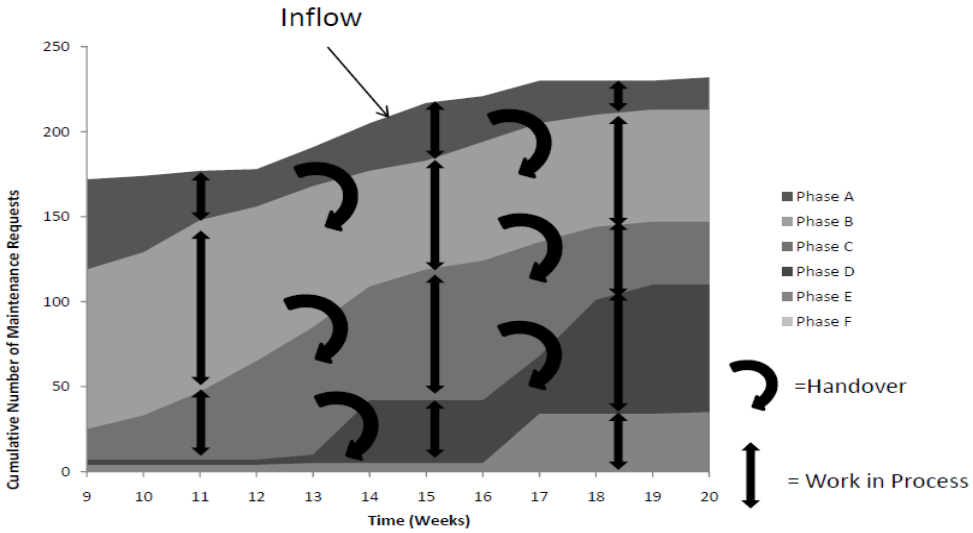


Figura 2.6: Diagrama de Fluxo Cumulativo (Petersen, 2010)

Na Figura 2.6 percebe-se claramente que o fluxo de transições da fase C para a fase D não é contínuo. Existe um longo tempo de inatividade (semanas de 9 a 13) e então, repentinamente, uma grande transição ocorre. Isto pode levar a uma situação de sobrecarga na fase D, quando muito trabalho tem que ser feito de uma vez.

2.3.6.2 *Lead Time*

Lead time é tempo que um item de trabalho leva entre o momento de sua entrada no sistema e a sua saída.

Esta métrica indica o quão previsível é a organização na entrega (ANDERSON, 2011).

Para melhor análise dessa métrica, tanto Anderson (2010) quanto Petersen (2010) consideram classificações dos tipos de itens de trabalho. Anderson (2010) cita as “classes de serviço”, cuja descrição consta desse trabalho na seção 2.3.4. Já Petersen (2010) refere apenas os itens como críticos e não críticos.

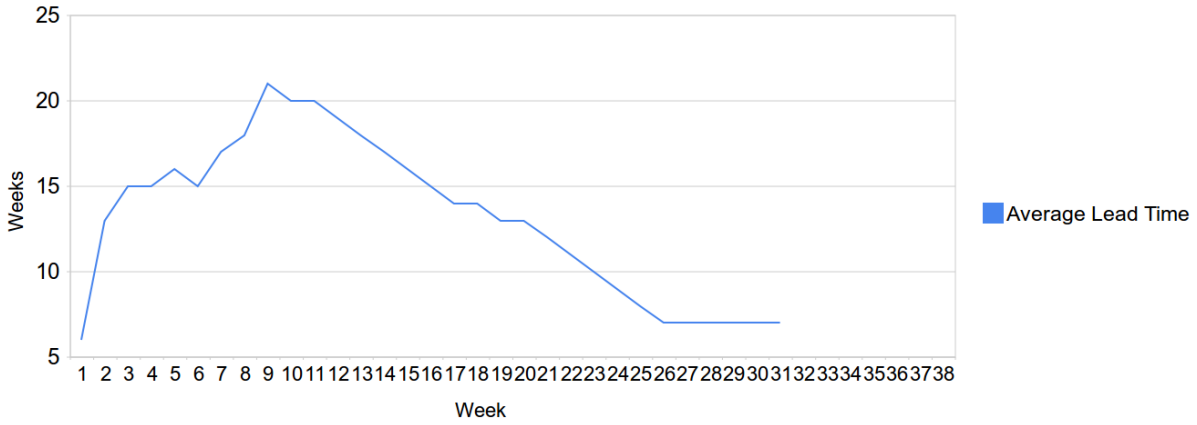


Figura 2.7: Variação do *lead time* médio ao longo do tempo (Hellesøy, 2010) .

Na Figura 2.7 é possível verificar que houve uma redução do *lead time* médio ao longo do tempo. Com essa informação é possível fazer a análise da ação que causou essa redução que pode, por exemplo, estar ligada à redução do trabalho em progresso a partir de um determinado momento.

A Figura 2.8 mostra exemplo de gráfico do *lead time*, porém considerando categorias diferentes de itens de trabalho.

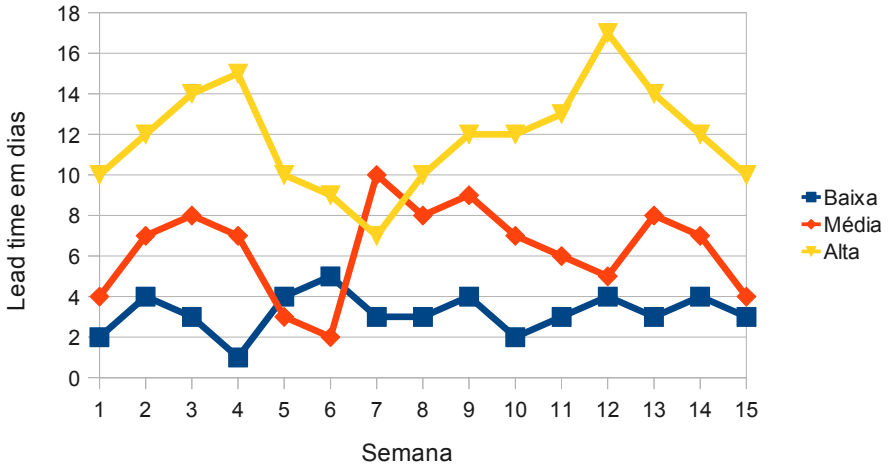


Figura 2.8: Exemplo de gráfico mostrando a tendência do *lead time*, considerando três categorias de itens de trabalho: prioridades baixa, média e alta.

2.3.7 Gerenciar o Fluxo

Conforme Boeg (2011), uma vez que o sistema *Kanban* está em operação, chega o momento de fazer uma leitura e tomar ações apropriadas quando for identificada uma oportunidade de melhoria. Algumas perguntas podem ser feitas para identificação dessas oportunidades:

- O limite do trabalho em progresso está correto?
- É possível encontrar uma forma de ter um tamanho menor para as *user stories*?

- Existe uma forma de identificar funcionalidades que irão explodir em tamanho antes que elas sejam introduzidas no sistema e causem o bloqueio da cadeia de valores por longos períodos de tempo?
- É possível nivelar o tamanho das *user stories* de modo a criar um fluxo mais contínuo?
- É possível treinar para flexibilidade para evitar silos e facilmente aliviar gargalos?
- Existem *buffers* adequados para lidar com variações?
- O exame para otimização está considerando o todo ao invés de estágios individuais?

Atenção especial deve ser dada para o último item. Boeg explica:

> A chave é sempre pensar em termos de fluxo do produto final e não focar em como você pode fazer uma pessoa ou um passo individual serem mais rápidos. Infelizmente muitos gerentes estão muito mais focados em ter pessoas trabalhando mais rápido do que em qualidade e fluxo do produto. Lembre sempre de olhar o produto e não as pessoas (2011, p. 68).

Poppendieck (2003) cita duas premissas sobre aquilo que considera ser o significado de maturidade. A declaração de Boeg vai ao encontro dessas premissas:

> Premissa *Lean 1*: Uma organização madura olha para o sistema como um todo; ela não foca em otimização de partes isoladas.
>
> Premissa *Lean* 2: Uma organização madura foca em aprendizagem efetiva e ampliação dos poderes[2] do time. (2003, p. 98-99).

Gerenciamento do fluxo está relacionado com melhoria contínua do sistema como um todo.

2.4 Trabalhos Relacionados

Diversos trabalhos tem sido publicados sobre métodos ágeis e manutenção de software, tanto pela indústria como pela academia. Contudo, pela pesquisa bibliográfica realizada, poucos se dedicam ao estudo específico de utilização de métodos ágeis em atividades de manutenção.

FRANCO (2007) propôs um modelo de gerenciamento de projetos baseado em *Scrum* e *Extreme Programming*, aliados aos princípios e valores de *Lean*. Já PADUELLI (2007) focou sua pesquisa na manutenção de software, notadamente na definição e estudo dos problemas relacionados com manutenção de software. O trabalho realizado por BASSI (2008) identificou e descreveu 22 práticas para desenvolvimento de software, com base em experiências de condução de projetos ágeis, que podem ser adotadas para aumentar o desempenho e qualidade do software.

Importante citar o trabalho de Petersen (2010), que focou especificamente na utilização de *Lean* para manutenção de software. Ele utiliza o paradigma GQM (*Goal Question Metric*) para identificar objetivos, questões e métricas para análise do processo de manutenção de software utilizando *Lean*. Petersen (2010) relatou, na sua seção de trabalhos relacionados ao tema, que pesquisa envolvendo manutenção de software em combinação com *Lean* ainda não tinha sido realizada, não citando referências específicas sobre os dois temas em um único trabalho.

2 *empowerment*

3 AVALIAÇÃO DE UTILIZAÇÃO DE *LEAN* E *KANBAN* PARA PROJETOS DE MANUTENÇÃO DE SOFTWARE

3.1 Metodologia

A metodologia aplicada no desenvolvimento do trabalho foi desenvolvida nos seguintes passos:

- levantamento bibliográfico;
- avaliação das práticas aplicáveis às atividades de manutenção;
- realização de experimento com as práticas;
- avaliação das vantagens e desvantagens das práticas em atividades de manutenção.

O levantamento bibliográfico forneceu o embasamento teórico a partir do qual as demais atividades propostas seriam conduzidas. Com ele, os principais conceitos e práticas de *Lean* e *Kanban* foram detalhados, de modo a orientar a avaliação. A avaliação das práticas aplicáveis à manutenção permitiu a identificação daquelas que tem utilização direta na minimização dos principais problemas de manutenção de software.

Em seguida, foi conduzido um experimento em ambiente controlado, de forma a verificar algumas das práticas através de uma simulação de manutenção. Por fim foi feita a avaliação do uso das práticas.

3.2 Cenário de Manutenção

A manutenção de software enfrenta diversos desafios, que podem variar de empresa a empresa e de equipe a equipe. Paduelli e Sanches (2006) fazem uma categorização dos problemas de manutenção, dividindo-os em técnicos e gerenciais:

Tabela 3.1: Problemas de manutenção de software.

Problemas Gerenciais	Ausência de um processo de manutenção de software
	Grande expectativa dos usuários

	Elevada rotatividade de membros e funções dentro da equipe
	Sobrecarga de tarefas
	Prazos não condizentes com a complexidade do software
	Baixa motivação entre profissionais de manutenção
	Ausência de manutenção preventiva
	Falhas de comunicação com o usuário
	Atrasos na entrega
Problemas Técnicos	Registro inexistente ou superficial de manutenções anteriores
	Ausência de um ambiente computacional específico para manutenção
	Validação insuficiente de manutenções efetuadas
	Documentação insuficiente ou superficial

Fonte: PADUELLI e SANCHES. 2006. p. 15.

O cenário que se deseja considerar nessa avaliação envolve alguns dos problemas citados acima, a saber:

- Grande expectativa dos usuários;
- Sobrecarga de tarefas;
- Prazos não condizentes com a complexidade do software;
- Baixa motivação entre profissionais de manutenção;
- Falhas de comunicação com o usuário;
- Atrasos na entrega.

Além desses, serão considerados ainda:

- Mudanças de prioridades entre diversas solicitações (DEKLEVA apud PADUELLI, 2006, p. 70);
- Dificuldade de obter previsibilidade de entrega (WANGENHEIM, 2000, p. 4);

3.3 Práticas de *Lean* e *Kanban* Para Projetos de Manutenção

O que se pretende é o estabelecimento de um sistema puxado, o qual inverterá a lógica mais comum em projetos de software em que o trabalho é “empurrado” pelo líder do projeto, pelo gerente, ao invés de ser “puxado” pelo time. Para isso, a abordagem considera os passos citados por Boeg (2011) para adoção de *Kanban*:

- Visualizar o fluxo de trabalho;
- Limitar o trabalho em progresso;
- Ajustar cadências e priorizar;
- Medir o fluxo;
- Identificar classes de serviço;
- Estabelecer acordos de nível de serviço;
- Gerenciar o fluxo;

Cada um desses passos está calcado nos princípios *Lean,* e deriva ações e práticas mais detalhadas. Na tabela abaixo está estabelecida uma relação entre problemas de manutenção e as respectivas práticas que irão auxiliar na sua solução:

Tabela 3.2: Relacionamento entre os problemas de manutenção de software e práticas Lean e Kanban

	Grande expectativa dos usuários	Sobrecarga de tarefas	Prazos não condizentes com a complexidade do software;	Falhas de comunicação com o usuário	Atrasos na entrega	Mudanças de prioridades entre diversas solicitações	Dificuldade de obter previsibilidade de entrega	Baixa motivação entre profissionais de manutenção
Visualizar o fluxo de trabalho				X				
Limitar o trabalho em progresso		X						
Ajustar cadências e priorizar	X					X		
Identificar classes de serviço	X					X		
Estabelecer acordo de níveis de serviço	X		X		X			
Medir o fluxo				X			X	
Gerenciar o fluxo								X

As próximas seções descrevem como as práticas poderão ajudar na minimização dos problemas de manutenção elencados.

3.3.1 Visualizar o Fluxo de Trabalho

3.3.1.1 Falha de Comunicação com o Usuário

É perceptível que os demandantes de uma funcionalidade desejam ter ciência da situação do atendimento de suas demandas. A visualização do fluxo de trabalho atua no auxílio da redução desse problema, na medida em que permite que qualquer pessoa interessada em conhecer o

andamento das funcionalidades demandadas pode obter informações sem que tenha que solicitá-las ao gerente do projeto ou ler relatórios. Uma inspeção visual ao quadro *kanban* descreve, por si só, a situação das demandas.

Várias informações podem ser obtidas apenas com a inspeção do quadro:

- Quais itens de trabalho estão em execução;
- Quem é responsável por cada um deles;
- Quais os itens que estão de alguma forma bloqueados;
- Qual a quantidade de trabalho em andamento no sistema;
- Qual a capacidade do sistema de receber novos itens.

Além do próprio benefício de melhoria da comunicação com o usuário final, o time é diretamente beneficiado com a visualização do trabalho via quadro *kanban*. Cada pessoa pode ir ao quadro e "puxar" um cartão para sua responsabilidade e dar andamento a ele. Todos sabem em que item cada componente do time trabalha.

O quadro possibilita uma visão do todo, mostrando como cada parte do trabalho afeta as outras e vice-versa, determinando um nível elevado de transparência.

3.3.2 Limitar o Trabalho em Progresso[3]

3.3.2.1 Sobrecarga de Tarefas

A limitação do trabalho em progresso e limitação da fila de entrada auxiliarão na resolução da sobrecarga de tarefas, na medida em que a associação de itens de trabalho aos componentes da equipe e a chegada de novos itens no *backlog* se dá apenas quando há capacidade para isso.

A inversão da forma de funcionamento do sistema, que antes era "empurrado" por um gerente ou líder de projeto, para um sistema agora "puxado" pelos componentes da equipe é fator importante para a redução desse problema. Todos os envolvidos com o trabalho sabem que, se o limite para o trabalho em progresso na coluna "Testes integrados" é 4, não haverá condição de que mais de 4 itens de trabalho para testes sejam atribuídos à equipe.

3.3.2.2 Volume de Demandas Maior Que a Capacidade de Atendimento

A limitação do trabalho em progresso tem ação imediata sobre o volume das demadas. As políticas estabelecidas para o volume de trabalho impedem que a equipe assuma mais tarefas do que pode realizar e passam aos demandantes a noção de que não podem fazer solicitações de manutenção de uma forma aleatória e sem controle.

3 Tradução de *WIP - Work in progress*

3.3.3 Ajustar Cadências e Priorizar

3.3.3.1 Grande Expectativa dos Usuários

Paduelli e Sanches (2006), ao citar esse problema, associam-no a outro, citado por Lientz e Swanson (1980 apud PADUELLI e SANCHES, 2006), o qual diz que usuários tem necessidades constantes por melhorias e novas funcionalidades.

De uma forma literal, também entende-se que usuários tem expectativas por receber aquilo que solicitaram, por saber quando ficará pronto. O estabelecimento de cadências tende a minimizar essas questões. O estabelecimento da cadência de entrada deixa claro para os envolvidos quando e quanto eles podem solicitar da equipe de desenvolvimento. A cadência de entrega dirá quando os envolvidos devem esperar a entrega.

3.3.3.2 Mudanças de Prioridades Entre Diversas Solicitações

O estabelecimento da cadência de entrada propicia aos envolvidos a discussão periódica do que deve ou não deve entrar no *backlog* do sistema. Com o estabelecimento do tamanho para o *backlog,* a reunião de priorização se concentrará apenas em escolher quais os próximos itens que cabem no limite de itens que foi estabelecido, considerando aqueles que trarão o maior valor de negócio.

Conhecendo a cadência de entrega e o *lead time* médio do time, os interessados que não forem contemplados terão a expectativa de quando poderão ter incluído o seu item no *backlog* do sistema.

3.3.4 Identificar Classes de Serviço

3.3.4.1 Grande Expectativa dos Usuários

O estabelecimento de um acordo de nível de serviço em termos de que políticas são apropriadas para cada item de trabalho de manutenção é fundamental para que se possa satisfazer às expectativas dos usuários.

Com o acordo estabelecido e as demandas de manutenção classificadas, cada demandante não poderá exigir do sistema *Kanban* algo que não estava previamente combinado.

O sistema, por outro lado, deverá conseguir responder com tempestividade às classes cujos acordos requerem prioridade na cadeia de valor.

3.3.4.2 Mudanças de Prioridades Entre Diversas Solicitações

As classes de serviço definem previamente que tipos de itens de trabalho terão prioridade na cadeia de valor e como esta prioridade afeta o andamento do trabalho com relação aos demais itens.

Com isso, a forma de tratamento da priorização já está combinada previamente e não será motivo de discussão constante. No momento da reunião para priorização, conforme a cadência de

entrada definida, os itens de trabalho serão classificados e receberão maior ou menor atenção do time sem a necessidade de condução por gerente ou líder de projeto. Supondo, como exemplo, a existência de uma classe de expedição que permite quebra dos limites para o trabalho em progresso e tem prioridade sobre todos os outros itens de trabalho, o time sabe que esse item deve ter atenção total, não necessitando de qualquer orientação sobre o que deve ser a prioridade.

3.3.5 Estabelecer Acordo de Nível de Serviço

3.3.5.1 Grande Expectativa dos Usuários

O acordo de nível de serviço servirá como parâmetro para que os *stakeholder*s tenham a expectativa correta em termos do que a equipe é capaz de entregar. Esse acordo também servirá como ferramenta no momento das reuniões de priorização.

3.3.5.2 Prazos Não Condizentes Com a Complexidade do Software

A minimização desse problema com o acordo de nível de serviço se dá na medida em que um número reduzido de classes terá prioridade no acordo. Todas as demais andarão conforme o fluxo normal da cadeia de valor, o que implica em menor necessidade de se fazer previsões baseadas em especulação.

Essa ideia está calcada em um dos princípios *Lean*, que é o comprometimento tardio.

3.3.5.3 Atrasos na Entrega

Supondo, por exemplo, um acordo de nível de serviço acertado com uma meta para itens de classe padrão e prioridade baixa em 80% dos itens entregues com 15 dias, 90% dos itens entregues com 18 dias e 100% com 20 dias. Na medida em que a equipe consiga obter *lead times* que se compreendam dentro desses limites, os atrasos nas entregas serão minimizados, pois há uma margem para que isso ocorra.

Itens de classes que tenham prioridade no fluxo, certamente serão entregues mais rapidamente, enquanto obtém maior atenção de todo o time.

3.3.6 Medir o Fluxo

3.3.6.1 Falhas de Comunicação Com o Usuário

Aliadas ao quadro *kanban*, as métricas exibidas através de gráficos serão de grande valia na minimização desse problema. A análise do diagrama de fluxo cumulativo irá dar aos envolvidos a visão do andamento do fluxo das demandas de manutenção, possibilitando a fácil identificação de gargalos. Métricas relacionadas ao *lead time* comunicarão aos envolvidos a evolução do tempo de entrega e uma noção do nível de expectativa que os demandantes podem ter a respeito do tempo de resposta do sistema *Kanban*. Essa expectativa será importante nos eventos de priorização das demandas.

3.3.6.2 Dificuldade de Obter Previsibilidade de Entrega

A utilização de métricas atuará como fator preponderante para que os acordos de níveis de serviço sejam estabelecidos. As medidas irão revelar o histórico de funcionamento do sistema *Kanban*, seus *lead time* médios por classe de serviço e é com esses números que os acordos de nível de serviço serão montados.

3.3.7 Gerenciar o Fluxo

3.3.7.1 Baixa Motivação Entre Profissionais de Manutenção

Sommerville (2007), ao discorrer sobre manutenção de software, cita que esta atividade é vista como sendo de segunda classe. Esse é um dos motivos pelo quais se torna necessário melhorar a motivação dos times que trabalham com manutenção.

O foco no fluxo como um todo, ao invés de pessoas, será o primeiro ponto que irá contribuir para o aumento dessa motivação. Outro ponto é a ampliação de poderes. O *empowerment* deve ser elevado ao ponto de modificar a tomada de decisões. As grandes decisões de projetos devem ser tomadas pelas equipes, conforme explica Poppendieck (2003):

> Nós acreditamos que o fator crítico em motivação não é medição, mas *empowerment*: mover as decisões para o nível mais baixo possível em uma organização, desenvolvendo a capacidade dessas pessoas de tomar decisões sabiamente. (2003, p. 97).

4 EXPERIMENTO COM PRÁTICAS DE *KANBAN*

4.1. Objetivos do Experimento

O objetivo do experimento foi simular a execução de um sistema *Kanban* para manutenção de um sistema já executando em ambiente produtivo. Com isso, obter percepção dos participantes sobre a aplicabilidade das técnicas em atividades de manutenção.

O conjunto de manutenções demandadas ao sistema compreendeu *bugs* e evoluções. Detalhadamente, o que se buscou verificar foi:

- Visualização do fluxo de trabalho através da utilização do quadro;
- Experimentação de limites para o trabalho em progresso;
- Testes de ajuste da cadência de entrada;
- Utilização de classes de serviço;
- Medição e geração de gráficos do fluxo.

4.2. Descrição do Sistema

O experimento conduzido consistiu na simulação do uso de *Kanban* para atividades manutenção de um sistema de PDV (Ponto de Venda) já em produção. Este sistema faz interface com uma impressora fiscal, e já conta com as seguintes funcionalidades implementadas:

Tabela 4.1: Lista de funcionalidades no sistema

Venda com emissão de cupom fiscal
Retirada de dinheiro do caixa
Suprimento de dinheiro no caixa
Fechamento do dia
Consulta de produtos
Cancelamento do cupom fiscal

Cancelamento de item durante a venda

A tabela 4.2 mostra a descrição do conjunto de solicitações de manutenção que fizeram parte do experimento, com a respectiva definição de classes de serviço e prioridade.

Tabela 4.2: Lista de solicitações de manutenção

Item	**Descrição**	**Classe de Serviço**	**Prioridade (para classe padrão)**
1	Ao realizar suprimento o sistema está dando uma mensagem de erro não compreensível pelo usuário e a operação não é executada	Expedição	-
2	Implementar a geração de informações para o SINTEGRA (requisito da legislação fiscal)	Expedição	-
3	O sistema não está exibindo o somatório correto do cupom na tela fechamento da venda	Expedição	-
4	Criar opção para captura de uma venda a partir de um pedido digitado, gerar e imprimir o cupom fiscal	Padrão	Alta
5	Na tela de fechamento da venda, criar opção para poder dar um desconto percentual. É necessária autorização do gerente para efetivar o desconto. O valor do desconto deve ser exibido na tela	Padrão	Alta
6	Quando o sistema cair por falta de energia ou travamento da máquina durante uma venda, ao ser acessado novamente deve voltar ao estado exato de venda em que se encontrava antes do problema	Padrão	Alta
7	Colocar na venda a opção para solicitação do nome e cpf do cliente que está comprando e imprimir os dados do cliente no cupom fiscal	Padrão	Baixa
8	Alterar o cancelamento de item para que o item cancelado seja destacado em uma cor diferente	Padrão	Baixa
9	Mecanismo para solicitar a autorização do gerente antes do cancelamento de um item ou cupom fiscal	Padrão	Baixa
10	Passar a exibir na tela a quantidade de volumes que compõem a venda	Padrão	Baixa
11	Pedir as informações detalhadas do cheque quando esta for a forma de pagamento escolhida	Padrão	Baixa
12	Exibir mensagem alertando para a necessidade de realizar retirada quando o valor acumulado no caixa superar R$ 300	Padrão	Baixa

Item	Descrição	Classe de Serviço	Prioridade (para classe padrão)
13	Criar tela para consulta ao estoque dos produtos	Padrão	Baixa
14	Durante uma venda aberta, o sistema está permitindo que o usuário saia apenas pressionando ESC. Esse comportamento não deve ser permitido.	Padrão	Baixa
15	Permitir a forma de pagamento “promissória”	Padrão	Média
16	Relatório gerencial que mostre as vendas por cada forma de pagamento em um período	Padrão	Média
17	Interface com a balança, para que o sistema capture o peso de um produto no campo quantidade	Padrão	Média
18	Gerar arquivo texto com informações sobre o estoque	Padrão	Média
19	Implementar tratamento para o caso da impressora fiscal ser desligada durante a venda	Padrão	Média
20	Criar opção para pagamento de promissória, com emissão de comprovante de pagamento	Padrão	Média
21	Criar rotina para atualização do estoque da frente de loja com base no banco de dados do servidor	Padrão	Alta
22	Criar rotina para enviar informações sobre os cupons fiscais para o banco de dados do servidor	Padrão	Alta
23	Implementar opção para registro de carga de créditos em telefone celular, com impressão de comprovante	Padrão	Alta
24	Ao fechar o dia, imprimir relatório gerencial com valores das vendas por forma de pagamento, nome do caixa e local para assinatura do gerente e do caixa	Padrão	Baixa
25	Quando a impressora fiscal estiver desligada, o sistema deve deixar habilitados apenas os opções de consulta	Padrão	Baixa

4.3. Dinâmica de Execução

4.3.1 Classes de Serviço

Apenas as classes de serviço “Padrão”, aplicável a todos os itens de trabalho, e “Expedição”, para itens de alta prioridade, foram utilizadas. A tabela 4.3 mostra a definição dessas classes:

Tabela 4.3: Definição das classes de serviço

Classe	Tipos de Itens de Trabalho	Tratamento Especial
Padrão	Todos	Ao ser selecionado pela equipe para atendimento dentro dos limites de trabalho em progresso, deve ser observada a ordem de prioridade: alta, média ou baixa, respectivamente.
Expedição	*Bugs* que impedem o trabalho em ambiente produtivo Demandas originadas de exigências legais	Permite quebra dos limites do trabalho em progresso; Pode parar outras atividades em andamento; Tem prioridade sobre todos os outros itens de trabalho; Só pode existir um item desse tipo de cada vez.

A tabela 4.2 já contém a definição da classe de serviço de cada um dos itens de solicitação de manutenção

4.3.2 Atividades Concluídas e Dia de Trabalho

A execução do experimento foi feita com base em uma simulação estocástica (SANTOS, 1999) de andamento das atividades através da cadeia de valor de manutenção. Foi utilizado o lançamento de um dado convencional de seis faces para definição do que se consideraria como uma atividade concluída.

Para inferir o que foi considerado como dia de trabalho, foi adotada a seguinte regra: todos os participantes do time lançavam o dado. Após o lançamento do dado pelo último componente do time, era considerado transcorrido um dia de trabalho.

4.3.3 Reuniões de Priorização e Entrada de Itens no Sistema

Para simulação das reuniões de priorização, foi adotado um sorteio dos itens da lista de solicitações de manutenção, considerando apenas aqueles que não faziam parte da classe de expedição.

Para simulação da entrada de itens da classe de expedição no sistema, foi utilizado o lançamento de dado, em um determinado momento do roteiro, conforme mostrado na Figura 4.1.

4.3.4 Fluxo de Execução

Com a finalidade de sistematizar e delimitar a execução do experimento, foi desenhado um fluxograma com o detalhamento de todos os passos a serem seguidos. Este fluxograma está mostrado na Figura 4.1. Foi montada a cadeia de valor a ser utilizada no começo do experimento, com as respectivas definições de limites para o trabalho em progresso.

A semântica para uma consideração do que seria uma tarefa concluída foi tomada como a obtenção de um número ímpar no lançamento, resultando em probabilidade de 50% de uma tarefa ser concluída.

Conforme mostrado na Figura 4.1, para entrada de uma demanda de expedição no sistema, o lançamento do dado foi realizado em um determinado passo do fluxo e, considerando os limites do *WIP* para esta classe de serviço e a obtenção do número 6 como resultado, um item desta classe entrava em atendimento.

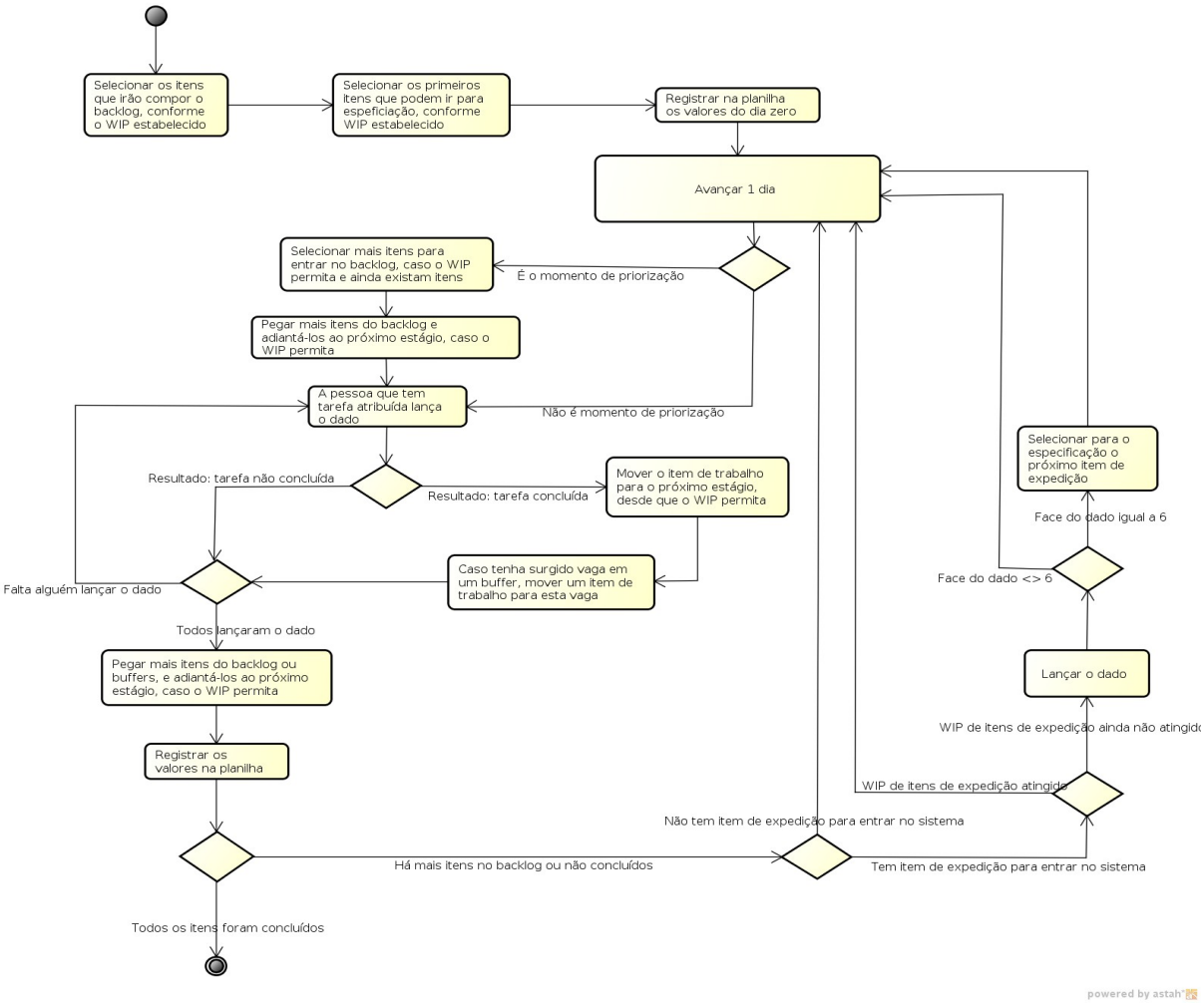


Figura 4.1: Fluxo de Execução do Experimento

4.4. Descrição da Execução

4.4.1 Seleção dos Participantes

A execução foi realizada com um time de 5 pessoas. Todos os participantes têm graduação em Ciência da Computação ou Engenharia Elétrica. Quatro deles detém títulos de mestre.

Foram selecionados por sua larga experiência em desenvolvimento de software. Dois participantes tem atuado há alguns anos como líderes de projeto, conduzindo, durante a maior parte do tempo, projetos de manutenção de software. Os demais tem experiência no exercício de papéis como desenvolvedores, analistas de requisitos, gestores de configuração, projetistas de testes e arquitetos de software.

4.4.2 Atividades Preliminares

Antes de iniciar a execução, os participantes tiveram uma orientação teórica a respeito de *Lean e* seus princípios, sistemas puxados, além de *Kanban* e suas principais práticas. Também lhes foram explicadas as premissas consideradas no início do experimento.

4.4.3 Cadeia de Valor e Limites para o Trabalho em Progresso

Para minimizar uma possível discussão sobre qual cadeia de valor e limites para o trabalho em progresso a serem adotados no início do experimento, foi estabelecida a cadeia de valor mostrada na Figura 4.2, com respectivos limites (mostrados entre parênteses abaixo do título de cada estágio).

CADEIA DE VALOR 1

	Backlog (5)	Especificação (2)	Implementação (2)	Testes integrados(1)	Pronto para implantação
Classe padrão					
Classe de expedição (WIP = 1)					

Figura 4.2: Cadeia de valor 1

Seguindo a política estabelecida para as classes de serviço, a classe de expedição permitia apenas um item de cada vez e por isso há a anotação “(*WIP*=1)”.

O time de cinco pessoas foi distribuído em seus papéis, a saber: dois especificadores, dois implementadores e um testador.

4.4.4 Execução Propriamente Dita

O time iniciou a execução do fluxo com o sorteio dos primeiros cinco itens para o *backlog*, seguindo o limite de trabalho em progresso estabelecido, passando a seguir os demais passos.

Figura 4.3: Equipe praticando a execução, através do lançamento de dados

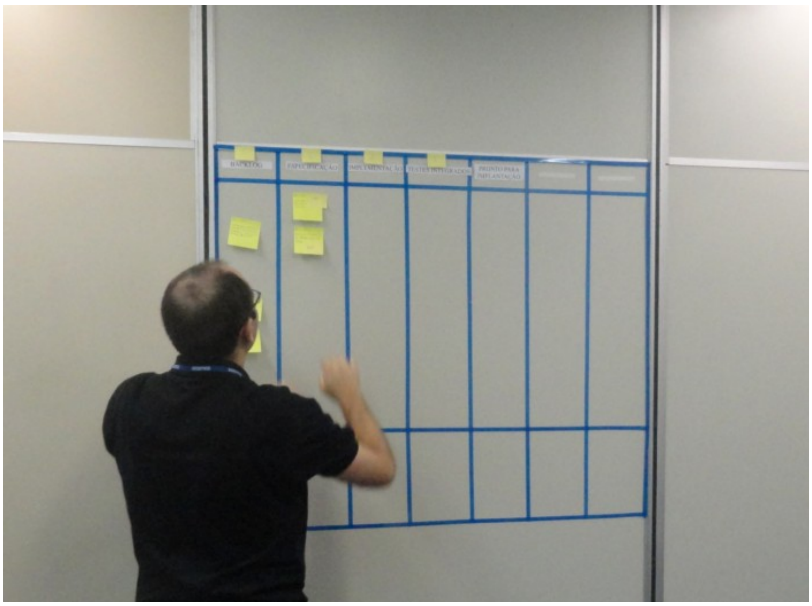


Figura 4.4: Membro do time selecionando item de trabalho para especificação

Houve um período de adaptação ao processo de execução e, logo no dia 3, ocorreu um debate na equipe sobre quem deveria ser a próxima pessoa a pegar uma atividade para executar, em virtude da conclusão de algumas atividades dos estágios iniciais.

Então, com base nessa discussão, a equipe sentiu a necessidade de fazer a primeira adaptação, colocando colunas de pronto para os estágios de especificação e implementação. Com isso, a cadeia de valor foi montada no quadro conforme a Figura 4.5.

CADEIA DE VALOR 2

	Backlog (5)	Especificação (2)	Especificação Pronta (2)	Implementação (2)	Implementação Concluída (2)	Testes integrados(1)	Pronto para implantação
Classe padrão							
Classe de expedição (WIP = 1)							

Figura 4.5: Cadeia de valor 2

A despeito dessa primeira adaptação, a equipe começou a perceber momentos de ociosidade de algumas pessoas.

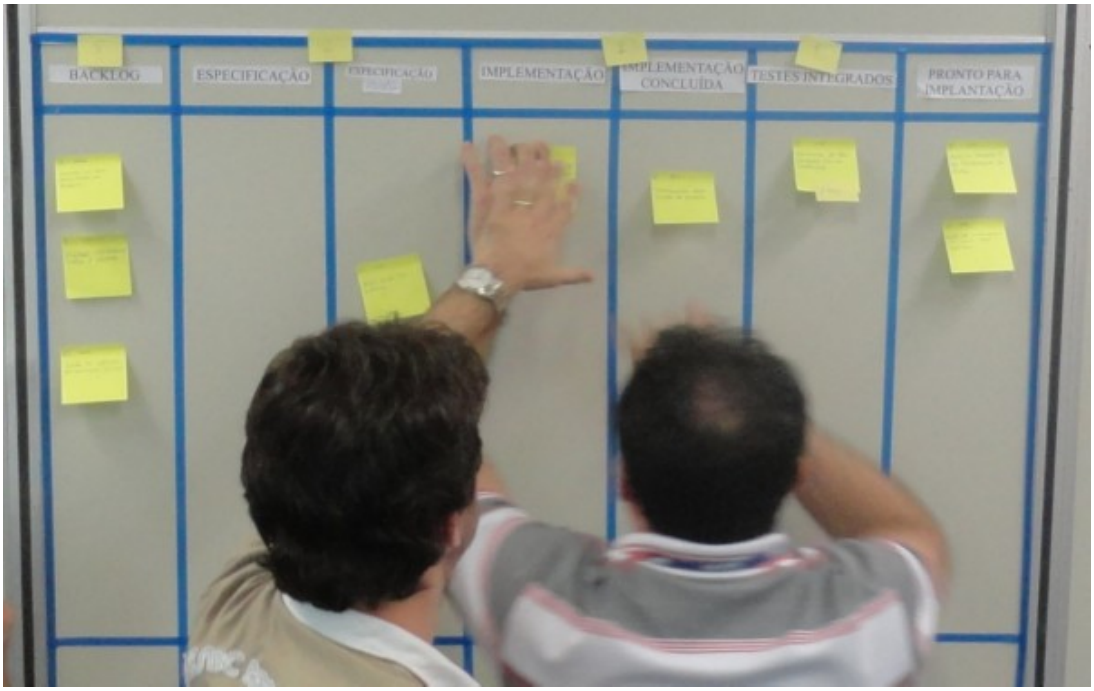


Figura 4.6: Equipe adaptando a cadeira de valor

A execução prosseguiu e, no dia 12, foi feita uma nova adaptação na cadeia de valor, conforme a Figura 4.7.

CADEIA DE VALOR 3

	5	**2**		**4**		**1**	
	Backlog	Especificação	Especificação Pronta	Implementação	Implementação Concluída	Testes integrados	Pronto para implantação
Classe padrão							
Classe de expedição (WIP = 1)							

Figura 4.7: Cadeia de valor 3

Para “Especificação” foi adotado um limite de quatro atividades em progresso, considerando dentro desse limite as atividades que estivessem no estágio “Especificação Pronta”. Para “Implementação” o limite também foi de quatro atividades, com a mesma consideração para o estágio “Implementação Concluída”.

Após essa adaptação e com a continuidade de ociosidade de alguns componentes, e também a

percepção do gargalo nas atividades de testes, foi feita uma nova adaptação no dia 17, considerando um ajuste de papéis no time e estabelecimento de limites para o trabalho em progresso. Até então cada pessoa estava especializada em um único papel. A partir desse dia, foi considerado que os implementadores também atuariam como testadores e assim o limite para "Testes Integrados" foi estabelecido em dois, conforme Figura 4.8.

CADEIA DE VALOR 4

	5	**2**		**4**		**2**	
	Backlog	Especificação	Especificação Pronta	Implementação	Implementação Concluída	Testes integrados	Pronto para implantação
Classe padrão							
Classe de expedição (WIP = 1)							

Figura 4.8: Cadeia de valor 4

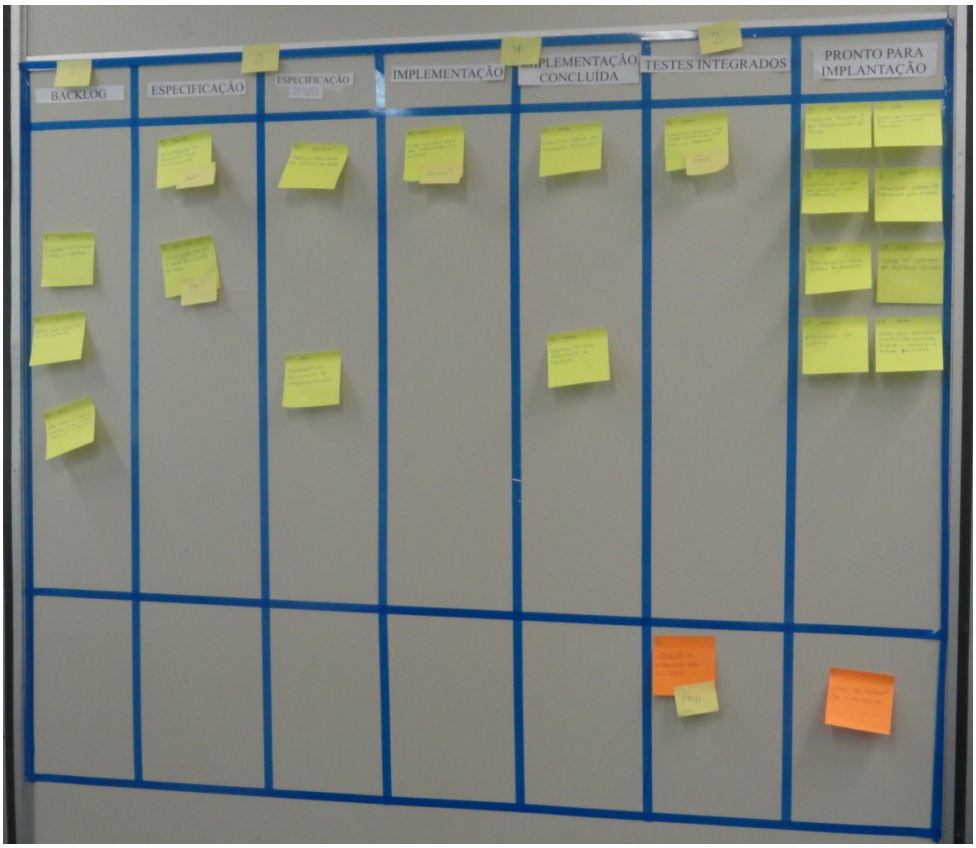


Figura 4.9:Quadro com cadeia de valor 4 mapeada e atividades em andamento

Após este último ajuste, a execução prosseguiu até a finalização de todos os 25 itens de trabalho, perfazendo um total de 39 dias de execução.

Vale relatar que os três itens que pertenciam à classe de expedição foram atendidos durante a execução do experimento, tendo sua entrada no sistema ocorrido de acordo com o critério aleatório definido no fluxograma.

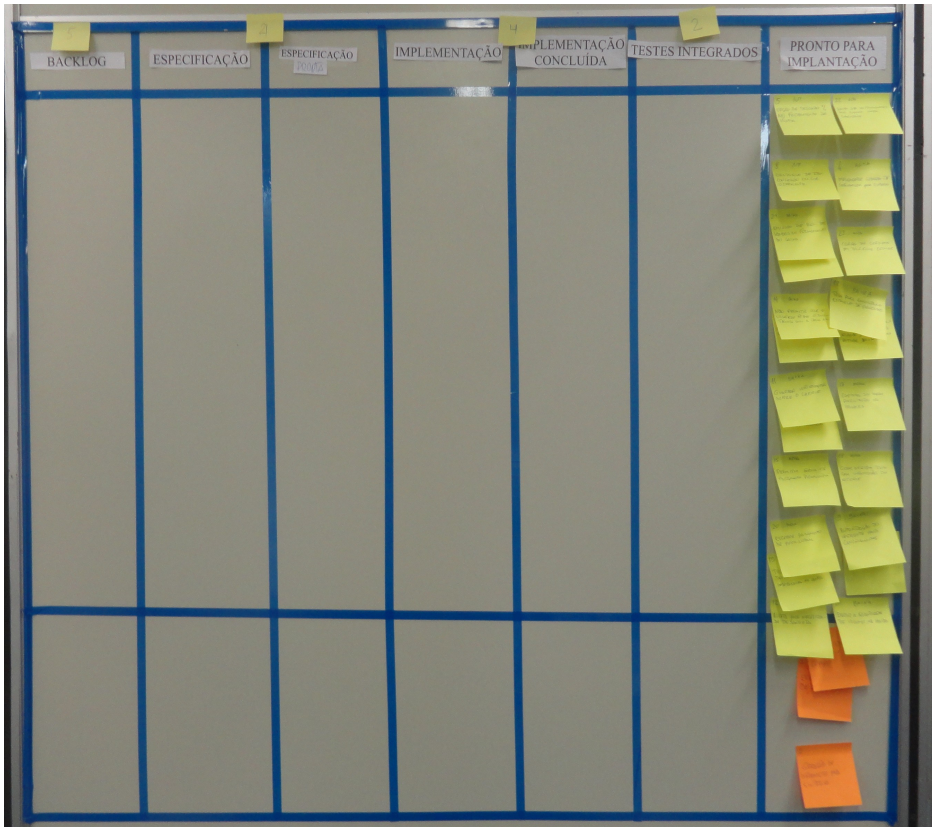


Figura 4.10: Quadro após finalização de todos os itens de trabalho

Outro fato que chamou a atenção foi o interesse e empenho do participantes na execução da simulação. Consideraram a forma de simulação lúdica e interessante, ensejando discussões sobre ociosidade, andamento de tarefas, coleta de informações e sobre as próprias técnicas de *Kanban*.

Um dos participantes questionou sobre o fato de que o percorrimento feito pelos itens de trabalho ao longo da cadeia de valor sugeria uma execução em cascata. Porém, após debate, ficou claro que as entregas rápidas permitiriam *feedback* imediato e adaptações em pouco tempo, minimizando os problemas de grandes alterações no futuro.

4.4.5 Métricas

Durante o experimento foi feita a coleta dos dados sobre o andamento das atividades. A Figura 4.11 mostra o diagrama de fluxo cumulativo do sistema. Nela é possível notar que não houve um fluxo contínuo de atividades entre os estágios da cadeia de valor, principalmente nos primeiros dias.

Pode-se observar, por exemplo, que no dia 2 não havia mais nenhum item no *backlog*. Isso poderia ser interpretado como uma indicação de que o limite de 5 itens deveria ser aumentado.

No dia 8, a despeito de haver item no *backlog*, não há atividade de especificação em andamento. Nesse caso, é possível verificar que há uma ociosidade porque o limite de trabalho em progresso para o próximo estágio já foi atingido, impedindo o prosseguimento da atividade no fluxo.

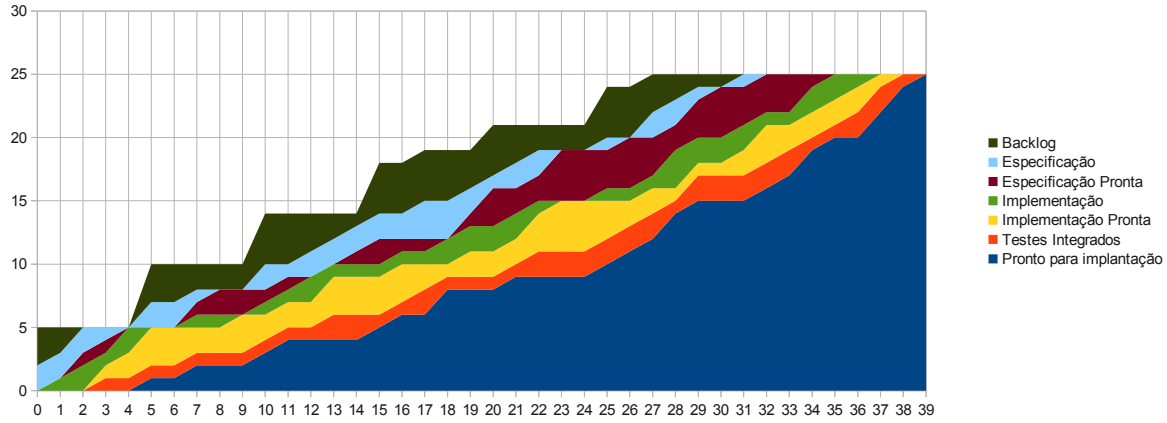


Figura 4.11: Diagrama de fluxo cumulativo

A Figura 4.12 mostra o comportamento do *lead time* médio. Nos dias iniciais do experimento, o *lead time* cresceu, mas, à medida em que o tempo passava, assumia uma tendência, permitindo o estabelecimento de limites inferior e superior.

Com esses limites seria possível definir acordos de nível de serviço garantindo, por exemplo, que 90% dos itens de trabalho seriam entregues em 16 dias.

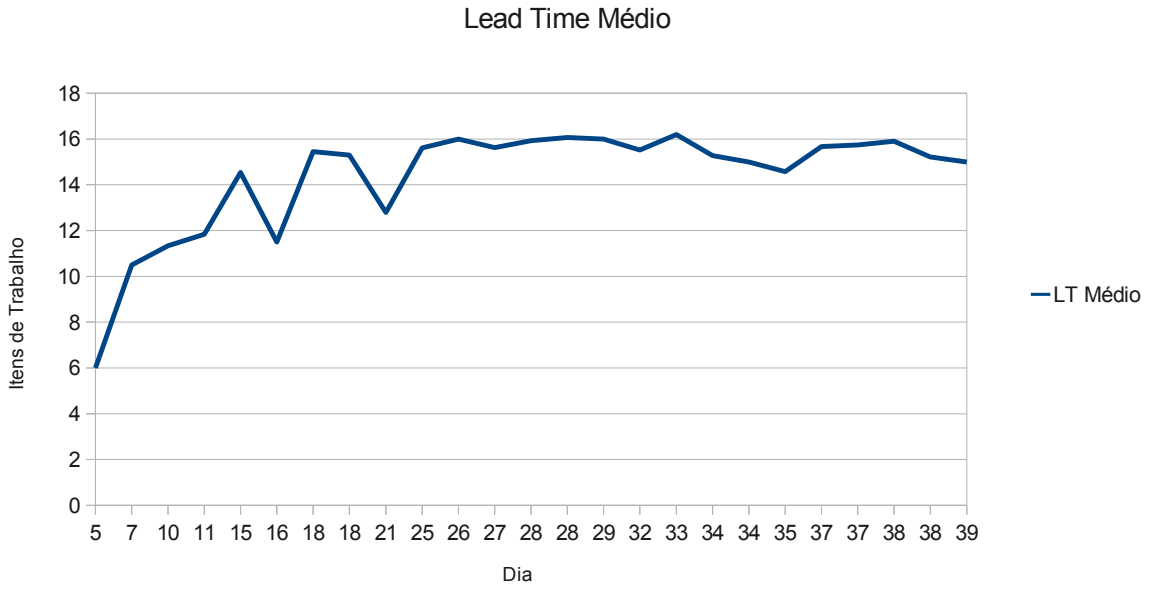


Figura 4.12: *Lead time* médio

Os *lead times* para os itens da classe de expedição foram, respectivamente, 7, 9 e 8 dias, levando a uma média de 8 dias.

Outra informação obtida como resultado do experimento foi a variação da ociosidade dos componentes do time ao longo do tempo, mostrada na Figura 4.13. Embora essa não tenha sido uma métrica citada no referencial teórico, ela demonstrou ser importante para o time e para orientar a tomada de algumas decisões. Um exemplo disso ocorreu quando implementadores também passaram a atuar como testadores, decisão que foi tomada a partir da análise da ociosidade e do

gargalo existente em “Testes Integrados”.

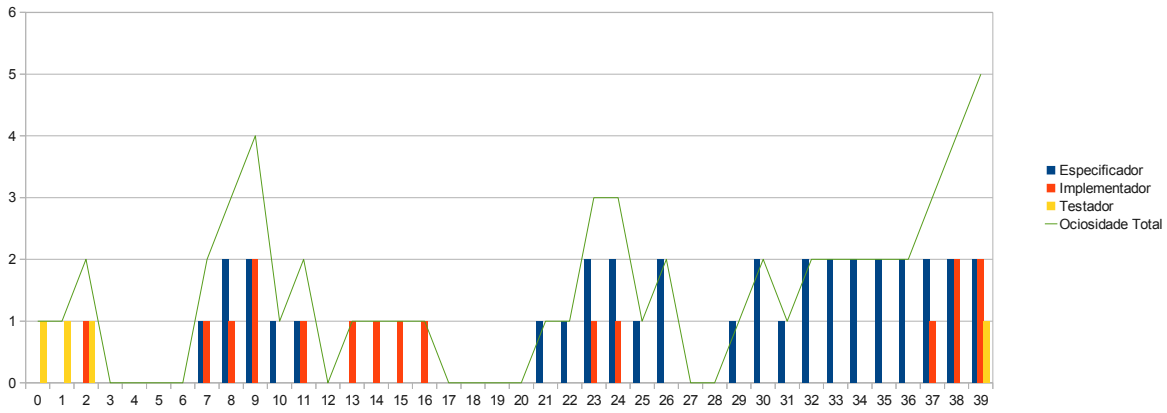


Figura 4.13: Ociosidade da equipe

No dia 9 é possível observar que houve um pico, com 4 pessoas sem atividades. Como todos os 25 itens de trabalho propostos foram tratados no experimento, temos um aumento natural de ociosidade a partir do dia 36, na medida em que a equipe começou a ser desalocada.

4.5. Análise dos Resultados

Ao final da execução, todos os participantes responderam ao questionário constante do Anexo A. O objetivo do questionário era obter a percepção do time sobre a aplicabilidade das práticas em projetos de manutenção.

Os participantes foram questionados a respeito do volume de atividades de manutenção de software conduzidas em suas equipes. Conforme se verifica na Figura 4.14, 60% informaram que a grande maioria das atividades de suas equipes refere-se à manutenção de software, mostrando que os respondentes tem vivência em projetos desse tipo.

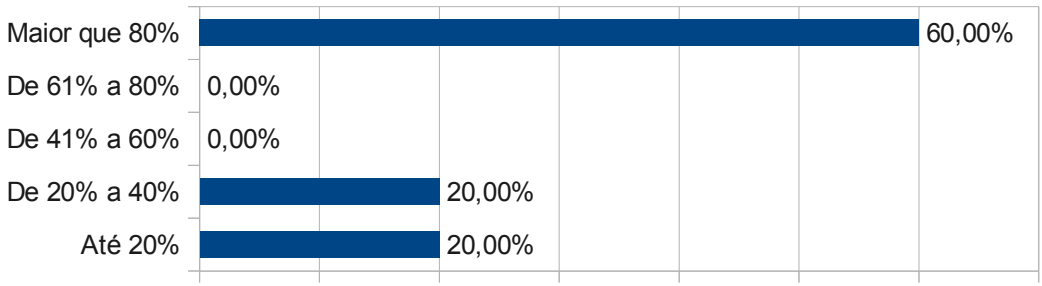


Figura 4.14: Qual o volume de projetos de manutenção de software conduzidos na sua equipe?

A primeira assertiva afirmava: “**A utilização de um sistema Kanban melhora o controle das tarefas**”. Analisando as respostas, mostradas na Figura 4.15, verifica-se que todos os participantes concordam total ou parcialmente com a afirmação.

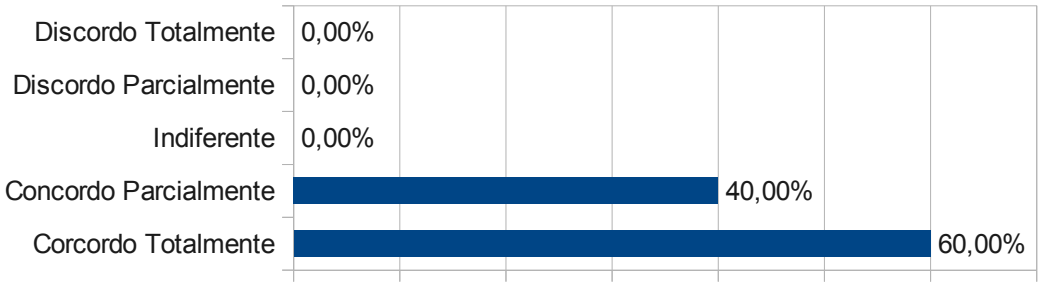
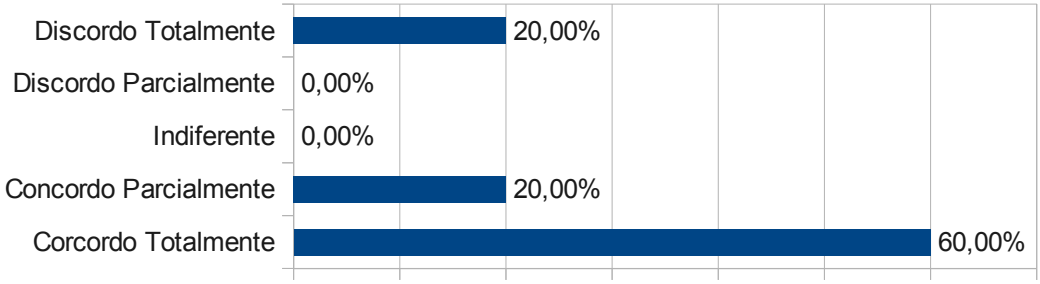


Figura 4.15: A utilização de um sistema *Kanban* melhora o controle das tarefas

A segunda assertiva afirmava: "**A utilização de um quadro *kanban* para visualização das tarefas e o fluxo segundo o qual a cadeia de valor é percorrida melhora a comunicação com os *stakeholders***". A Figura 4.16 mostra que 20% dos participantes discordam da afirmação. Entretanto, a maioria (80%), concorda total ou parcialmente com a afirmação.

Figura 4.16: A utilização de um quadro *kanban* para visualização das tarefas e o fluxo segundo o qual a cadeia de valor é percorrida melhora a comunicação com os stakeholders



Prosseguindo, foi feita afirmação a respeito da limitação do *WIP*: "**Limitação do trabalho em progresso é uma prática que contribui para redução na sobrecarga de tarefas dos componentes da equipe**". A Figura 4.17 mostra que não houve concordância total dos participantes, mas 100% deles responderam que concordavam parcialmente.

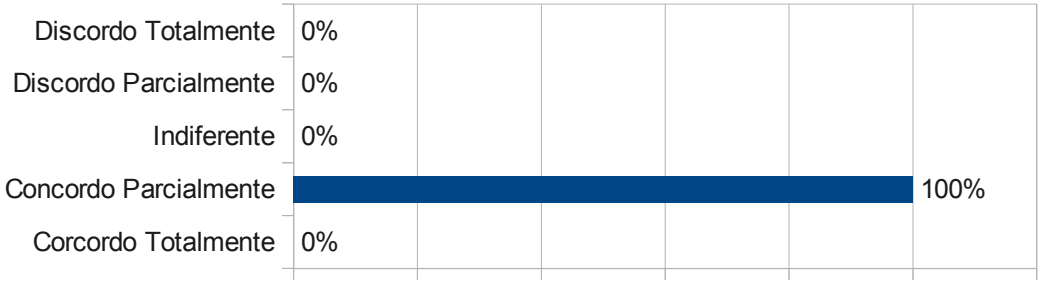


Figura 4.17: Limitação do trabalho em progresso é uma prática que contribui para redução na sobrecarga de tarefas dos componentes da equipe

A afirmação seguinte dizia: "**Ajuste de cadências e priorização, combinados, podem**

contribuir para atenuar as expectativas dos *stakeholders* e reduzir o conflito entre os demandantes de manutenções". A maioria dos participantes concordou total ou parcialmente com a assertiva. Na Figura 4.18 percebe-se que 80% deles concordam totalmente.

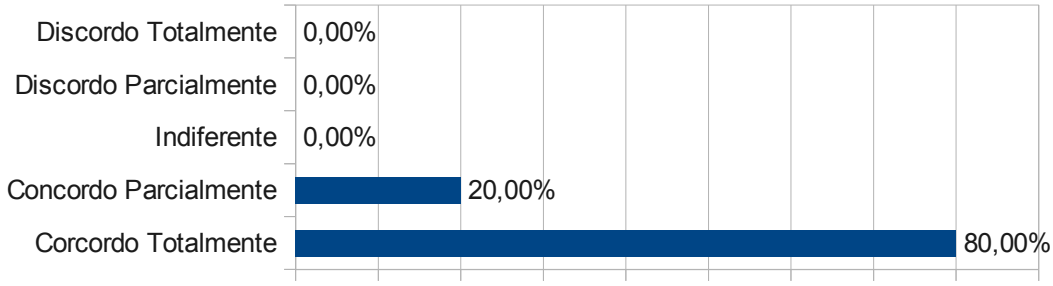
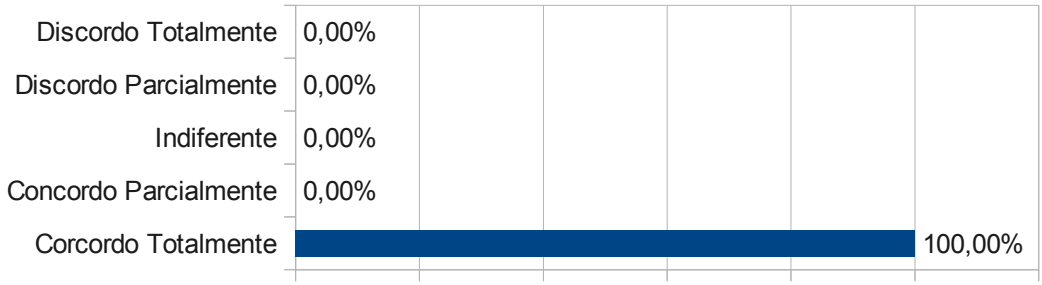


Figura 4.18: Ajuste de cadências e priorização, combinados, podem contribuir para atenuar as expectativas dos *stakeholders* e reduzir o conflito entre os demandantes de manutenções

A próxima afirmação referia-se à utilização de classes de serviço: "**Classes de serviço devem ser sempre utilizadas em um sistema *Kanban*, pois vão atenuar as expectativas dos stakeholders e vão facilitar as negociações de priorização**". A Figura 4.19 mostra que todos os participantes concordam totalmente com esta assertiva.

Figura 4.19: Classes de serviço devem ser sempre utilizadas em um sistema *Kanban*, pois vão atenuar as expectativas dos *stakeholders* e vão facilitar as negociações de priorização



Continuando, o questionário fazia uma negativa: "**Gerenciamento do fluxo de trabalho e medições não tem influência no nível de previsibilidade das entregas da equipe**", com a qual 100% dos participantes discordaram, conforme exibição na Figura 4.20.

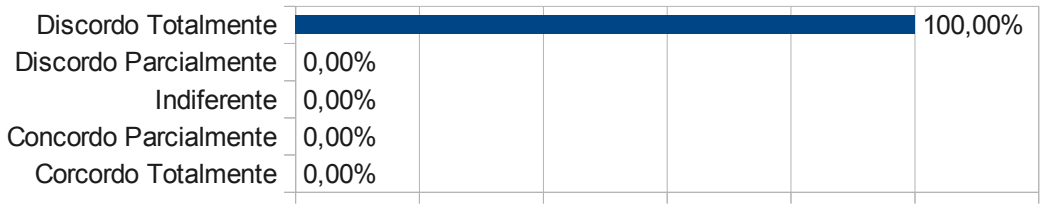


Figura 4.20: Gerenciamento do fluxo de trabalho e medições não tem influência no nível de previsibilidade das entregas da equipe

Finalizando a assertivas, foi afirmado: "**A utilização de um sistema *Kanban* é aplicável a projetos de manutenção de software**". Na Figura 4.21 é possível verificar que 100% dos participantes concordam que um sistema *Kanban* é aplicável a projetos de manutenção.

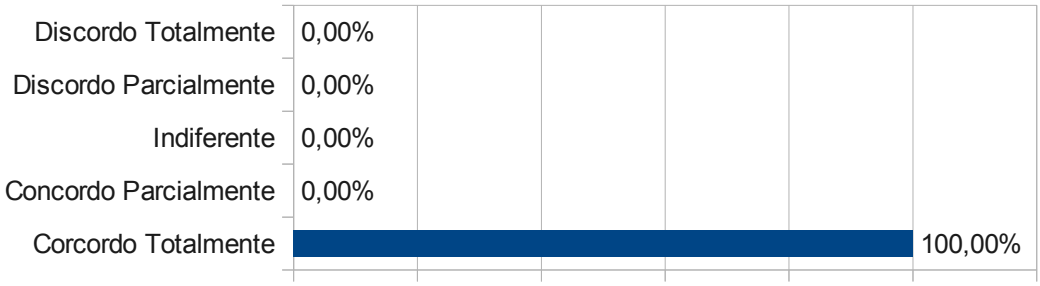


Figura 4.21: A utilização de um sistema Kanban é aplicável a projetos de manutenção de software

Os participantes tiveram ainda a possibilidade de fazer uma avaliação em texto livre das vantagens e desvantagens da aplicação da técnica, através da pergunta: "**Quais as vantagens e desvantagens da aplicação das técnicas utilizadas no experimento no contexto da sua equipe?**".

Algumas das vantagens citadas foram:

> *Acompanhamento efetivo do fluxo de trabalho; clareza das regras empregadas para distribuição do trabalho com limites de tarefas por fase; melhor controle de crescimento do backlog, permitindo que a equipe se organize e chegue a um ritmo sustentável; existe fluxo especial para demandas prioritárias.*
>
> *Maior controle e maior visibilidade das tarefas, além de maior previsibilidade das entregas.*
>
> *Melhor distribuição das tarefas, melhor controle das atividades (etapas de teste) concluídas, melhor visibilidade da alocação da equipe, facilidade de visualização do fluxo de cada projeto de teste.*
>
> *Definição das regras para o controle das solicitações. Maior probabilidade de se obter uma base histórica mais confiável. Alocação do analista em vários papéis. Maior visibilidade do projeto.*
>
> *Melhor visibilidade do andamento do projeto. A utilização da cadência e priorização são importantes para diminuir sobrecarga e expectativas.*

Algumas das desvantagens citadas foram:

> *Momentos de ociosidade de alguns membros da equipe. Contudo, essa ociosidade foi importante para mudanças nos limites das tarefas e papel de membros da equipe, reforçando o papel de testador.*
>
> *Necessidade da equipe conhecer a técnica, além da necessidade de conseguir adaptar à realidade da equipe, que nem sempre é trivial.*

> *Como trabalhamos com vários clientes simultâneos (~ 20 equipes de desenvolvimento) , não consegui visualizar a forma como iríamos priorizar as demandas no backlog.*
>
> *Limites que podem causar ociosidade.*

Analisando os resultados quantitativos e também vantagens e desvantagens citadas textualmente, é possível concluir que a utilização de um sistema *Kanban* tem aplicação positiva em projetos de manutenção, com efetividade no ataque da maior parte dos problemas relacionados.

A existência de ociosidade foi a desvantagem mais citada pelos participantes, mas uma dessas citações refere a derivação de ação de mudança de limites para o trabalho em progresso, mostrando que inspeção e adaptação poderiam contribuir para mitigação desse problema.

5. CONCLUSÃO

5.1. Considerações Gerais

Esse trabalho buscou avaliar a utilização das técnicas *Lean* e *Kanban* para atividades de manutenção de software. Foi feito estudo compreendendo manutenção de software e seus problemas. Também foram objeto de estudo os princípios e práticas das técnicas. Foi realizado um experimento prático para avaliar a aplicabilidade das práticas na resolução dos problemas.

A hipótese considerada era de que a utilização de *Lean* e *Kanban* traria benefícios para atividades de manutenção de software, contribuindo para solução dos problemas apresentados. No conexto de simulação considerado, foi possível verificar a hipótese, na medida em que uma equipe experiente pode utilizar as principais práticas, examinando sua utilização em um cenário experimental de manutenção e respondendo a um questionário de avaliação do uso das práticas sugeridas.

Este experimento foi conduzido através de uma simulação estocástica do atendimento de um *backlog* de demandas de manutenção, utilizando o resultado obtido com o lançamento de um dado como consideração de que uma determinada atividade tinha sido ou não concluída.

Inicialmente, a proposta era a realização de uma execução das técnicas em um projeto de manutenção real, mas não houve condição operacional para alcançar esta meta. Todavia, o experimento simulado se mostrou eficaz como instrumento de pesquisa e permitiu a geração dos subsídios para as conclusões deste trabalho.

A realização da simulação agregou várias possibilidades ao estudo, permitindo execução em tempo consideravelmente inferior ao que seria necessário em um projeto real. Além disso, também deu flexibilidade de testar inspeções e adaptações que poderiam ser inviáveis em projeto real inserido em um contexto organizacional sobre o qual o pesquisador não tinha gestão.

Poderiam ter sido realizadas mais execuções do experimento, porém não foi possível mobilizar um segundo grupo de pessoas com disponibilidade para participação.

Conforme os resultados obtidos na análise do experimento, pode-se concluir que *Lean* e *Kanban* são adequados para projetos de manutenção de software, atuando na mitigação dos principais problemas apresentados. Todos os participantes do experiemento concordaram totalmente com a afirmação de que *Kanban* é aplicável a projetos de manutenção de software. As demais respostas ao questionário aplicado também convergiram para uma concordância de que as práticas utilizadas no experimento resolvem os problemas que foram propostos neste trabalho.

Os participantes citaram as questões referentes à ociosidade como fator negativo na aplicação das técnicas. Contudo, considera-se que este ponto não inviabiliza a conclusão da aplicabilidade, pois novos ciclos de inspeção e adaptação poderiam ser conduzidos para mitigação deste problema.

5.2. Contribuições

As contribuições deste trabalho consistem nos seguintes pontos:

- Relacionamento de problemas de projetos de manutenção de software com práticas de *Lean* e *Kanban* a serem utilizadas para sua solução, com exemplos de quadros, cartões de trabalho, classes de serviço, métricas e gráficos. Foi demonstrado que o conjunto de práticas sugeridas tem atuação efetiva sobre os problemas propostos;
- Técnica de simulação utilizada, que se mostrou adequada à experimentação de práticas de engenharia de software em um curto espaço de tempo e pode ser utilizada em outros trabalhos e também para treinamentos.

5.3. Trabalhos Futuros

A condução e os resultados desta monografia sugerem alguns trabalhos futuros, a saber:

- Utilizar o mesmo mecanismo de simulação para experimentação de outras práticas como, por exemplo, reuniões de diárias, cadência de saída (considerando simulação de implantações), classes de serviço para itens de trabalho com data fixa de entrega, regras mais rígidas de priorização entre itens de trabalho, bloqueio de itens de trabalho e assim por diante. Buscar também ajustes de *WIP* até conseguir fluxo contínuo de itens de trabalho na cadeia de valor;
- Extrapolar a técnica de simulação utilizada neste trabalho para experimentação científica de outros métodos de engenharia de software;
- Realizar estudo para entendimento sobre o que é uma manutenção de software no contexto de desenvolvimento ágil, vez que, conforme IEEE1219 (1998 apud SWEBOK, 2004) manutenção consiste na modificação de um produto de software após sua entrada em produção, mas, em métodos ágeis, a entrada em produção ocorrerá rapidamente e ainda existirá escopo a ser desenvolvido. Esse escopo é de manutenção?
- Executar a aplicação das técnicas em um projeto real, em médio prazo, coletando os resultados para nova avaliação;
- Construir software para realização automatizada da simulação estocástica conduzida no experimento feito neste trabalho.

6. BIBLIOGRAFIA

ANDERSON, David J. *Kanban. Mudança Evolucionária de Sucesso para Seu Negócio de Tecnologia*. Blue Hole Press. 2011.

BASSI, Dairton L., Filho. *Experiências Com Desenvolvimento Ágil*. Teste (Mestrado) – Instituto de Matemática e Estatística, Universidade de São Paulo. São Paulo, SP. 2008.

BOEG, J. *Priming Kanban. A 10 step guide do optimizing your software delivery system.* InfoQ. 2011.

FRANCO, Eduardo F. *Um Modelo de Gerenciamento de Projetos Baseado nas Metodologias Ágeis de Desenvolvimento de Software e nos Princípios da Produção Enxuta*. Tese (Mestrado) – Escola Politécnica da Universidade de São Paulo. São Paulo, SP. 2007.

GOMES, A. *Lean Software Development – Qualidade e Velocidade sem Desperdícios*. Revista Java Maganize, n. 81. 2010. Disponível em <http://www.devmedia.com.br/lean-software-development-java-magazine-81/17442>. Acesso em 04/05/2012.

HELLESØY, A. *Cumulative Flow Diagrams with Google Spreadsheets*. 2010. Disponível em <http://open.bekk.no/cumulative-flow-diagrams-with-google-spreadsheets>. Acesso em: 27/04/2012.

LEANWAY. *História do Lean System.* Disponível em <http://leanway.com.br/historia%20do%20lean%20system>. Acesso em 04/05/2012.

PADUELLI, Mateus M. *Manutenção de Software: problemas típicos e diretrizes para uma disciplina específica*. Tese (Mestrado) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos, SP, 2007.

PADUELLI, Mateus M.; SANCHES, R. *Problemas em manutenção de software: caracterização e evolução*, In: III Workshop de Manutenção de Software Moderna, Vila Velha, ES, Brasil, Maio. 2006.

PETERSEN, K. *Implementing Lean and Agile Software Development in Industry*. Tese (Doutorado) - Blekinge Institute of Technology, School of Computing, Suécia. 2010.

POPPENDIECK, M; POPPENDIECK, T. *Lean Software Development.An Agile Toolkit.* Addison Wesley. 2003.

PRESSMAN, R.; *Software Engineering: a practitioner's approach. 7th ed.* McGraw Hill. 2010.

SANTOS, Maurício P. *Introdução à Simulação Discreta*. Universidade Estadual do Rio de Janeiro. 1999. Disponível em <http://www.mpsantos.com.br/simul/arquivos/simul.pdf>. Acesso em 22/05/2012.

SWEBOK. *Software Engineering Body of Knowledge* – SWEBOK. Disponível em: http://www.swebok.org. 2004.

WANGENHEIM, Christiane, G. *Utilização do GQM no Desenvolvimento de Software*. Universidade do Vale do Rio dos Sinos. Rio Grande do Sul. 2000. Disponível em <http://www.cordeiro.pro.br/aulas/engenharia/qualidade/GQM.pdf>. Acesso em 24/05/2012.

ANEXO A - QUESTIONÁRIO APLICADO COM OS PARTICIPANTES DO EXPERIMENTO

Após a realização do experimento, o seguinte questionário foi aplicado com os participantes:

1. A utilização de um sistema *Kanban* é aplicável a projetos de manutenção de software.

( ) Discordo totalmente
( ) Discordo parcialmente
( ) Indiferente
( ) Concordo parcialmente
( ) Discordo totalmente

2. A utilização de um sistema *Kanban* melhora o controle das tarefas.

( ) Discordo totalmente
( ) Discordo parcialmente
( ) Indiferente
( ) Concordo parcialmente
( ) Discordo totalmente

3. A utilização de um quadro *kanban* para visualização das tarefas e o fluxo segundo o qual a cadeia de valor é percorrida melhora a comunicação com os *stakeholders*.

( ) Discordo totalmente
( ) Discordo parcialmente
( ) Indiferente
( ) Concordo parcialmente
( ) Discordo totalmente

4. Limitação do trabalho em progresso é uma prática que contribui para redução na sobrecarga de tarefas dos componentes da equipe.

( ) Discordo totalmente
( ) Discordo parcialmente
( ) Indiferente
( ) Concordo parcialmente
( ) Discordo totalmente

5. Ajuste de cadências e priorização, combinados podem contribuir para atenuar as expectativas dos *stakeholders* e reduzir o conflito entre os demandantes de manutenções.

( ) Discordo totalmente
( ) Discordo parcialmente
( ) Indiferente
( ) Concordo parcialmente
( ) Discordo totalmente

6. Classes de serviço devem ser sempre utilizadas em um sistema *Kanban*, pois vão atenuar as expectativas dos *stakeholders* e vão facilitar as negociações de priorização.

( ) Discordo totalmente
( ) Discordo parcialmente
( ) Indiferente
( ) Concordo parcialmente
( ) Discordo totalmente

7. Gerenciamento do fluxo de trabalho e medições não tem influência no nível de previsibilidade das entregas da equipe.

( ) Discordo totalmente
( ) Discordo parcialmente
( ) Indiferente
( ) Concordo parcialmente
( ) Discordo totalmente

8. Qual o volume de projetos de manutenção de software conduzidos na sua equipe?

( ) Até 20%
( ) De 20% a 40%
( ) De 41% a 60%
( ) De 61% a 80%
( ) Maior que 80%

9. Quais as vantagens e desvantagens da aplicação das técnicas utilizadas no experimento no contexto da sua equipe?