UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA DE PRODUÇÃO E SISTEMAS

PEDRO HENRIQUE DE ALENCAR MACHADO

REDUÇÃO DE DESPERDÍCIOS NO DESENVOLVIMENTO DE *SOFTWARE* DE GRANDE PORTE POR MEIO DE FERRAMENTAS LEAN

PATO BRANCO

2017

PEDRO HENRIQUE DE ALENCAR MACHADO

REDUÇÃO DE DESPERDÍCIOS NO DESENVOLVIMENTO DE *SOFTWARE* DE GRANDE PORTE POR MEIO DE FERRAMENTAS LEAN

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia de Produção e Sistemas – PPGEPS, da Universidade Tecnológica Federal do Paraná – UTFPR, Campus Pato Branco, como requisito parcial para obtenção do título de Mestre em Engenharia de Produção e Sistemas.

Orientador: Prof. Dr. Marcelo Gonçalves Trentin

PATO BRANCO

2017

M149r Machado, Pedro Henrique de Alencar.
Redução de desperdícios no desenvolvimento de software de grande porte por meio de ferramentas Lean / Pedro Henrique de Alencar Machado. -- 2017.
126 f. : il. ; 30 cm.

Orientador: Prof. Dr. Marcelo Gonçalves Trentin
Dissertação (Mestrado) - Universidade Tecnológica Federal do Paraná. Programa de Mestrado em Engenharia de Produção e Sistemas. Pato Branco, PR, 2017.
Bibliografia: f. 115 - 122.

1. Desenvolvimento de Software. 2.Lean. 3. Software de grande porte. 4. Lean *Software Development*. 5. Metodologias Ágeis. I. Trentin, Marcelo Gonçalves, orient. II. Universidade Tecnológica Federal do Paraná. Programa de Mestrado de Pós-Graduação em Engenharia de Produção e Sistemas. III. Título.

CDD (22. ed.) 670.42

Ficha Catalográfica elaborada por
Maria Juçara Vieira da Silveira CRB-9/1359
Biblioteca da UTFPR Campus Pato Branco

TERMO DE APROVAÇÃO DE DISSERTAÇÃO Nº 22

A Dissertação de Mestrado intitulada **"Redução de desperdícios no desenvolvimento de *software* de grande porte por meio de ferramentas *Lean*"**, defendida em sessão pública pelo candidato **Pedro Henrique de Alencar Machado**, no dia 04 de dezembro de 2017, foi julgada para a obtenção do título de Mestre em Engenharia de Produção e Sistemas, área de concentração Gestão dos Sistemas Produtivos, e aprovada em sua forma final, pelo Programa de Pós-Graduação em Engenharia de Produção e Sistemas.

BANCA EXAMINADORA:

Prof. Dr. Marcelo Gonçalves Trentin - Presidente – UTFPR

Prof. Dr. Guilherme Luz Tortorella – UFSC

Prof. Dr. Dalmarino Setti – UTFPR

A via original deste documento encontra-se arquivada na Secretaria do Programa, contendo a assinatura da Coordenação após a entrega da versão corrigida do trabalho.

Pato Branco, 04 de dezembro de 2017.

Prof. Dr. Fernando José Avancini Schenatto
Vice-Coordenador do PPGEPS

MACHADO, A. H. Pedro; **REDUÇÃO DE DESPERDÍCIOS NO DESENVOLVIMENTO DE *SOFTWARE* DE GRANDE PORTE POR MEIO DE FERRAMENTAS LEAN.** 2017. Dissertação de Mestrado em Engenharia de Produção e Sistemas – Universidade Tecnológica Federal do Paraná, Pato Branco, 2017.

RESUMO

O processo de desenvolvimento de sistemas de grande porte, envolvem particularidades que tornam a sua gestão mais complexa, quando comparada ao desenvolvimento de sistemas tradicionais. Estudos ilustram resultados favoráveis quando aplicado a filosofia ágil durante o processo de fabricação de um *software*, no entanto, desperdícios ainda são observados quando o contexto do desenvolvimento são aplicações de grande porte. Assim sendo, a pesquisa em questão procura a redução de desperdícios identificados durante o processo de desenvolvimento de sistemas de grande porte. Foram realizados estudos teóricos e práticos objetivando analisar as técnicas e ferramentas Lean para obtenção de resultados mais eficientes. O estudo apresenta e discute conceitos do Lean para desenvolvimento de *software* e a sua relação com as metodologias ágeis para desenvolvimento de sistemas de informação. Resultados apontam melhorias no processo de desenvolvimento de *software* além da redução de desperdícios com superprodução de funcionalidades e defeitos encontrados. A dissertação em questão será constituída de uma etapa de revisão da literatura, e duas próximas etapas com aplicações práticas de ferramentas e técnicas *Lean* para a redução de desperdício no processo de desenvolvimento de *software* de grande porte.

Palavras-chave: Desenvolvimento de *Software*. Lean. *Software* de grande porte. Lean *software Development*. Metodologias Ágeis.

MACHADO, A. H. Pedro; **REDUCTION OF WASTE IN THE LARGE SCALE SOFTWARE DEVELOPMENT WITH THE APPLICATION OF LEAN TOOLS**. 2017. Master's Thesis in Production and Systems Engineering - Federal Technological University of Parana, Pato Branco, 2016.

ABSTRACT

The development process of large systems involves particularities that make their management more complex when compared to the development of traditional systems. Studies illustrate favorable results when applied to agile philosophy during the manufacturing process of a software, however, wastes are still observed when the development context are large applications. Therefore, the research in question seeks the reduction of wastes identified during the development process of large systems. Theoretical and practical studies were carried out to analyze Lean techniques and tools to obtain more efficient results. The study presents and discusses Lean concepts for software development and its relationship with agile methodologies for the development of information systems. Results point to improvements in the software development process besides the reduction of waste with overproduction of functionalities and defects found. The dissertation in question consists of a literature review stage, and two next steps with practical applications of tools and Lean techniques to reduce waste in the large software development process.

Keywords: Software Development. Lean. large software. Lean software development. Agile methodologies.

SUMÁRIO

1 INTRODUÇÃO

Como resposta às ineficiências identificadas pelos modelos de processo de desenvolvimento de *software* tradicionais, em meados dos anos 2000 surgiram os modelos ágeis (AMs) (HIGHSMITH, 2002). De acordo com COOPER, 2006, AMs são processos da engenharia de *software* que compartilham dos mesmos objetivos: Indivíduos e interações antes de processos e ferramentas, *software* funcionando antes de documentação abrangente, colaboração com o cliente antes das negociações de contratos e rápida resposta às mudanças ao longo de todo ciclo de vida.

A partir da sua publicação por meio do manifesto ágil, diversas metodologias foram e ainda estão sendo concebidas, como por exemplo: *Extreme Programming* (BECK 1999), *Cristal* (COCKBURN 2001), *Scrum* (SCHWABER e BEEDLE 2002), *FDD* (COAD e PALMER 2002) entre outras. Práticas ágeis de desenvolvimento de sistemas de informação estão ganhando cada vez mais aceitação na prática (DINGSOYR et. al 2012). O objetivo principal da sua utilização é a disponibilidade que elas possuem em serem reativas às frequentes mudanças no projeto, contribuindo para os valores percebidos pelos clientes (custo, qualidade e simplicidade) (CONBOY 2009).

Embora bons resultados tenham sido observados a partir da aplicação de métodos ágeis para desenvolvimento de *software*, algumas abordagens dos referentes modelos têm sido cada vez mais questionadas. Este questionamento é levantado principalmente quando o assunto é, gerenciamento visual, métricas de equipe e produção, priorização de demandas de fabricação, entre outras características essenciais para o contexto de *softwares* de grande porte (BIFFL, et. al 2005, KUPIAINEN et. al 2014). Alguns desses problemas poderão ser resolvidos a partir da aplicação de práticas do desenvolvimento enxuto.

Empresas responsáveis pelo desenvolvimento de *softwares* tendem a procurar continuamente por melhores práticas para a entrega de seus sistemas em um tempo e custo cada vez mais competitivo sem comprometer, é claro, a qualidade do produto final. Além disso, grandes empresas desenvolvedoras de sistemas, buscam melhorar seus *softwares* com o propósito de atender um número maior de clientes, o que faz de seus sistemas, produtos cada vez mais

robustos e complexos de se manter, ou seja, *softwares* de grande porte. (NIVOIT, 2013)

Uma característica dos *softwares* de grande porte são as necessidades contínuas de atualização, melhorias e revisões procurando melhor atender às necessidades dos clientes e aumento da eficiência do programa como um todo. Estas melhorias são geralmente realizadas em partes do *software* (módulos) e denominadas de *features.*

O desenvolvimento enxuto (*Lean*), já alinhado a muitos dos princípios ágeis (melhoria contínua, redução de desperdício etc.), tem sido considerado como uma forma de ultrapassar as limitações dos processos ágeis para desenvolvimento de *softwares* de grande porte, conceito este conhecido como *Lean Software Development (LSD)*(POPPENDIECK e POPPENDIECK 2003; LARMAN e VODDE 2008; VILKKI 2010; LAANTI 2012). O interesse da indústria de *software* pela aplicação dos conceitos *Lean* cresceu significativamente nos últimos anos, e o LSD, que inicialmente era considerado como um dos métodos ágeis (DYBA et. al 2008), está adquirindo cada vez mais sua própria identidade (RODRIGUEZ 2013).

No entanto, o conceito em questão, também denotado como '*Leagile*' em referência à combinação de práticas ágeis com métodos *Lean*, ainda é algo não muito difundido, tanto na literatura quanto em sua aplicação prática (WANG 2012).

1.1. PROBLEMA DE PESQUISA

A maior parte dos projetos de *software* de grande porte são difíceis de se manter ao longo do tempo, isso devido à complexidade do negócio e a alta interdependência entre os fluxos de trabalho. Esses projetos exigem uma coordenação mais estreita, e também a refinamentos frequentes com o propósito de permanecer atendendo às necessidades de seus clientes. Essa gestão é principalmente aplicada em metodologias convencionais da Engenharia de Software, isso porque as práticas ágeis melhor se aplicam em projetos menores. (NIVOIT, 2013).

Considerando os ganhos em diferentes aspectos destacados a partir da utilização das metodologias ágeis, comprova-se a aplicação de cada um dos

seus conceitos, porém, particularidades identificadas em projetos de *software* de grande porte, destacadas por Kupiainem (2015), faz-se com que surja a principal questão desta pesquisa: Como a utilização das práticas *Lean* podem tornar o processo de desenvolvimento de software de grande porte mais eficiente?

1.2. OBJETIVOS

1.2.1. Objetivo Geral

Este estudo tem como objetivo a aplicação de ferramentas e práticas *Lean* com o propósito de redução de desperdício no processo de desenvolvimento de *software* de grande porte, visando melhorar a gestão de uma organização, considerando os quesitos necessários para sua operacionalização.

1.2.2. Objetivos Específicos

Contribuindo para a construção do estudo alguns objetivos específicos foram traçados. Destacam-se os seguintes:

- Identificar, no contexto de desenvolvimento de *software* de grande porte, quais são os principais problemas de processo, e de que forma a aplicação das ferramentas *Lean* podem minimizar a sua ocorrência;
- Caracterizar particularidades do desenvolvimento de *software* de grande porte, tais como gerenciamento visual, métricas de equipe e produção, priorização de demandas de fabricação;
- Priorizar demandas em uma fábrica de desenvolvimento de *software* de grande porte, reduzindo o desperdício com produção de funcionalidades não utilizadas;
- Melhorar o fluxo das atividades, priorizando demandas, e buscando uma maior estabilidade durante o desenvolvimento de softwares, reduzindo o desperdício com funcionalidades inacabadas e como consequência, propor uma forma de gerenciamento das atividades de desenvolvimento de um sistema de grande porte.

1.3. JUSTIFICATIVA

Mesmo após a crise do *software*, descrita por Pressman (2002) como o período, no final dos anos 70, em que a atividade de desenvolvimento de *software* era executada de forma desorganizada, desestruturada e sem planejamento, gerando um produto final de má qualidade, alguns dos problemas difundidos naquela época, mesmo após a estruturação e criação do processo de desenvolvimento de sistemas, perduram até hoje: projetos atrasados, erros de estimativa de custo e tempo, problemas para gerenciá-los, entre outros (DOMINGOS, et al., 2010). Além disso, a alta competitividade da indústria de desenvolvimento de software está fazendo com que a capacidade de desenvolver e entregar produtos de maneira mais eficiente, venha a ser um alvo cada vez mais desejado por qualquer empresa do setor (BIFFL, et al., 2005). Neste sentido, as metodologias ágeis de desenvolvimento de *software* baseiam-se em valores e princípios que tornam o processo mais rápido e adepto à mudanças e necessidades dos clientes (PETERSEN, 2010).

Mesmo a partir de melhores resultados obtidos com a utilização das práticas implementadas pelas metodologias ágeis de desenvolvimento de *software*, alguns aspectos são questionados quando as respectivas práticas são aplicadas à contextos distintos, como é o caso do domínio de desenvolvimento de *softwares* de grande porte. Essa esfera possui algumas particularidades específicas, como por exemplo: documentação de *software*, gerenciamento visual, métricas de equipe e produção, priorização de demandas de fabricação, entre outras (MISHRA, 2011; LADAS, 2012; KUPIAINEN, 2015).

Quando voltado à literatura, acerca de conteúdos dessas particularidades apresentadas no contexto de desenvolvimento de *software* de grande porte, pouco se têm encontrado. Dessa forma, têm-se a principal justificativa teórica do presente trabalho científico: Aplicação de ferramentas *Lean* para a redução de desperdício em fábricas de *software*s de grande porte.

Como justificativa prática do respectivo trabalho, têm-se: Aplicação de um estudo de caso em uma empresa desenvolvedora de software de grande porte, o qual irá ser objeto de análise e coleta de dados.

1.4. PROCEDIMENTOS METODOLÓGICOS E ESTRUTURA DO TRABALHO

O presente trabalho trata-se de uma pesquisa aplicada com caráter exploratório e descritivo.

Optou-se pela estruturação da dissertação em três etapas de pesquisa, as quais compõe a o encadeamento final (LUNDAHL, 2010).

Dentro deste conceito, as três etapas desenvolvidas compreendem procedimentos metodológicos específicos, sendo que o primeiro dos trabalhos é de caráter teórico e os demais de caráter aplicado.

A primeira etapa, apresenta características que combinam as abordagens qualitativa e quantitativa. Já as etapas aplicadas, caracterizadas pela utilização de estudos de caso, onde os testes tendem a responder anseios de abordagens qualitativas.

A Figura 1 destaca a estrutura produzida e principais resultados alcançados. A partir da produção da primeira etapa, foi identificado as principais lacunas da pesquisa, as quais foram insumos para as próximas produções. Como resultado da segunda etapa, foi realizado a priorização de demandas do desenvolvimento de sistemas de grande porte, a qual foi insumo para a gestão de atividades e criação de um fluxo contínuo de atividades demandadas durante o desenvolvimento de um sistema de grande porte, resultado principal da última etapa.

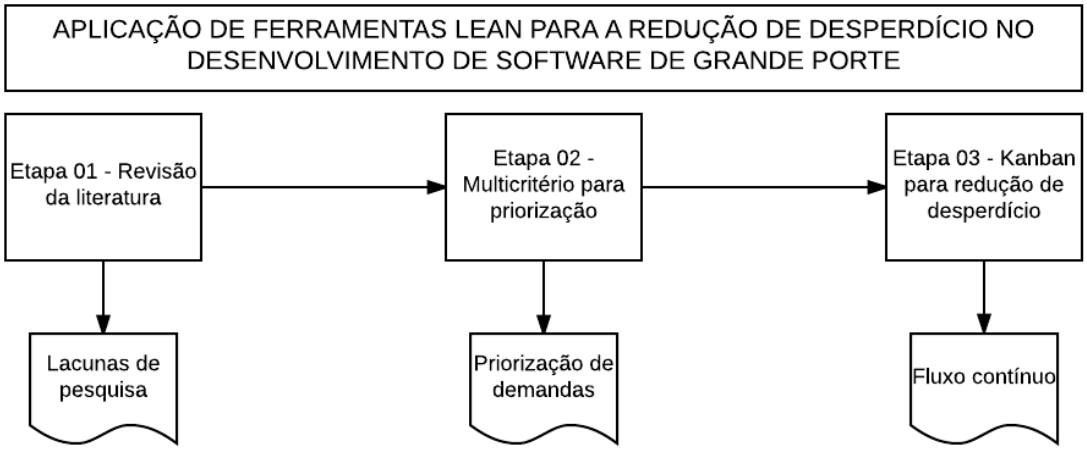

Figura 1 - Estrutura metodológica da dissertação. Fonte: Dados da pesquisa

1.4.1. Etapa 1: Revisão da literatura

Para a construção da etapa de revisão da literatura, foi adotada como ferramenta de pesquisa o *Proknow-C* (*Knowledge Development Process – Construtivist*). Esta técnica de estruturação de pesquisa bibliográfica, serviu para a seleção estruturada de um portfólio de artigos de destaque sobre o tema da pesquisa. Além disso, permitiu uma análise bibliométrica do portfólio, e, na sequência, a realização de uma análise sistêmica do tema em questão (MARAFON, ENSSLIN, et al., 2012). A análise sistêmica reflete à produção de conteúdo sobre as principais lacunas identificadas a partir da pesquisa bibliográfica.

1.4.2. Etapa 2: Priorização de demandas das atividades de desenvolvimento

Um dos problemas encontrados na literatura acerca do contexto de desenvolvimento de *software* de grande porte, é a priorização de demandas (BIFFL, et. al 2005, KUPIAINEN et. al 2014). Com relação às metodologias ágeis de desenvolvimento de *software*, nenhum processo estruturado é sugerido para a resolução da problemática em questão.

A principal finalidade da filosofia *Lean* é a redução de custo e eliminação de todas e quaisquer atividades que geram desperdício (LEAL 2003). Assim sendo, é essencial a realização de uma análise e observação de um processo produtivo a fim de identificar atividades que agregam ou não valor ao produto final. Com o propósito de priorizar as demandas para o desenvolvimento

de um sistema de grande porte, o próximo passo foi a priorização das funcionalidades solicitadas pelos clientes. Esta necessidade ocorre devido a elas possuírem características diferentes em termos de complexidades, acarretando tempos de produção diversos. Com base em um estudo preliminar sobre os métodos de análise multicritério, o método Electre II foi selecionado para a resolução da problemática previamente destacada.

Nessa etapa da pesquisa, foi realizado um estudo de caso em uma empresa desenvolvedora de *software* de grande porte, objetivando resolver a problemática de ordenação, classificação e ranqueamento de demandas à serem desenvolvidas, usando critérios que favoreçam o balanceamento da linha de produção de desenvolvimento.

1.4.3. Etapa 3: Redução de desperdício no processo de produção de *software* de grande porte

Na última etapa desse trabalho, a partir de uma demanda priorizada (obtida como resultado da etapa 2) para o desenvolvimento de sistemas de grande porte, foi proposto e concebido um quadro Kanban com o propósito de reduzir o desperdício com estoques além de criar um fluxo contínuo e balanceado. O objetivo do trabalho é produzir uma linha de produção puxada com um fluxo balanceado na execução de atividades inerentes ao desenvolvimento de software de grande porte.

Com a aplicação da técnica de Kanban, criou-se um fluxo de trabalho balanceado, além de outras características necessárias para o seu bom funcionamento. Como principal referência para a proposição do quadro Kanban, foi usado um estudo realizado por Ladas (2009) intitulado: *Scrumban: Essays on Kanban Systems for Lean software development*.

Nessa etapa, também foi realizado um estudo de caso em uma empresa desenvolvedora de *software* de grande porte, objetivando coletar dados para a concepção do quadro Kanban, além de colocá-lo em prática e visualizar os pontos fortes e fracos da respectiva proposição.

1.5. DELIMITAÇÃO

O presente estudo pretende realizou a aplicação de ferramentas *Lean* com o intuito de reduzir o desperdício e melhorar a gestão e resultados de uma fábrica desenvolvedora de *softwares* de grande porte. Para tanto, será realizado um estudo qualitativo e exploratório, objetivando a coleta de informações necessárias para a concepção do respectivo modelo.

Serão analisadas características de uma empresa do contexto de produção de *softwares* de grande porte, cujos dados coletados serão utilizados como base para a concepção do modelo. Além disso, as informações serão confrontadas com dados coletados com especialistas no contexto da pesquisa.

Assim sendo, a principal delimitação deste trabalho é o fato de utilizar de um estudo de caso único a ser realizado em uma empresa do sudoeste do estado do Paraná. A aplicação das ferramentas foi feita durante o processo de desenvolvimento de um mesmo produto de *software*, podendo ter variações em outros desenvolvimentos.

2 REFERENCIAL TEÓRICO

A seguir será apresentado as referências teóricas sobre as questões gerais da presente pesquisa. Assuntos como processos ágeis de desenvolvimento de *software*, sistemas de grande porte e a forma que a filosofia Lean pode auxiliar no contexto de desenvolvimento de *softwares* de grande porte, serão apresentadas nas próximas seções.

2.1.1 Lean Thinking

Na literatura atual, os conceitos “ágeis” e “*Lean*” são muito pouco difundidos, e a utilização desses termos é muitas vezes considerada ambígua e inconsistente (CONBOY, 2009). Alguns autores veem os dois termos como nomes diferentes para tratarem a mesma ocorrência, por exemplo, nos estudos de KETTUNEN 2008, CHOW 2008, nenhuma distinção significativa é feita entre as duas práticas. Entretanto, outros estudos, como por exemplo, WANG 2012, PERNSTÅL 2013, destacam a prática *Lean* para desenvolvimento de *software* como uma metodologia independente da engenharia de *software*, utilizando uma síntese de práticas dos princípios da filosofia enxuta para a construção de

softwares. Neste estudo, a prática *Lean* para desenvolvimento de *software* é considerada como uma abordagem independente dos conceitos ágeis, já anteriormente destacados.

O conceito enxuto (*Lean*) foi primeiramente identificado na empresa Toyota em meados dos anos 80. Wornack e Jones, nos anos 90, através da publicação *"The Machine That Changed The World"*, popularizaram o termo, e criou-se então uma nova filosofia, denominada "*Lean Thinking*". A filosofia é mapeada em torno de 5 principais princípios: (i) identificar valor para o cliente; (ii) identificar todas as etapas do fluxo de valor de cada produto, eliminado possíveis etapas que não agregam valor; (iii) criar um fluxo sequencial para cada produto; (iv) estabilizar e fazer com que o fluxo de valor seja puxado pelo cliente, a partir das suas reais necessidades; e (v) garantir a contínua melhoria de todo o processo, a fim de atingir o estado a perfeição, sem nenhum tipo de desperdício. A Figura 1 destaca o fluxo dos princípios acima destacados.

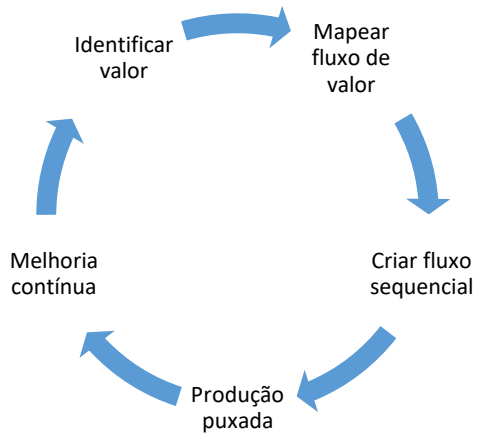

Figura 2 - Princípios do Lean Thinking. Fonte: Womack 1996

A principal essência do pensamento enxuto é a contínua eliminação de atividades desnecessárias, ou seja, eliminar os desperdícios, gastando o esforço em atividades que realmente agregam valor para o cliente. Desperdício é qualquer atividade executada que não agrega valor ao produto.

No contexto de desenvolvimento de *software*, POPPENDIECK 2003 destaca que se um componente foi desenvolvido e não mais está sendo utilizado, trata-se de um desperdício; se um requisito coletado e documentado está somente armazenado e não foi utilizado pela equipe e nem entregue ao cliente final, trata-se também de uma atividade desnecessária; se um desenvolvedor

criou rotinas de códigos a mais que não estão e não serão executadas, trata-se de uma atividade executada que gerou desperdício para todo o ciclo de vida do *software*.

O quer que esteja no caminho entre entregar um produto que satisfaça o cliente e a sua entrega propriamente dita, então é considerada uma atividade de desperdício (POPPENDIECK, 2003). A Figura 2 ilustra os sete principais desperdícios do pensamento enxuto, que serão, posteriormente, contextualizados no escopo de desenvolvimento de *software*.

Figura 3 - Desperdícios Lean Thinking. Fonte: Wornack 1996

FOGELSTROM 2010 destaca que os conceitos e práticas da filosofia *Lean* podem ser expandidas para demais áreas, além da indústria automotiva. Foi então que, no início dos anos 2000, Mary e Tom Poppendieck transferiram os princípios e práticas enxutas para o ambiente de desenvolvimento de *software* e chamou-lhes de "*Lean Software Development" (LSD)* (POPPENDIECK, 2003), tratado mais especificamente na sequência.

2.1.2 *Lean Software Development* (LSD)

Conforme destacado anteriormente, os benefícios obtidos a partir da implementação da filosofia ágil em projetos de desenvolvimento de *software* são indiscutíveis, entretanto, existem algumas questões de contextos específicos que não são contempladas a partir da utilização dos princípios ágeis. Dessa forma, para este estudo, a prática *Lean* para desenvolvimento de *software* é considerada como uma abordagem independente dos conceitos ágeis,

proporcionando assim, a sua utilização em conjunto com os métodos ágeis. Conceito este, adotado por diferentes autores (WANG 2012; PERNSTÅL 2013; CHOW 2008; NERUR 2005; MIDDLETON 2012; POPPENDIECK 2007; PETERSEN 2010; PETERSEN 2011; EBERT 2012).

FOGELSTRÖM 2010 destaca que os conceitos e práticas da filosofia *Lean* podem ser expandidas para demais áreas além da indústria automotiva. Foi então que, no início dos anos 2000, Mary e Tom POPPENDIECK transferiram os princípios e práticas enxutas para o ambiente de desenvolvimento de *software* e chamou-lhes de "*Lean Software Development*" (LSD) (POPPENDIECK , 2003).

Além dos cinco princípios *Lean* já apresentados, um dos fundamentos do pensamento enxuto é a contínua eliminação de atividades desnecessárias, ou seja, eliminar os desperdícios, usando os esforços em atividades que realmente agregam valor para o cliente (WORNACK 1991). Desperdício é qualquer atividade executada que não agrega valor ao produto. Para o ambiente de desenvolvimento de *software*, POPPENDIECK 2003 destaca os seguintes: o desenvolvimento de um componente que não está sendo usado, trata-se de um desperdício; um requisito coletado, documentado e não foi utilizado pela equipe e nem entregue ao cliente final, trata-se também de uma atividade desnecessária; desenvolvimento de rotinas de códigos que não estão e não serão executadas, é uma atividade que gerou desperdício para todo o ciclo de vida do *software*.

O quer que esteja no caminho entre entregar um produto que satisfaça o cliente e a sua entrega propriamente dita, é então considerada uma atividade de desperdício (POPPENDIECK , 2003).

POPPENDIECK 2003 traduz os sete desperdícios da filosofia Lean para o contexto de desenvolvimento de *software*, e propõe os seguintes: (i) Requisitos, (ii) Processos a mais, (iii) Funcionalidades a mais, (iv) Troca de tarefas, (v) Atrasos, (vi) Defeitos e (vii) Movimento, conforme ilustrado no Quadro 1.

Desperdício	Tradução para Desenvolvimento de *Software*
Requisito	Trabalhos parcialmente feitos tendem a perder sua credibilidade e comprometem o funcionamento do sistema.
Processos a mais	Burocracia, atividades, métricas e documentação desnecessária que não geram valor tendem a diminuir o tempo de resposta da equipe.
Funcionalidades a mais	Código não utilizado introduz complexidade ao sistema e tendem a diminuir a eficácia de manutenibilidade do sistema.
Troca de Tarefas	Trocar entre uma tarefa e outra, tendem a tornar a equipe mais improdutiva, perde-se muito tempo entre sair de uma atividade, concentrar-se na próxima e iniciar o seu desenvolvimento.
Atrasos	Atrasos no projeto tendem a aumentar o custo do sistema.
Defeitos	Defeitos não agregam valor, não satisfazem o cliente e custam caro para o projeto.
Movimento	Tempo e esforço gasto para encontrar informações do projeto. Não se deve perder tempo lendo paginas de um documento para encontrar informações.

Quadro 1 - Tradução dos 7 desperdícios. Fonte: POPPENDIECK 2003

2.1.3 *Agile Software Development* (ASD)

A indústria de desenvolvimento de *software* está se tornando cada dia mais competitiva devido a muitos motivos. Um deles é a própria evolução tecnológica, que exige produtos com níveis de qualidade cada vez melhores. Assim sendo, a capacidade de desenvolver e entregar produtos de maneira mais eficiente tornou-se algo desejado por qualquer empresa do setor (BIFFL, et. al 2005). Nesse sentido, os processos ágeis para desenvolvimento de *software* baseiam-se em valores e princípios que tornam o processo mais rápido e adepto à mudanças, de acordo com a necessidade de cada cliente (PETERSEN, 2010).

As metodologias ágeis de desenvolvimento de *software* nasceram a partir de uma reação contra os métodos tradicionais, que eram considerados como incapazes de atender às dinâmicas do mercado (RODRIGUEZ, 2013). A essência das práticas ágeis foi primeiramente apresentada na década de 90, por meio do Manifesto para Desenvolvimento Ágil de *Software*. Este é o resultado de um estudo de 17 profissionais de *software* que criaram uma filosofia comum,

nomeada como *“Agile”* (MISRA, et. al, 2012). Os valores e princípios expressos nesse manifesto constituíram um *framework* para o desenvolvimento ágil de *software*, que se baseia principalmente nos seguintes valores: (i) indivíduos e Interações, ao invés de processos e ferramentas; (ii) *software* funcionando, ao invés de compreensão de documentação; (iii) colaboração com o cliente, ao invés de contratos burocráticos; e (iv) rápida resposta à mudanças, ao invés de seguir o planejado. A Figura 1 ilustra os respectivos valores destacados.

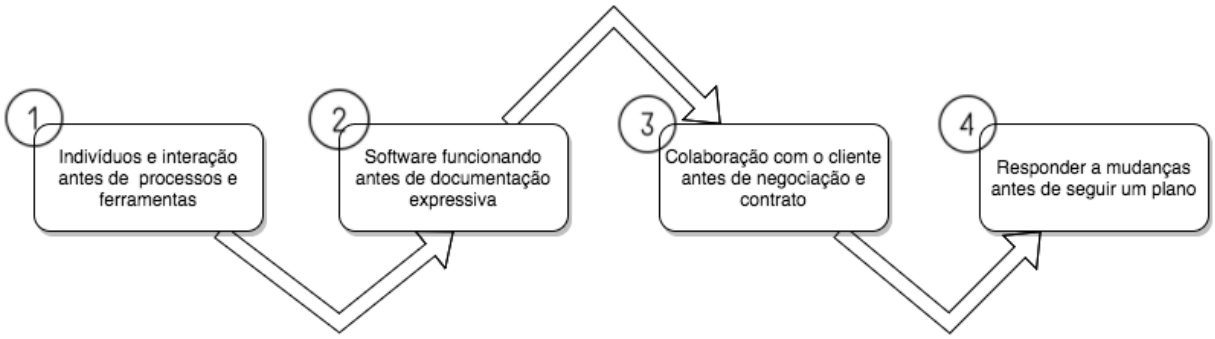

Figura 4 - Princípios ágeis. Adaptado de: Beck, et. al 2001

A partir da propagação do manifesto ágil, diferentes metodologias foram criadas, basicamente implementando o *framework* em questão, porém, apresentando focos determinados nos princípios específicos, dentre os anteriormente destacados. As práticas mais comuns e utilizadas são: *Scrum, Cristal methods, Feature-Driven-Development (FDD)* e *Extreme Programming (XP) (*KUPIAINEM, 2015).

As metodologias ágeis estão sendo cada vez mais aplicadas no mercado atual, uma vez que proporcionam menores custos de desenvolvimento, qualidade na entrega do produto final, maior produtividade da equipe de desenvolvimento e consecutivamente maior satisfação do cliente (MISHRA et. al, 2011). De facto, os referentes práticos provaram ser mais eficientes que as tradicionais, especialmente em projetos de tamanho e escopo moderados, uma vez que oferecem flexibilidade e agilidade que ajudam a produzir um sistema de alta qualidade em um período curto de tempo.

1.6.3.1 Lacunas dos Processos Ágeis

Apesar de todos os benefícios destacados pelas práticas ágeis, as suas aplicabilidades muitas vezes podem ser consideradas um desafio, devido a diferentes fatores, discutidos a seguir.

Um dos mais importantes princípios ágeis é a comunicação e partilha do conhecimento, o que gera alta dependência de comunicação informal entre desenvolvedores e clientes, a fim de compartilhar a ciência da informação entre as partes envolvidas (equipe de desenvolvimento e cliente) (CHAU et. Al, 2003; NERUR et. Al, 2005). Essa dependência pode ser um problema em projetos de grande porte, devido ao grande número de partes envolvidas e a quantidade de informação que “trafegam” entre os membros (MISHRA, 2011).

Outro desafio comum que se observa em projetos de grande porte que implementam as metodologias em questão, é o fato de que as práticas ágeis envolvem os clientes diretamente na licitação dos requisitos iniciais, o que demanda que eles tenham conhecimento total sobre todo o domínio que a aplicação está inserida. A complexidade implícita em todo o domínio da aplicação, que pode estar além da experiência do cliente, é então desconsiderada, tornando-se um risco para a entrega do produto final (CHAU et. al 2003; NERUR et. al 2005, MISHRA, 2011).

Com o propósito de se utilizar dos benefícios dos processos ágeis e melhorar as lacunas identificadas na aplicabilidade do *framework* em projetos de grande porte, muitas empresas observam a abordagem Enxuta (*Lean*), já difundida na manufatura, como uma possível saída para os problemas identificados pela aplicação das práticas ágeis (WANG 2012). Porém, a aplicação dos conceitos da filosofia *Lean* para o contexto de desenvolvimento de *software* não é tão simples, uma vez que a sua aplicação depende muito do contexto e dos objetivos do ambiente a ser introduzido (POPPENDIECK 2007).

De acordo com KOLA 2014, as práticas ágeis, resumidamente, consistem na utilização do conhecimento e experiência de mercado para observar as oportunidades mais rentáveis em um ambiente de alta flexibilidade; enquanto a filosofia *Lean* consiste no desenvolvimento de aplicações de alto valor agregado com um custo reduzido e melhor qualidade, através da eliminação de atividades que geram desperdício.

2.1.4 Desenvolvimento de *Software* de Grande Porte

O que faz com que as empresas de desenvolvimento de *software* tenham sucesso é, principalmente, atender as necessidades dos clientes no

tempo devidamente acordado. Assim sendo, para garantir a satisfação do cliente, o foco não pode ser apenas melhorias no processo de desenvolvimento do código. Todas as etapas do processo devem ser levadas em consideração, no entanto, as ferramentas da abordagem ágil, suprem, principalmente, a fase de desenvolvimento. Por isso, pesquisadores afirmam que as metodologias ágeis, popularmente conhecidas, não são suficientes para garantir a entrega de valor para o usuário final (COHEN et. al, 2003).

Além da lacuna destacada por COHEN et. al 2003, outros autores destacam diferentes desafios na utilização de paradigmas ágeis, principalmente, quando inserido em um contexto de desenvolvimento de *softwares* de grande porte. Um dos principais valores que as práticas ágeis exigem é o envolvimento do cliente durante todo o ciclo de desenvolvimento do produto, o que demanda que eles tenham conhecimento total sobre todo o domínio que o *software* está inserido. Levando em conta a complexidade e magnitude envolta de um *software* de grande porte, na maior parte dos casos, uma única pessoa não tem conhecimento de todo o domínio da aplicação, ou seja, os conhecimentos necessários estão além da experiência do cliente. Dessa forma, as experiências necessárias para a concepção de partes do sistema são desconsideradas, tornando um risco a entrega do produto final (CHAU et. al 2003; NERUR et. al 2005, MISHRA, 2011, DA SILVA et. al 2011).

Outra particularidade de um *software* de grande porte é a maneira que são concebidos. Segundo PERNSTÅL 2013 um sistema de grande porte é normalmente construído em subsistemas que posteriormente serão integrados. A construção desses subsistemas pode ser feita por pessoas e equipes distintas, cuja, em alguns casos, estão separadas geograficamente. Essas condições exigem algumas práticas não tão consolidadas em metodologias ágeis, como a documentação e gerenciamento de requisito, gestão visual do andamento do projeto, gestão de recursos, métricas do produto e da equipe, entre outras (BIFFL et. al, 2005).

Essas características de *softwares* de grande porte, fizeram com que empresas de desenvolvimento de *software*, buscassem investigar o processo como um todo, afim de melhorar a entrega do produto, o que resultou em um

crescente interesse no pensamento *Lean* para desenvolvimento de *software* (MUSAT, 2010).

3 ETAPAS DO DESENVOLVIMENTO DA DISSERTAÇÃO

Conforme apresentado na seção 1.5 Procedimentos Metodológicos e Estrutura do Trabalho, na sequência seguem as etapas elaboradas conforme o planejamento designado.

3.1 PROCESSO ENXUTO PARA DESENVOLVIMENTO DE *SOFTWARES* DE GRANDE PORTE: UM MAPEAMENTO DA LITERATURA

A primeira etapa que irá compor a dissertação em questão, trata-se de uma revisão bibliográfica intitulada PROCESSO ENXUTO PARA DESENVOLVIMENTO DE *SOFTWARES* DE GRANDE PORTE: UM MAPEAMENTO DA LITERATURA. O respectivo trabalho será submetido ao periódico IEE América Latina, cujo, de acordo com a classificação do ano de 2016 do Qualis, qualificou-o como B2 na área de Engenharias III. O objetivo do trabalho em questão foi o de cumprir com alguns dos escopos específicos destacados na seção 1.3.2, segue abaixo:

- Identificar, no contexto de desenvolvimento de *software* de grande porte, quais são os principais problemas de processo, e de que forma a aplicação das ferramentas *Lean* podem minimizar a sua ocorrência;
- Caracterizar particularidades do desenvolvimento de *software* de grande porte, tais como gerenciamento visual, métricas de equipe e produção, priorização de demandas de fabricação;
- Relatar os resultados da pesquisa científica em periódicos e congressos na área da pesquisa em questão.

3.1.1 Introdução

Alinhado com a alta velocidade com que seus requisitos e características evoluem, torna-se evidente que o seu desenvolvimento exige flexibilidade, tendo em vista as dúvidas que se apresentarão durante o processo de criação, e deverão ser resolvidas em tempo hábil, garantido a qualidade ao

produto final, juntamente com a sensibilidade às mudanças ambientais e exigências dos clientes (PINTO, 2015).

Métodos ágeis provaram ter uma agilidade e flexibilidade muito maior do que o desenvolvimento de *software* tradicional (QUMER et al., 2008) e são usados para produzir *software* de qualidade superior em um curto período de tempo. Independentemente do método utilizado, a maioria dos projetos de *software* se esforçam para alcançar um equilíbrio entre três objetivos básicos: qualidade, custo direto e entrega pontual.

Entretanto, as abordagens ágeis têm sido questionadas quando o assunto é documentação, gerenciamento visual de *software*, métricas de desenvolvimento, entre outras características essenciais durante o processo de desenvolvimento de aplicações de grande porte e para a entrega do produto final (BIFFL, et al., 2005). Desta forma, a fim de melhorar os métodos e processos de desenvolvimento de *software*, as empresas começaram a observar a abordagem *Lean* no processo de Manufatura com o viés para desenvolvimento de *software*.

Partindo da importância que têm o conhecimento de metodologias ágeis e *Lean* para desenvolvimento de sistemas computacionais, salienta-se que a pesquisa bibliográfica, especialmente dentro da área de Engenharia de *Software* ligada à Engenharia de Produção, é indispensável. Para tal, inicia-se um processo metodológico estruturado e consistente para selecionar materiais de cunho científico, que permitam ao pesquisador analisar a relevância da base bibliográfica, e especialmente identificar as lacunas presentes na sua linha de pesquisa (CAUCHICK et al., 2010).

A especial motivação para esta pesquisa é a carência de materiais e informações relevantes na área de engenharia e desenvolvimento de *software* de grande porte alinhados aos métodos ágeis, e especialmente a filosofia *Lean*. De acordo com Poppendieck 2006, o desenvolvimento de *software Lean* é a aplicação dos princípios da *Toyota product development system* para o desenvolvimento de *software*. Quando ele é aplicado corretamente, o desenvolvimento é de alta qualidade, além de ser realizado rapidamente e possuir um baixo custo.

Levando em consideração os assuntos expostos acima, surgiu então o principal problema desta pesquisa: Como manter os benefícios obtidos com as práticas ágeis e otimizar os métodos e os processos de desenvolvimento de *software* de grande porte? Para responder à essa pergunta, faz-se essencial um levantamento do estado da arte da temática que circunda este trabalho, o LSD.

O presente trabalho tem por objetivo apresentar uma melhor compreensão sobre o tema LSD, além de possibilitar a identificação de lacunas para próximas atividades de pesquisa. Para isso, tem-se os objetivos específicos: (i) Selecionar um portfólio bibliográfico sobre desenvolvimento *Lean* de *software* em publicações internacionais; (ii) evidenciar o portfólio os artigos, autores, periódicos e palavras-chaves de maior relevância e (iii) analisar o portfólio de artigos, identificando suas lentes (lacunas) de abordagem.

Sendo assim, este trabalho busca contribuir para a comunidade acadêmica, uma vez que alcançado o resultado da pesquisa, há ampliação do conhecimento nas áreas de interesse dos pesquisadores, especialmente Desenvolvimento *Lean* de *Sofware* (LSD). Como limitação dessa pesquisa, destaca-se que foram considerados, somente, estudos que enfatizavam as práticas ágeis de desenvolvimento de *software*, ou seja, estudos que tratavam das metodologias tradicionais foram desconsideradas devido ao foco direcionado à temática abordada nesse trabalho.

Dessa forma, esta pesquisa estrutura-se nas seguintes seções: esta seção inicial de introdução, a próxima que irá contextualizar, cada uma das temáticas apresentadas no decorrer deste trabalho, a terceira seção irá apresentar a metodologia utilizada para obter os resultados e em seguida serão apresentados os resultados relativos na aplicação da pesquisa, podendo ser dividido nas lentes analisadas. Por fim é apresentado as considerações finais e sugestões de novas pesquisas.

3.1.2 Referencial teórico

Esta seção tem por objetivo contextualizar as principais temáticas abordadas nesse trabalho: Processos ágeis para desenvolvimento de *software*, aplicações de grande porte e por fim, a filosofia *Lean* aplicada ao contexto de

desenvolvimento de sistemas, conhecida como *Lean Software Development* (LSD).

2.1.2.1 Processos ágeis para desenvolvimento de *software*

A indústria de desenvolvimento de *software* está se tornando cada dia mais competitiva devido a muitos motivos. Um deles é a própria evolução tecnológica, que exige produtos com níveis de qualidade cada vez melhores. Assim sendo, a capacidade de desenvolver e entregar produtos de maneira mais eficiente tornou-se algo desejado por qualquer empresa do setor. Nesse sentido, os processos ágeis para desenvolvimento de *software* baseiam-se em valores e princípios que tornam o processo mais rápido e adepto à mudanças, possibilitando assim, produtos cada vez melhores (PETERSEN, 2010; BIFFL, et. al 2005).

Métodos ágeis para desenvolvimento de software nasceram a partir de uma reação às práticas “tradicionais”, cujas são consideradas uma abordagem inflexível, com o propósito de tornar o processo de desenvolvimento de software, uma atividade eficiente e previsível. Os “tradicionalistas” defendem um planejamento extensivo e processos altamente documentados, para tornar o processo de desenvolvimento de software algo mais rigoroso. Em contraste, os métodos ágeis enfrentam o desafio da imprevisibilidade, confiando nas pessoas e suas técnicas criativas ao invés de processos e planejamentos. Ericksson 2005 define agilidade da seguinte forma: “Agilidade siginifica tirar o máximo de robustez associados aos processos tradicionais, para promover respostas rápidas e eficientes em ambientes instáveis e flexíveis”. (RODRIGUEZ 2013; ERICKSSON, 2005).

A essência das práticas ágeis foi primeiramente apresentada na década de 90, por meio do Manifesto para Desenvolvimento Ágil de *Software*. Este é o resultado de um estudo de 17 profissionais de *software* que criaram uma filosofia comum, nomeada como “Agile” (MISRHA, et. al 2012). Os valores e princípios expressos nesse manifesto constituíram um *framework* para o desenvolvimento ágil de *software*, que se baseia principalmente nos seguintes valores: (i) indivíduos e interações, ao invés de processos e ferramentas; (ii) *software* funcionando, ao invés de compreensão de documentação; (iii)

colaboração com o cliente, ao invés de contratos burocráticos; e (iv) rápida resposta às mudanças, ao invés de seguir o planejado.

A partir da propagação do manifesto ágil, diferentes metodologias foram criadas, basicamente implementando o *framework* em questão, porém, apresentando focos determinados nos princípios específicos, dentre os anteriormente destacados. As práticas mais comuns e utilizadas são: *Scrum*, *Cristal methods*, *Feature-Driven-Development* (FDD) e *Extreme Programming* (XP) (KUPIAINEM, 2015).

As metodologias ágeis estão sendo cada vez mais aplicadas no mercado atual, uma vez que proporcionam menores custos de desenvolvimento, qualidade na entrega do produto final, maior produtividade da equipe de desenvolvimento e consecutivamente maior satisfação do cliente (MISHRA et. al 2011). De fato, as práticas provaram ser mais eficientes que as tradicionais, especialmente em projetos de tamanho e escopo moderados (BOSH et. al 2011), uma vez que oferecem flexibilidade e agilidade que ajudam a produzir um sistema de alta qualidade em um período curto de tempo.

2.1.2.2 Desenvolvimento de *software* de grande porte

O que faz com que as empresas de desenvolvimento de *software* tenham sucesso é, principalmente, atender as necessidades dos clientes no tempo devidamente acordado. Assim sendo, para garantir a satisfação do cliente, o foco não pode ser apenas melhorias no processo de desenvolvimento do código. Todas as etapas do processo devem ser levadas em consideração, no entanto, as ferramentas da abordagem ágil, suprem, principalmente, a fase de desenvolvimento e produção de código-fonte. Por isso, pesquisadores afirmam que as metodologias ágeis, popularmente conhecidas, não são suficientes para garantir a entrega de valor para o usuário final (COHEN et. al, 2003).

Além da lacuna destacada por COHEN et. al 2003, outros autores destacam diferentes desafios na utilização de paradigmas ágeis quando aplicados em cenários distintos, principalmente, no contexto de desenvolvimento de *softwares* de grande porte, devido às especificidades de suas características.

Softwares de grande porte apresentam uma série de particularidades, cujas propiciam desafios para à atual engenharia de software e respectivas metodologias e processos de desenvolvimento. Softwares de grande porte, ou *large-scale software*, são construídos e mantidos de maneira evolutiva, ou seja, o produto passa por constantes alterações, vindas por equipes de desenvolvimento que trabalham de forma paralela. Perry 2001 destaca quatro principais propriedades de aplicações de grande porte: (i) Constante evolução do software; (ii) Complexidade da aplicação; (iii) Múltiplias dimensões do sistema (Modularidade); (iv) Distribuição do conhecimento entre os desenvolvedores.

Pernstal 2013 destaca que, devido à um software de grande porte ser composto de vários módulos, ou subsistemas, o desenvolvimento e/ou evolução de cada uma das partes, ocorrem em paralelo. A construção e/ou evolução desses subsistemas podem ser feitas por pessoas e equipes distintas, cujas, em alguns casos, estão separadas geograficamente. Essas condições exigem algumas práticas não tão consolidadas em metodologias ágeis, como a documentação e gerenciamento de requisito, gestão visual do andamento do projeto, gestão de recursos, métricas do produto e da equipe, priorização de novas demandas, entre outras necessidades (BIFFL et. al, 2005; CHAU et. al 2003; NERUR et. al 2005, MISHRA, 2011, DA SILVA et. al 2011; PERRY 2001).

Essas características de *softwares* de grande porte, fizeram com que empresas de desenvolvimento de *software*, buscassem investigar o processo como um todo, afim de melhorar a entrega do produto, o que resultou em um crescente interesse no pensamento Lean para desenvolvimento de *software* (MUSAT, 2010).

2.1.2.3 Lean software development (LSD)

Na literatura atual, os conceitos “ágeis” e “Lean”, quando aplicados no mesmo contexto, são pouco difundidas e a utilização desses termos é muitas vezes considerada ambígua e inconsistente (CONBOY, 2009). Alguns autores veem os dois termos como nomes diferentes para tratarem a mesma ocorrência, por exemplo, nos estudos de KETTUNEN 2008, CHOW 2008, nenhuma distinção significativa é feita entre as duas práticas. Entretanto, outros estudos, como por exemplo, WANG 2012, PERNSTÅL 2013, destacam a prática Lean para desenvolvimento de *software* como uma metodologia independente da

engenharia de *software*, utilizando uma composição dos princípios da filosofia enxuta para a construção de softwares.

Conforme destacado anteriormente, os benefícios obtidos a partir da implementação da filosofia ágil em projetos de desenvolvimento de *software* são indiscutíveis, entretanto, existem algumas questões de contextos específicos que não são contempladas a partir da utilização dos princípios ágeis. Dessa forma, para este estudo, a prática Lean para desenvolvimento de *software* é considerada como uma abordagem independente dos conceitos ágeis, proporcionando assim, a sua utilização em conjunto com os métodos ágeis. Conceito este, adotado por diferentes autores (WANG 2012; PERNSTÅL 2013; CHOW 2008; NERUR 2005; MIDDLETON 2012; POPPENDIECK 2007; PETERSEN 2010; PETERSEN 2011; EBERT 2012).

O conceito enxuto (Lean) foi primeiramente identificado na empresa Toyota em meados dos anos 80. WORNACK e JONES, nos anos 90, através da publicação “The Machine That Changed The World”, popularizaram o termo, e criou-se então uma nova filosofia, denominada “Lean Thinking”. A filosofia é mapeada em torno de 5 principais princípios: (i) identificar valor para o cliente; (ii) identificar todas as etapas do fluxo de valor de cada produto, eliminado possíveis etapas que não agregam valor; (iii) criar um fluxo sequencial para cada produto; (iv) estabilizar e fazer com que o fluxo de valor seja puxado pelo cliente, a partir das suas reais necessidades; e (v) garantir a contínua melhoria de todo o processo, a fim de atingir o estado da perfeição, sem nenhum tipo de desperdício.

FOGELSTRÖM 2010 destaca que os conceitos e práticas da filosofia Lean podem ser expandidas para demais áreas além da indústria automotiva. Foi então que, no início dos anos 2000, Mary e Tom POPPENDIECK transferiram os princípios e práticas enxutas para o ambiente de desenvolvimento de *software* e chamou-lhes de “*Lean Software Development*” (LSD) (POPPENDIECK , 2003).

Além dos cinco princípios Lean anteriormente apresentados, um dos fundamentos do pensamento enxuto é a contínua eliminação de atividades desnecessárias, ou seja, eliminar os desperdícios, usando os esforços em

atividades que realmente agregam valor para o cliente (WORNACK, 1991). Desperdício é qualquer atividade executada que não agrega valor ao produto. Para o ambiente de desenvolvimento de *software*, POPPENDIECK (2003) destaca os seguintes: o desenvolvimento de um componente que não está sendo usado, trata-se de um desperdício; um requisito coletado, documentado e não foi utilizado pela equipe e nem entregue ao cliente final, trata-se também de uma atividade desnecessária; desenvolvimento de rotinas de códigos que não estão e não serão executadas, é uma atividade que gerou desperdício para todo o ciclo de vida do *software*.

Tudo que esteja no caminho entre entregar um produto que satisfaça o cliente e a sua entrega propriamente dita, é então considerada uma atividade de desperdício (POPPENDIECK , 2003). A Figura 1 ilustra os sete principais desperdícios do pensamento enxuto.

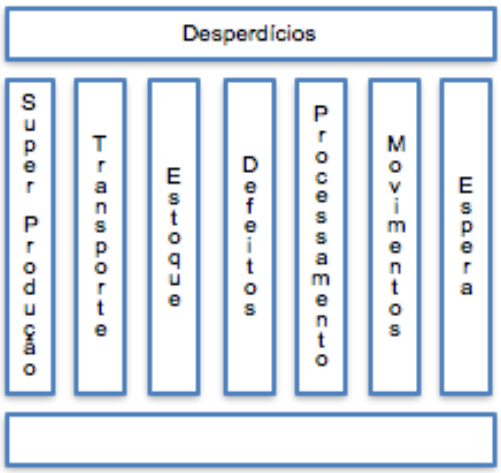

Figura 1 - Desperdícios Lean Thinking. Fonte: Wornack 1991

POPPENDIECK 2003 traduz os sete desperdícios da filosofia Lean para o contexto de desenvolvimento de *software*, e propõe os seguintes: (i) Requisitos, (ii) Processos a mais, (iii) Funcionalidades a mais, (iv) Troca de tarefas, (v) Atrasos, (vi) Defeitos e (vii) Movimento, conforme ilustrado no Quadro 1.

Quadro 1 - Tradução dos 7 desperdícios. Fonte: Adaptado de Poppendieck 2003

Produção	Desenvolvimento de Software
Superprodução	Funcionalidades extras: Do mesmo modo que a superprodução é o pior dos 7 desperdícios da produção, uma funcionalidade sem valor para o cliente, é o pior dos desperdícios para desenvolvimento de software.
Transporte	Transferência de controle: A grande questão aqui, está em como minimizar a perda quando há transferência de conhecimento tácito entre colaboradores.
Estoques	Trabalho inacabado: O objetivo é partir do início do trabalho para um código integrado, testado, documentado e entregue, tudo isso, em um fluxo rápido e único.
Defeitos	Defeitos: Quanto mais cedo e mais frequente forem realizados os testes, menos a chance de defeitos serem encontrados pelos clientes em ambiente de produção.
Processamento adicional	Reaprendizagem: Redescobrir algo que que sabíamos, porém, esquecemos é a definição de “retrabalho” no desenvolvimento de software.
Movimentação	Troca de tarefas: Executar múltiplas tarefas ao mesmo tempo, ao invés de executar uma a uma, entregando o seu valor o quanto antes.
Esperas	Atrasos: Esperar a disponibilidade de pessoas que estão trabalhando em outras áreas é um grande desperdício proveniente do atraso.

2.1.3 Metodologia da pesquisa

2.1.3.1 Instrumento de Intervenção

De acordo com Tasca 2010 um processo de revisão da literatura se inicia a partir do lançamento de uma problemática, ou seja, algo que motive o pesquisador a procurar informações sobre um determinado tema em bibliotecas ou acervos digitais. Devido à variedade e quantidade de informações contidas nos diferentes meios de pesquisa, faz-se necessário a utilização de um processo estruturado de busca e pesquisa, a fim de tomar conclusões sobre o conjunto de informações coletadas e analisadas. Em conjunto, a bibliometria e a revisão sistêmica, são ferramentas extremamente eficazes (JUNIOR, 2012).

Segundo Ensslin 2010, a bibliometria ou análise bibliométrica é um processo de avaliação que permite a análise de um conjunto definido de artigos por meio de métodos estatísticos, objetivando a gestão de informações de um dado assunto, como por exemplo: periódicos com maior relevância e maior número de publicações de um determinado tema. Para a análise bibliométrica

do presente trabalho, foi escolhido uma análise estatística contemplando os seguintes aspectos: Periódicos mais aludidos, artigos mais citados, autores com maior participação e palavras-chave mais utilizadas.

Segundo Mozzato e Grzybovski, 2011, a revisão sistêmica consiste em um conjunto de técnicas de análise sistêmica que objetivam enriquecer os dados já coletados e prover maior conhecimento. A partir de então, as lacunas nos respectivos trabalhos são identificadas, cujas podem direcionar futuros trabalhos sobre a temática abordada. Para a realização da análise sistêmica deste trabalho, foram utilizadas as principais lacunas encontradas nos artigos do portfólio bibliográfico, bem como os temas mais relevantes da abordagem do trabalho.

A fim de formar um portfólio bibliográfico, capaz de prover condições para uma análise bibliométrica e sistêmica, a metodologia utilizada para a concepção do respectivo trabalho científico, é o Proknow-C (*Knowledge Development Process – Constructivist*) (ENSSLIN et al., 2010). O respectivo método é composto de 3 principais etapas, conforme ilustrado na Figura 2:

Figura 2 – Etapas Proknow – C. Fonte: Adaptado de Tasca 2010

2.1.3.2 Seleção dos Artigos

A seleção dos artigos contempla uma série de atividades executas que tem como artefato final, o portfólio bibliográfico. Cada uma das atividades executadas e respectivos resultados são apresentados no Quadro 2.

Quadro 2 - Seleção dos artigos. Fonte: Dados da pesquisa

	Atividade	**Resultados**
1	Definição dos eixos de pesquisa e palavras chave.	Os eixos de pesquisa e respectivos conjunto de palavras chave.
2	Busca nas bases de dados.	Foram encontrados 2263 trabalhos científicos a partir da busca nas bases de dados ISI e Scopus.
3	Teste de aderência das palavras chave.	Não houve necessidade de inclusão de nenhuma nova palavra chave.
4	Alinhamento com os eixos de pesquisa.	Eliminando artigos duplicados e não alinhados com a temática, restaram 309 trabalhos.
5	Reconhecimento científico.	Dos 309 trabalhos resultados, 100 foram identificados com reconhecimento científico, os outros 209 passaram por nova análise.
6	Leitura do resumo.	84 artigos permaneceram.
7	Leitura integral.	24 artigos restaram a partir da exclusão dos trabalhos não alinhados e não disponíveis.

(i) Definição dos eixos de pesquisa e palavras chave: A primeira atividade executada antes da realização das buscas nas respectivas bases de dados, foi a definição dos eixos principais que norteiam as palavras chave. Sendo assim, foi definido os dois principais eixos: (1) Desenvolvimento de *Software* e (2) Desenvolvimento Enxuto. Para o primeiro, as seguintes palavras-chave foram definidas: *Software Development, Startup, Information System, Inovative Products e Maturity*. As seguintes palavras-chave foram criadas para o segundo eixo: *Lean, Agile, Leagile, Kanban e LSD*.

(ii) Busca nas bases de dados: A partir das definições descritas na seção anterior, iniciou-se o processo de busca nas bases de dados. As buscas foram feitas a partir da combinação cruzada de cada uma das palavras chave previamente definidas. A partir das 10 palavras, foram criadas um total de 25 combinações distintas, as quais foram utilizadas em pesquisas nas duas bases de dados (ISI e Scopus), totalizando 2263 artigos localizados.

(iii) Teste de aderência das palavras chave: Após o primeiro resultado, o processo prescreve a realização de um teste de aderência nas palavras-chave previamente definidas. A etapa é importante para definir se há a necessidade da incorporação de novos termos não inseridos anteriormente. O processo resume-se à escolha aleatória de 2 a 3 artigos, leitura dos seus resumos a fim de verificar

se está enquadrado em algum dos eixos da pesquisa. Caso sim, é feito uma análise das palavras chave de cada um dos trabalhos a fim de verificar se os termos anteriormente escolhidos estão contemplados nos referentes.

Para o teste de aderência, os seguintes artigos foram selecionados:

- Wang, X. Conboy, K. Cawley, O. "Leagile" *software* development: An experience report analysis of the application of Lean approaches in agile *software* development. Journal of Systems and *Software*. Vol. 85. P. 1287-1299. 2012;
- Ebert, C. Abrahamsson, P. Oza, N. Lean *software* development. IEEE *Software*. Vol. 29. P. 22- 25. 2012;

Dos artigos, as seguintes palavras chave estão contidas:

- *Agile, software development, Leagile, Scrum, Large-scale software systems, Lean product development, Software process improvement, Agile, Kanban, Lean software development, Software engineering, Agile software development, Software development process e Product development*.

Das palavras chave encontradas nos trabalhos escolhidos, 5 delas foram contempladas nas combinações, ou seja, quase 40% delas já estão consideradas. O resultado foi a não necessidade de incorporação de novos termos.

(iv) Alinhamento com os eixos de pesquisa: O passo seguinte do processo é fazer uma análise do título de cada um dos artigos, a fim de garantir que estão enquadrados em algum dos eixos de pesquisa do presente trabalho. Com o objetivo de auxiliar no processo de análise bibliográfica, os 2263 artigos foram importados na ferramenta de gerenciamento bibliográfico *EndNote* (http://endnote.com) e então foi realizado o processo de eliminação de artigos duplicados. A eliminação foi então realizada na própria ferramenta, cuja encontrou um total de 397 artigos repetidos. Fazendo a eliminação dos referentes, teve-se um total de 1866 artigos. Após a leitura de cada um dos títulos, decidiu-se por eliminar 1555 artigos, restando um total de 309 trabalhos, cujos estavam alinhados com a temática do trabalho.

(v) Reconhecimento científico: Com o propósito de manter no portfólio, artigos com maior relevância acadêmica, a respectiva etapa objetiva eliminar trabalhos com pouco ou nenhum reconhecimento científico e manter aqueles com maior relevância. Para tanto, tabulou-se a quantidade de vezes que cada trabalho foi citado por outros autores, ordenando-os do maior ao menor e estabeleceu-se um ponto de corte, que determinará o conjunto de trabalhos com maior representatividade científica. Como entrada da referente etapa teve-se os 309 artigos já alinhados. Utilizou-se a ferramenta Google Scholar (http://scholar.google.com) para encontrar o número de citações de cada trabalho. As pesquisas foram realizadas entre os meses outubro a novembro de 2015. Após a tabulação dos resultados, estabeleceu-se um ponto de corte de 20 citações, que representam 90,15% do total de citações, neste escopo, aqueles trabalhos com menos que 20 citações são considerados com menor reconhecimento científico e consecutivamente, excluídos do portfólio bibliográfico.

(vi) Leitura do resumo: Uma vez selecionado os artigos com maior representatividade científica, os seus resumos foram analisados com o propósito de identificar se os mesmos estão alinhados com o foco da pesquisa em questão. Dos 100 artigos analisados, 48 deles foram descartados devido a não estarem aderente à temática da pesquisa em questão. Por fim 52 trabalhos foram mantidos.

(vii) Leitura integral: A última etapa da concepção do portfólio bibliográfico, é a leitura integral de cada um dos trabalhos, a fim de garantir a sua relação à temática da pesquisa em questão. Como entrada desta etapa teve-se 84 trabalhos. O próximo passo é procurar nas respectivas bases de dados, cada um dos artigos na íntegra e prosseguir com a leitura. A partir desta etapa, 60 trabalhos foram descartados por não estarem alinhados ao tema do trabalho ou por não estarem disponíveis na íntegra nas bases de dados *ISI* ou *Scopus*. Por fim, formulou-se o portfólio bibliográfico com um total de 24 artigos. A partir de então obteve-se-se o portfólio bibliográfico do trabalho em questão, o qual será referência para todas as análises seguintes. O Quadro 3 apresenta o portfólio bibliográfico.

Quadro 3 - Portfólio Bibliográfico. Fonte: Dados da Pesquisa

Nº	Artigo
1	Baskerville, R. Ramesh, B. Levine, L. Pries-Heje, J(2006). "High-speed *software* development practices: What works, what doesn't". **IT Professional: 29-36**
2	Baptista, G. L. Vanalle, R. M. Salles, J. A. A(2015). "A *Software* Development Process Model Integrated to a Performance Measurement System". **IEEE Latin America Transactions: 739 – 745**
3	Chow, T. Cao, D. B(2008). "A survey study of critical success factors in agile *software* projects". **Journal of Systems and *Software*: 961-971**.
4	Da Silva, I. F. Da Mota Silveira Neto, P. A. O'Leary, P. De Almeida, E. S. De Lemos Meira, S. R(2011). "Agile *software* product lines: A systematic mapping study". ***Software* - Practice and Experience: 899-920**
5	Dybå, T. Dingsøyr, T(2008) "Empirical studies of agile software development: A systematic review". **Information and Software Technology: 9-10**
6	Ebert, C. Abrahamsson, P. Oza, N(2012). "Lean *software* development". **IEEE *Software*: 22-2**
7	Fogelström, N. D. Svahnberg, T. G. M. Olsson, P(2010). "The impact of agile principles on market-driven *software* product development". **Journal of Software Maintenance and Evolution: 53-80**
8	Hansson, C. Dittrich, Y. Gustafsson, B. Zarnak, S(2005). "How agile are industrial *software* development practices?". **Journal of Systems and *Software*: 1295-1311**
9	Kettunen, P(2009). "Adopting key lessons from agile manufacturing to agile *software* product development-A comparative study". **Technovation: 408-422**
10	Middleton, P. Joyce, D(2012). "Lean *software* management: BBC worldwide case study". **IEEE Transactions on Engineering Management: 20-32**
11	Mishra, D. Mishra, A(2011). "Complex *software* project development: Agile methods adoption". **Journal of *Software* Maintenance and Evolution: 549-564**
12	Misra, S. C. Kumar, V. Kumar, U(2009). "Identifying some important success factors in adopting agile *software* development practices". **Journal of Systems and *Software*: 1869-1890**
13	Nerur, S. Mahapatra, R. Mangalaraj, G(2005). "Challenges of migrating to agile methodologies". **Communications of the ACM: 72-78**
14	Petersen, K. Wohlin, C(2011). "Measuring the flow in Lean *software* development". ***Software* - Practice and Experience: 975-996**

- continua

- continuação

15	Petersen, K. Wohlin, C(2010). "*Software* process improvement through the Lean Measurement (SPI-LEAM) method". **Journal of Systems and *Software*: 1275-1287**
16	Poppendieck, M. Cusumano, M. A(2012). "Lean *software* development: A tutorial". **IEEE *Software*: 26-32**
17	Zhi, J. Garousi-Yusifoȟlu, V. Sun, B. Garousi, G. Shahnewaz, S. Ruhe, G(2015). "Cost, benefits and quality of *software* development documentation: A systematic mapping**". Journal of Systems and *Software*: 175-198.**
18	Torrecilla-Salinas, C. J. Sedeño, J. Escalona, M. J. Mejías, M(2015). "Estimating, planning and managing Agile Web development projects under a value-based perspective". **Information and *Software* Technology: 124-144**
19	Suomalainen, T. Kuusela, R. Tihinen, M(2015). "Continuous planning: An important aspect of agile and Lean development". **International Journal of Agile Systems and Management: 132-162**
20	Serrador, P. Pinto, J. K. "Does Agile work? - A quantitative analysis of agile project success"(2015). **International Journal of Project Management: 1040 – 1051**
21	Sharp, H. Robinson, H. Petre, M(2009). "The role of physical artefacts in agile *software* development: Two complementary perspectives". **Interacting with Computers: 108 - 116**
22	Van Waardenburg, G. Van Vliet, H(2013). "When agile meets the enterprise". **Information and *Software* Technology: 2154-2171**
23	Wang, X. Conboy, K. Cawley, O(2012). "Leagile" *software* development: An experience report analysis of the application of Lean approaches in agile *software* development. **Journal of Systems and *Software*: 1287-1299**
24	Pernstål J. Feldt R. Gorschek T(2013). The Lean gap: A review of Lean approaches to large-scale *software* systems development. **Journal of Systems and *Software*: 2797 - 2821**

O Quadro 3 apresenta o portfólio bibliográfico do referente trabalho de seleção,os quais serão objeto das próximas etapas do processo: (2) Análise Bibliométrica e (3) Revisão Sistêmica.

2.1.4 Resultados

2.1.4.1 Análise Bibliométrica

Segundo Ensslin 2010, bibliometria é um processo quantitativo de mensuração de dados estatísticos a partir de um conjunto pré-definido de

artigos, com o propósito de identificar periódicos, autores e artigos mais relevantes no contexto de uma pesquisa científica.

O conjunto de artigos selecionados para a análise bibliográfica é o portfólio bibliográfico (Tabela 3). O processo em questão, foi dividido em três principais etapas: (1) Análise dos artigos do portfólio bibliográfico, (2) Análise das referências dos artigos do portfólio e (3) Análise combinada dos artigos e das referências. A seguir é apresentado em detalhes cada uma das etapas e respectivas conclusões.

2.1.4.1.1 Análise dos artigos do portfólio bibliográfico

A primeira análise realizada a partir dos artigos selecionados, é a busca pelo grau de relevância dos periódicos. Como resultado, nota-se que um periódico se destaca dos demais, trata-se do Journal of System and *Software*. Dos 24 trabalhos selecionados, 8 deles foram publicados no periódico destacado, ou seja, quase 30% dos trabalhos do portfólio foram expostos no *journal* em questão. A Figura 5 ilustra o resultado acima descrito.

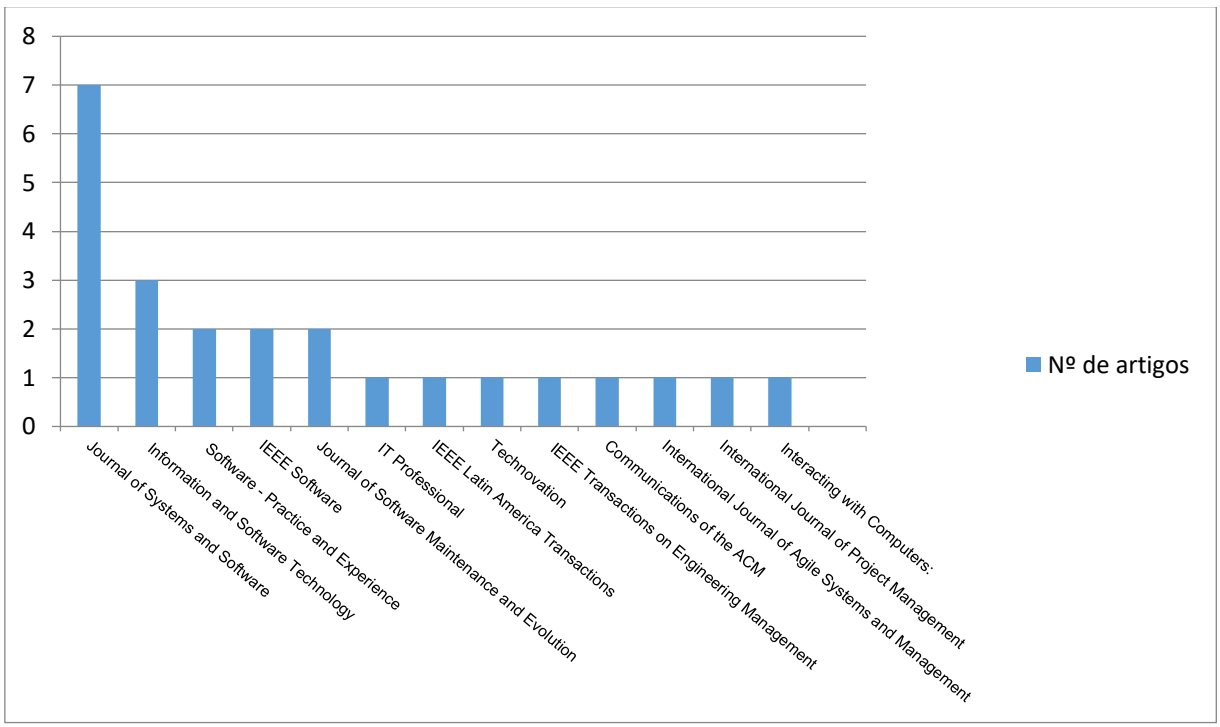

Figura 5 - Relevância dos periódicos do Portfólio. Fonte: Dados da Pesquisa

Na próxima etapa do processo de análise bibliométrica, foi feito a análise do reconhecimento científico dos artigos presentes no portfólio bibliográfico. Destaca-se o trabalho *Empirical studies of agile software*

development: A systematic review, o presente trabalho, apresentou mais de 1100 citações em pesquisa realizada entre os meses outubro e novembro de 2015. O valor representa cerca de 38% do total de citações de todos os artigos contidos no portfólio. Já aquele que menos foi citado, o trabalho *Continuous planning: An important aspect of agile and Lean development*, não apresentou nenhuma citação. A Figura 6 ilustra o resultado da referente análise.

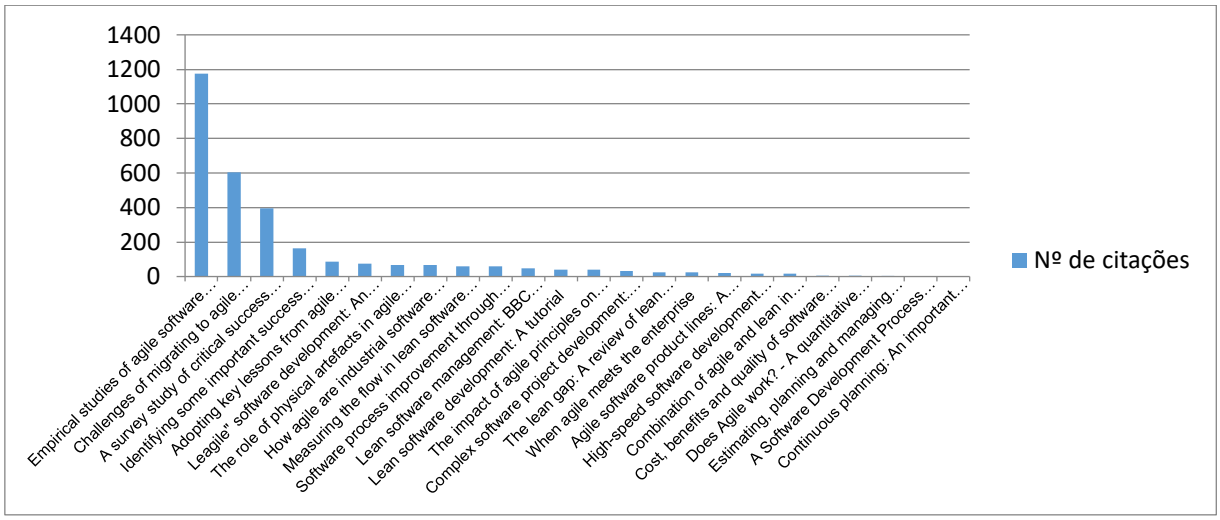

Figura 6 - Relevância das citações do Portfólio. Fonte: Dados da Pesquisa

Outra etapa do processo, é a relevância dos autores do portfólio bibliográfico. A referente análise revelou um número considerável de autores distintos. Do total de 24 trabalhos, 68 autores foram encontrados, média de 2.9 autores por trabalho. Conclui-se dessa forma, que as publicações estão espalhadas entre diferentes autores. Vale a pena destacar apenas dois autores: Petersen K. e Wohlin, C., P., os quais possuem 2 artigos cada. Um trabalho de cada um foi publicado no periódico *Journal of System and Software* destacado como aquele de maior relevância dentre os trabalhos do portfólio bibliográfico.

A última fase desta primeira etapa, é a análise da relevância das palavras chave. Com o propósito de identificar termos mais importantes para o contexto da pesquisa em questão, as palavras chave de todos os trabalhos do portfólio bibliográfico foram tabuladas e o número de vezes que são utilizadas foi também catalogado. Dessa forma, as seguintes palavras chave se destacaram: *Software Design, Agile e Software Development*, aparecendo, respectivamente, em 16, 8 e outras 8 vezes nos artigos selecionados. É importante ressaltar que dos termos mais relevantes, 2 deles foram contemplados no processo de

definição dos eixos e palavras chave, apresentados na seção 2.3 deste trabalho. A Figura 7 apresenta uma parte do resultado da análise bibliométrica das palavras chave mais relevantes no portfólio bibliográfico.

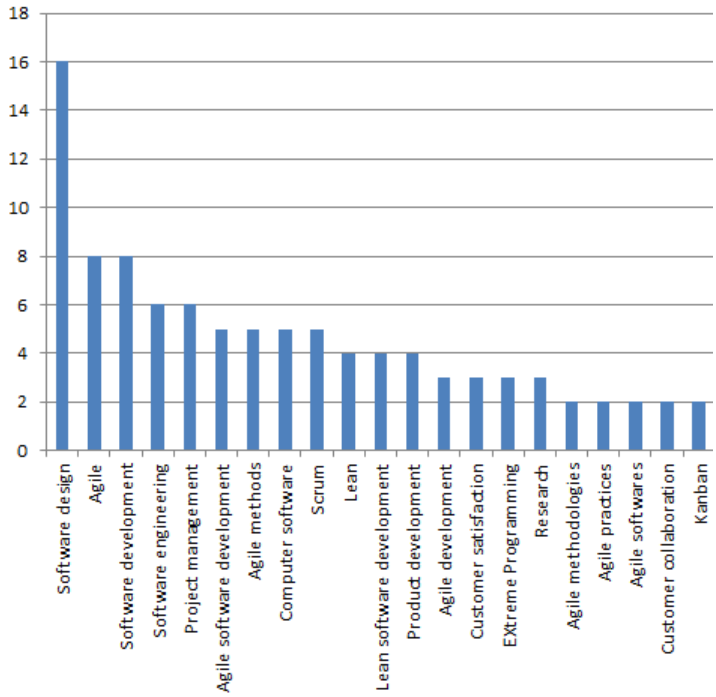

Figura 7 - Grau de relevância das palavras chave no portfólio bibliográfico. Fonte: Dados da pesquisa

2.1.4.1.2 Análise das referências dos artigos do portfólio

Neste tópico, a análise a ser realizada será feita do ponto de vista das referências dos artigos selecionados no portfólio bibliográfico, ou seja, para cada artigo do portfólio foram extraídas suas referências.

A análise bibliométrica das referências dos 24 artigos científicos contidos no portfólio bibliográfico resultou em um total de 214 trabalhos. Ao se realizar uma análise sobre os periódicos que os trabalhos foram publicados, destacam-se os seguintes: *IEE Computer, IEE Software e Journal of System and Software*, os quais apresentam respectivamente 28, 26 e 19 publicações cada, representando 31% do total das publicações.

A seguir procurou-se revelar os autores que mais vezes foram referenciados. Destacaram-se os seguintes: *Cockburn A. e Wohlin C.,* ambos foram citados 6 vezes cada. É importante ressaltar que o segundo foi também destaque quando a mesma análise foi feita sobre os artigos do portfólio

bibliográfico. A Figura 8 ilustra o comparativo entre o número de citações que cada autor teve na análise do portfólio e quantas vezes os mesmos foram referenciados nos artigos do portfólio. Destacam-se, além dos autores já separados na análise de relevância dos autores do portfólio, o autor *Dyba T.* com 5 referências nos trabalhos e 1 artigo selecionado.

É importante destacar que o trabalho publicado pelo autor *Dyba T*. é aquele que se destacou, anteriormente, na análise da representatividade científica dos artigos do portfólio bibliográfico, apresentando mais de 1100 citações.

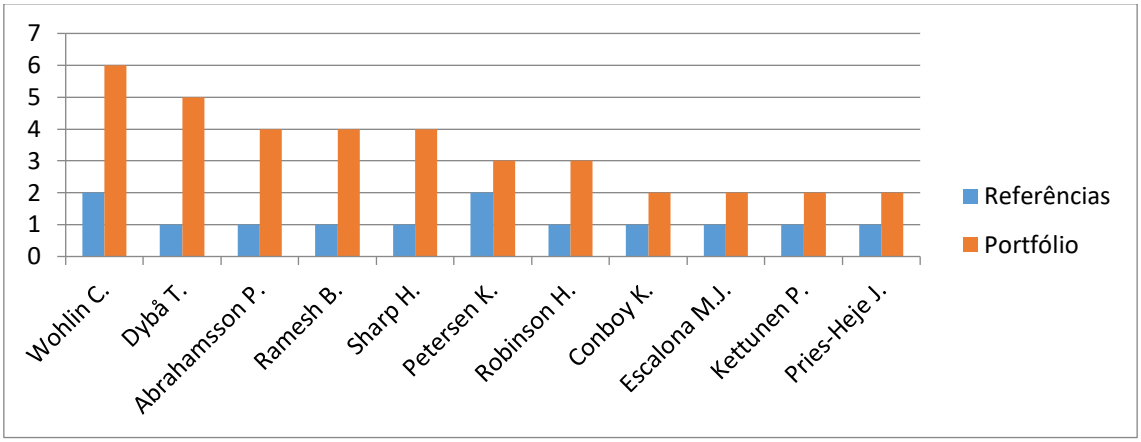

Figura 8 - Comparação da relevância dos autores no portfólio e nas referências. Fonte: Dados da pesquisa

2.1.4.1.3 Análise combinada dos artigos e das referências

Na última etapa da análise bibliométrica, faz-se uma análise combinada, comparando os resultados da primeira e segunda etapa. Segundo Marafon 2012, esta etapa tem o objetivo de indicar quais são os periódicos e autores mais relevantes no meio acadêmico para o contexto da pesquisa.

Objetivando alcançar o resultado destacado, a análise combinada foi dividida em duas etapas: (1) Análise combinada entre os periódicos onde estão publicados os artigos do portfólio bibliográfico com os periódicos onde estão publicados os trabalhos presentes nas referências e (2) Análise combinada entre o número de vezes que um autor foi citado no portfólio bibliográfico com os o número de citações dos autores presentes nas referências.

Para facilitar a análise e identificação dos periódicos com maior relevância, foi concebido um gráfico de quadrantes, conforme Figura 9. Foi

possível destacar que o periódico *Journal of System and Software* é aquele que se posicionou no melhor dos quadrantes do gráfico, aquele que identifica os periódicos que obtiveram destaque no portfólio bibliográfico e nas referências.

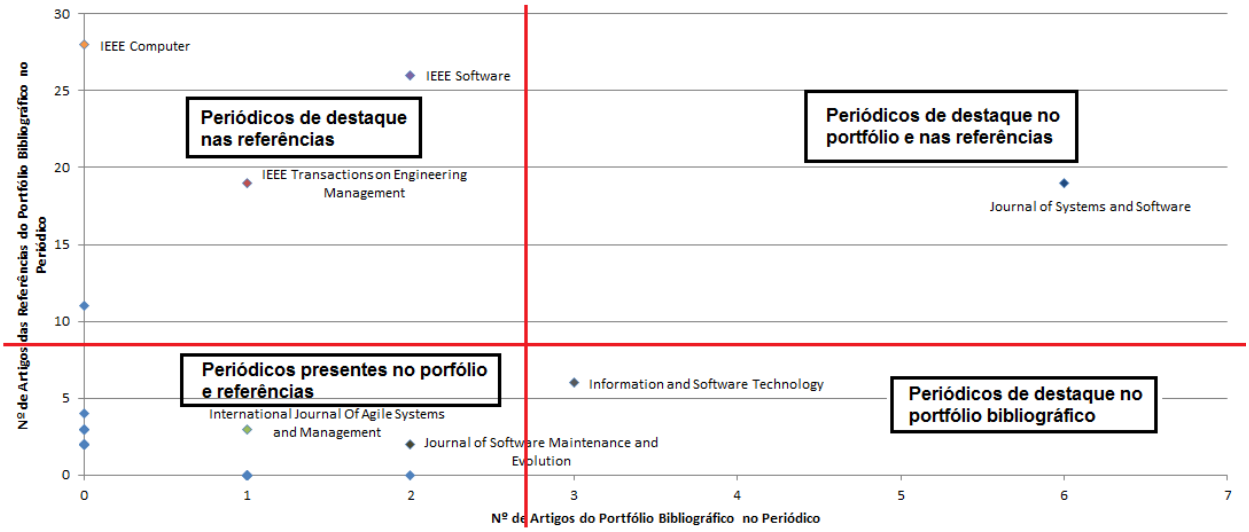

Figura 9 - Análise combinada dos periódicos. Fonte: Dados da pesquisa

Por fim, em análise à Figura 10, destaca-se o autor Petersen, *K.* por ter sido destacado como autor de evidência nos artigos do portfólio bibliográfico e suas referências.

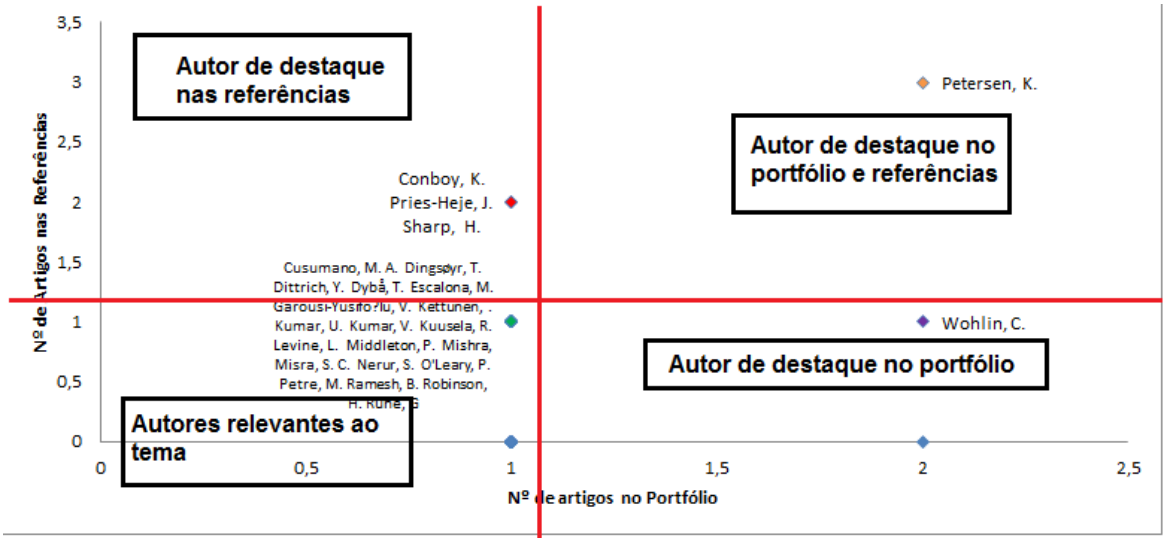

Figura 10 - Análise combinada dos autores. Fonte: Dados da pesquisa

2.1.4.2 Análise Sistêmica

Após a definição do portfólio bibliográfico e a análise quantitativa de dados estatísticos dos trabalhos selecionados, desenvolveu-se a etapa de revisão do conteúdo, ou Revisão Sistêmica. O processo de análise sistêmica

permite identificar as lacunas encontradas na literatura sobre um determinado tema de pesquisa, cujos podem ser analisados de forma mais profunda e relevante. (BORTOLUZZI, 2011)

Essas lacunas também são chamadas de lentes, cujo objetivo é evidenciar os destaques e oportunidades sobre o contexto da pesquisa em questão. Para a definição das respectivas lentes devem ser analisadas as problemáticas e temas que circundam cada um dos artigos que compõe o portfólio bibliográfico, e partir de então, descrever a cerca do posicionamento de cada trabalho. A escolha de cada lente foi feita com o intuito de aprofundar o estudo em temas relevantes para a temática em questão.

Devido a temática Lean para desenvolvimento de *software* ser pouco difundida na literatura (CONBOY, 2009), optou-se por compreender melhor a forma que os dados estão sendo coletados e se estão sendo aplicados estudos práticos. O objetivo dessa análise é, além de identificar as metodologias utilizadas, quais são as ferramentas e técnicas (Lean e *Agile*) aplicadas e quais os resultados obtidos e destacados. Para corresponder às respectivas expectativas, foram criadas as duas primeiras lentes: Lente 1: Coleta e Análise dos Dados e Lente 2: Aplicação prática dos estudos.

Apesar de originalmente, Lean para desenvolvimento de *software*, ser visto como mais uma instancia dos métodos ágeis (DYBA, 2009; HIGHSMITH, 2002; WANG 2012), nos últimos anos, pesquisadores e estudiosos têm abordado o desenvolvimento de *software* enxuto como uma nova categoria, ou seja, como uma teoria e não mais como um processo tal qual Scrum ou XP. (WANG, 2012). Alguns estudos argumentam que o Lean deve ser aplicado em um nível mais organizacional e gerencial, cujos métodos ágeis não conseguem uma abordagem satisfatória, objetivando assim, aumentar a eficiência em projetos e equipes. (SMITH, 2007).

A necessidade de ser efetivo é ainda mais evidente quando o escopo do projeto é relevante para o contexto, ou seja, para sistemas considerados *softwares* de grande porte, a necessidade de gerir o projeto e as equipes, tendem a ser mais necessária, devido às dificuldades e particularidades que circudam um projeto desse porte (BJORNSON, 2016; BEGEL, 2008; PERRY, 2001;

WOLVERTON, 1974). Objetivando analisar a forma que os autores destacam Lean para desenvolvimento de *software*, foi criado a Lente 3: Apresentação da Filosofia Lean, e por fim, foi criada a Lente 4: Escopo do projeto, com o intuito de analisar a forma que os pesquisadores destacaram o escopo do *software* como sendo relevante para a aplicação das práticas *Agile* e *Lean*.

Definidas as lentes da pesquisa, as próximas seções, descrevem os resultados da análise sistêmica dos artigos do portfólio. O Quadro 3 apresenta as lentes definidas para a execução e sequência da pesquisa.

Quadro 4 – Lentes. Fonte: Dados da pesquisa

	Lentes
1	Coleta e Análise dos Dados
2	Aplicação prática dos estudos
3	Apresentação da Filosofia Lean
4	Escopo do projeto

2.1.4.2.1 Lente 1 – Coleta e Análise dos Dados

Para considerar a análise sistêmica desta lente, é necessário analisar de que forma ela é interpretada. Segundo Marconi et al (2007), uma das fases mais importantes de uma pesquisa é a coleta e análise dos dados. Essa fase ajuda a analisar cada um dos pontos e fenômenos que estão ocorrendo em uma organização, sendo o ponto de partida para a elaboração e execução de um trabalho de cunho científico. Sendo assim, pode-se afirmar que problemas nesta fase, irão dificultar a sequencia de uma pesquisa.

Devido à temática, Lean para Desenvolvimento de *Software* em conjunto com Ágil, ser pouco recorrente na literatura, autores como Creswell (2003) afirmam que estudos de caso exploratórios são mais eficientes que outras abordagens, quando procura-se novas ideias e melhor compreenssão do objeto de pesquisa. Cauchick 2007 afirma que estudos de caso, devido à sua natureza de investigação empírica, são efetivos quando o contexto da pesquisa em questão ainda não está claro para o pesquisador. “Seu objetivo é aprofundar o conhecimento acerca de um problema não suficientemente definido.”

(CAUCHICK, 2003; MATAR, 1996). Para garantir um melhor acompanhamento e conhecimento dos processos orientados dentro dos projetos de desenvolvimento de *software*, observa-se maior aproveitamento com os estudos de caso (YIN, 2005), entretanto, é necessário uma condução metodologicamente adequada, cuja proverá para a pesquisa, maior confiabiliade e validade nos resultados alcançados.

Assim sendo, a lente em questão objetiva mensurar e analisar os processos metodológicos utilizados para análise e coleta dos dados em cada um dos artigos que constituem o portfólio bibliográfico. Em específico aqueles que utilizaram de estudos de caso bem como as particularidades que os classificam. De acordo com Cauchick 2003, os estudos de caso podem ser classificados segundo o seu conteúdo (exploratórios, explanatórios ou descritivos) e quantidade de casos (único ou múltiplo).

Com relação aos 24 artigos que compõe o respectivo portfólio bibliográfico, a maior parte deles (42%) correspondem à estudos de caso. No total são 10 trabalhos que utilizaram da referente prática (Baskerville et al., 2006; Fogelström et al., 2010; Hansson et al., 2006; Middleton et al., 2012; Mishra et al., 2009; Petersen et al., 2009; Petersen et al., 2010; Torrecillas-Salinas et al., 2015; Suomalainem et al. 2015; Sharp et al., 2008), dentre eles, 6 tratam-se de estudos de um único caso e os demais classificados como múltiplos casos. Mishra (2009) recomenda a aplicação de estudos de caso quando pouco se sabe sobre o contexto de pesquisa, principalmente quando se faz necessário uma maior profundindade no assunto, dessa forma, a utilização dessa metodologia, provém resultados relevantes, levando em consideração os poucos resultados práticos, encontrados na literatura, sobre a temática Lean *Software Development* integrado aos princípios ágeis para desenvolvimento de *software*. (Conboy et al., 2008; Mishra et al. 2009; Petersen et al. 2010). A Figura 11 ilustra as metodologias aplicadas em cada artigo que compõe o portfólio bibliográfico, bem como a classificação, quanto ao número de casos, dos trabalhos que utilizaram de estudos de caso.

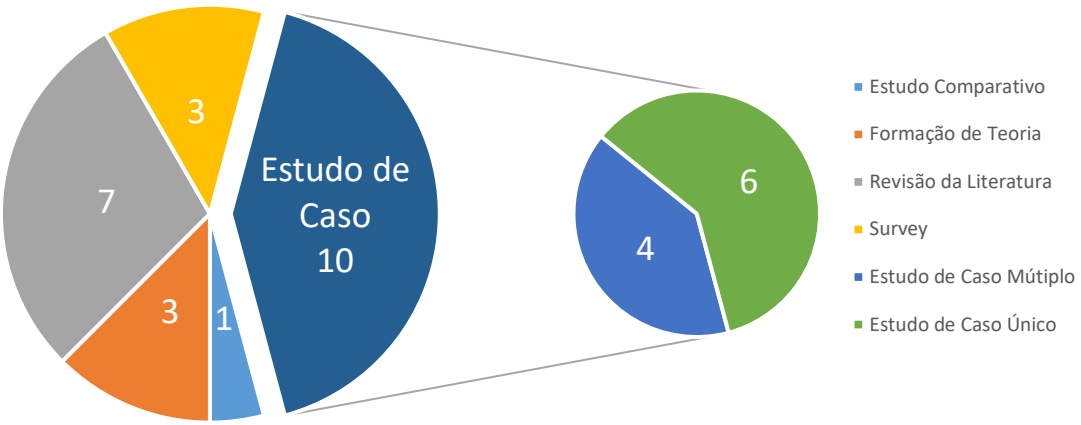

Figura 11 - Metodologias do portfólio bibliográfico - Fonte: Dados da pesquisa

Ainda sobre a forma de análise e coleta dos dados, Petersen (2011) enfatiza a importancia na confiabilidade dos dados coletados em um estudo de caso. De acordo com o autor, a má interpretação ou coleta das informações, podem comprometer os resultados de uma pesquisa científica. Dessa forma, é ressaltado a importancia da triangulação dos dados, ou seja, consultar diferentes fontes de dados, como documentação, entrevistas e observações direta do fluxo de trabalho do(s) caso(s) em estudo. Com relação aos artigos do portfólio bibliográfico que utilizaram de estudos de caso, 50% deles coletaram os dados de diferentes fontes de dados: observação direta ou indireta, análise de documentos e entrevistas (Fogelström et al., 2010; Middleton et al., 2012; Petersen et al., 2010; Suomalainem et al. 2015; Sharp et al., 2008). A Figura 12, ilustra as formas de coletas de dados de cada um dos estudos de caso.

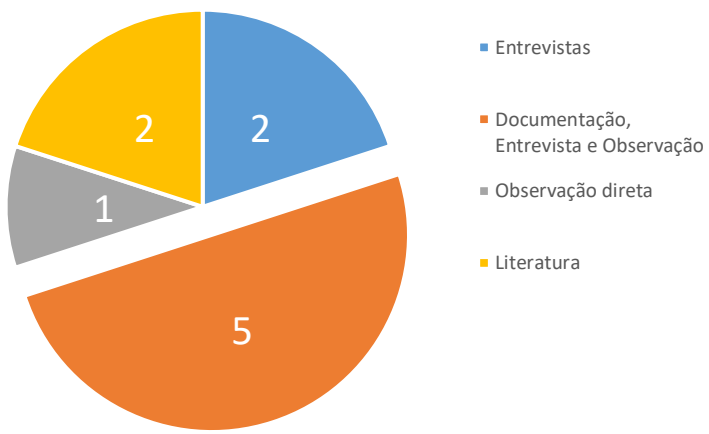

Figura 12 - Coleta de dados dos estudos de caso. Fonte: Dados da pesquisa

Como oportunidade de pesquisa para esta lente, ressalta-se a importância que há no processo de coleta de dados, bem como a necessidade da estruturação de uma pesquisa, principalmente quando o contexto da pesquisa ainda não é de total conhecimento do pesquisados. A escolha certa da metodologia e a forma coerente de análise e coleta dos dados, fazem parte do início de uma pesquisa bem estruturada e relevante.

2.1.4.2.2 Lente 2 – Escopo do Projeto

Muitos representantes do movimento ágil de desenvolvimento afirmam que suas formas de desenvolvimento de *software* são mais apropriadas em projetos de desenvolvimento curtos, além de projetos onde a equipe conhece e tem domínio sobre o contexto do problema (Beck, 2000; Cockburn, 2002). Embora os métodos ágeis sejam extremamente eficazes nesses contextos, (Mishra, 2009), os produtos de *softwares* grandes e complexos, exigem uma disciplina sistemática com processos adicionais para garantir o sucesso das suas entregas. A aplicabilidade de métodos ágeis às grandes organizações e projetos, é muitas vezes considerada um desafio, devido à complexidade do domínio do problema, cujo, muitas vezes está além da experiência do time de desenvolvimento.

Mishra (2009) destaca alguns desafios quanto à aplicação das práticas ágeis no contexto de desenvolvimento de *softwares* de grande porte: (i) desafios com relação à realização de testes contínuos; (ii) grande esforço relacionado à manutenção do produto; (iii) necessidade de coordenação da equipe de desenvolvimento; (iv) dependência entre os módulos à serem desenvolvidos; (v) necessidade de gerenciamento de requisitos devido à processos complexos; (vi) dificuldade em manter a lista de prioridades de desenvolvimento; (vii) dependência entre as atividades, o que gera uma espera entre os processos e (viii) falta de testes exploratórios independentes do projeto. Outro importante desafio destacado pelo autor, relacionando a aplicabilidade de métodos ágeis em projetos de grande porte, é o fato de que, em geral, um sistema de larga escala não é desenvolvido para um único cliente, o que faz com que haja um time extremamente conhecedor do domínio da aplicação, cujo deverá interagir com o maior número de possíveis cliente para definir cada uma das funcionalidades do produto final.

Mesmo tendo em vista todos esses desafios a cerca da adoção de práticas ágeis no desenvolvimento de *software* de grande porte, os mais diferentes projetos, e empresas dos mais variados portes mostram-se interessadas na adoção dessas práticas com o intuito demelhor satisfazer seus clientes, realizar as entregas em um tempo menor e entregar produtos de mais qualidade. Dessa forma, a lente em questão, objetiva analisar o portfólio bibliográfico, e identificar, os trabalhos que se referem aos desafios da implementação de práticas ágeis em produtos ou projetos de grande porte, bem como os principais resultados e análises observadas pelos respectivos, a fim de melhor compreender a adoção de práticas ágeis no contexto de desenvolvimento de *software* de grande porte e utilizar dos ganhos obtidos a partir da sua implementação (FOGELSTRÖM ET AL., 2010; HANSSON ET AL., 2006; MIDDLETON ET AL., 2012; MISHRA ET AL., 2009; PETERSEN ET AL., 2009; PETERSEN ET AL., 2010).

Dentre os artigos que compõe o portfólio, 14 trabalhos destacam os desafios da utilização dos conceitos e práticas ágeis no desenvolvimento de software de grande porte (DA SILVA ET AL., 2011; FOGELSTROM ET AL., 2010; HANSSON ET AL., 2006; KETTUNEN, P, 2009; MIDDLETON ET AL., 2012; ; MISHRA ET AL., 2009; PETERSEN ET AL., 2010; PETERSEN ET AL., 2009; POPPENDIECK ET AL., 2012; ZHI ET AL. 2015; TORRECILLAS-SALINAS ET AL., 2015; SUOMALAINEM ET AL. 2015; WANG, X. ET. AL, 2012; PETERSEN ET AL., 2010). O Quadro 4 sumariza as principais dificuldades destacadas nos trabalhos anteriormente destacados.

Quadro 4 - Resumo das dificuldades identificadas no desenvolvimento de software de grande porte

Dificuldade	Mishra	Da Silva	Fogelstro	Hansson	Kettunem	Middleton	Pernstal	Petersen	Petersen	Poppendi	Zhi 2015	Torrecilas	Suomalai	Wang
Execução de testes contínuos e exploratórios (A)	x					x								

- continua

- continuação

Custo com manutenção do produto a longo prazo (B)	x		x				x		x				
sCoordenação de multiplos times de desenvolvimento (C)	x				x		x	x				x	
Dependência entre módulos do sistema (D)	x		x				x			x	x		x
Gestão de requisitos (E)	x		x	x			x		x		x		
Priorização e distribuição de atividades (F)	x				x		x	x			x	x	x	x
Dependência entre atividades do ciclo de desenvolvimento (G)	x	x			x		x				x			
Comunicação entre os times de desenvolvimento (H)				x	x		x	x						
Desenvolvimento de software paralelo (I)			x		x		x	x			x			x

Quando cada um dos autores mencionam as dificuldades a cerca da implementação do conceito ágil para desenvolvimento de *softwares* de grande porte, todas as dificuldades são citadas por no mínimo dois trabalhos. A Figura 13 ilustra cada uma das dificuldades mencionadas nos respectivos trabalhos, e a quantidade de vezes que cada uma foi reportada.

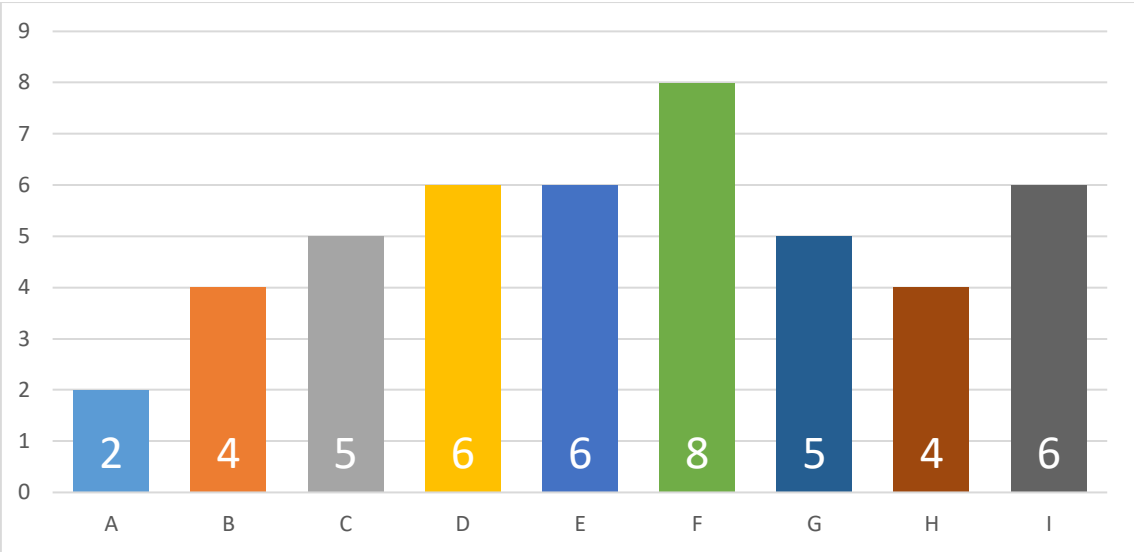

Figura 13 - Dificuldades na implementação ágil no desenvolvimento de softwares de grande porte

A dificuldade “Priorização e distribuição das atividades” (F) foi a que mais vezes apareceu nos trabalhos do portfólio. Torrecilla – Salinas 2015 destaca que a não priorização de atividades e novas funcionalidades, ocasiona no desenvolvimento de funções que nunca ou raramente serão utilizadas, o que representa recursos investidos que dificilmente retornarão à organização. Os autores vão além, e apresentam um estudo que abrange cerca de 2000 projetos realizados por 1000 empresas desenvolvedoras de *softwares.* A pesquisa relata que mais da metade das funcionalidades desenvolvidas em um projeto são quase nunca utilizadas. Com o propósito de resolver a respectiva problemática, o estudo propõe a concepção de um modelo de priorização de demandas, cuja contempla o retorno sobre o investimento (ROI) no desenvolvimento de cada funcionalidade de um projeto. Suomalainem 2015, afirma que, um dos fatores de sucesso no desenvolvimento de *software* é saber desenvolver a funcionalidade certa na hora certa.

Além dessa, outras dificuldades também são repetidamente mencionadas e consecutivamente merecem também atenção. Sendo assim,

essa lente destaca a oportunidade de estudos em cada um dos termos apresentados no Quadro 4, objetivando, às organizações, desfrutar das vantagens provindas a partir da utilização dos preceitos ágeis, independente do contexto e escopo dos projetos ou produtos.

2.1.4.2.3 Lente 3 - Aplicação Prática dos Estudos

Para considerar a análise da respectiva lente, faz-se necessário entender a sua importância. Para entender o tema em questão, é preciso uma breve contextualização sobre senso comum e ciência. Segundo Cauchick et al. 2010, o senso comum também conhecido como “sabedoria popular”, é obtido a partir de experiências práticas, costumes e hábitos, sendo que a ciência se diferencia do senso comum, principalmente, pela utilização de um rigor metodológico. “Geralmente, o termo: rigor metodológico, é utilizado para demarcar a diferença entre a ciência e senso comum”.

Principalmente na Engenharia de Produção, o senso comum e a ciência têm uma forte interação, devido ao fato de que muitos problemas tiveram sua solução primeiramente na prática (senso comum) e depois tornaram interessante para a comunidade científica a partir dos seus resultados obtidos. De acordo com Cauchick et al. 2010, um dos maiores exemplos disso é o Sistema Toyota de Produção.

Assim sendo, uma análise foi feita sobre cada um dos trabalhos que constituem o portfólio bibliográfico, a fim de identificar quais deles apresentam alguma aplicação prática, seja ela estudos de caso práticos, entrevistas com especialistas, entre outras formas de abordagem empírica. Assim sendo, dos 24 trabalhos do portfólio, 10 artigos apresentaram alguma forma de aplicação prática (TORRECILLA-SALINAS ET AL., 2015; HANSSON, C. ET AL., 2005; MIDDLETON ET AL., 2012; MISHRA ET AL., 2009; PETERSEN ET AL., 2009; PETERSEN ET AL., 2010; FOGELSTROM ET AL., 2010; KETTUNEN, P, 2009; WANG, X. ET. AL, 2012; SUOMALAINEM ET AL. 2015).

Dos 10 trabalhos que possuem algum tipo de aplicação prática, 8 deles destacam a aplicação de entrevistas presenciais semiestruturadas com especialistas. (TORRECILLA-SALINAS ET AL., 2015; HANSSON, C. ET AL., 2005; MIDDLETON ET AL., 2012; PETERSEN ET AL., 2010; FOGELSTROM ET

AL., 2010; KETTUNEN, P, 2009; WANG, X. ET. AL, 2012; SUOMALAINEM ET AL. 2015). Torrecilla-Salinas et al., 2015, em seu estudo, concebeu um modelo para estimativa de *software* baseado em retorno de investimento à organização desenvolvedora. Os dados utilizados para a construção do modelo, foram coletados da Literatura, entretanto, objetivando melhorar ainda mais a proposta da pesquisa, o respectivo modelo foi posto em prática em uma empresa da região. Os dados coletados a partir da aplicação empírica, foram utilizados para melhorias e adequações no modelo proposto.

Os outros 2 trabalhos que apresentaram uma aplicação prática, entretanto não destacaram a entrevista como sendo essa (Mishra et al., 2009; Petersen et al., 2009), apontam duas outras abordagem empírica: Observação direta e Validação com especialistas. Mishra et al., 2009 utilizou da abordagem de observação direta em diferentes projetos de desenvolvimento de *softwares* complexos, com o intuito de analisar como é a adoção das metodologias ágeis nesses contextos. Petersen et al. 2009, utilizou da literatura para conceber um *framework* para melhoria contínua em processos de desenvolvimento de *software*. Objetivando coletar dados relevantes para melhorar o seu modelo, o *framework* foi validado com profissionais especialistas da temática Lean e Ágil para desenvolvimento de *software.*

A importância dessa lente é notória tendo em vista que, grande parte dos artigos que compões o portfólio bibliográfico destacaram a sua contribuição prática, cuja, na área de Engenharia de Produção, como já destacado pelo autor Cauchick 2010, é uma medida de alta contribuição e relevância para uma pesquisa dessa área.

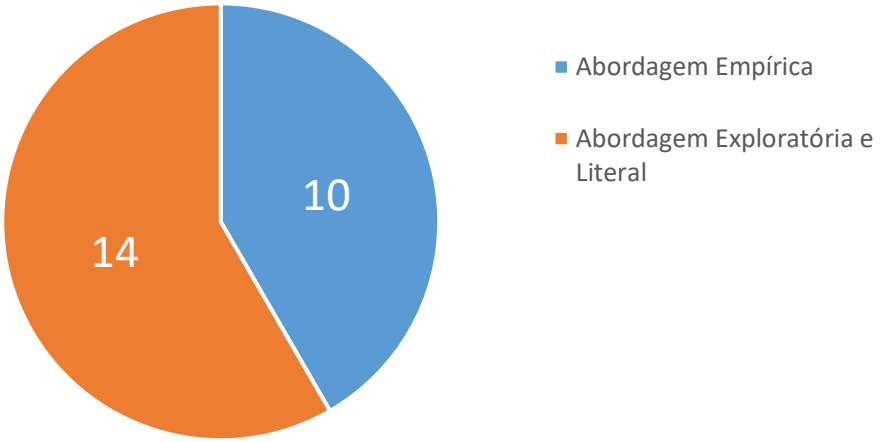

Figura 14 - Lente 3 - Aplicação Prática dos Estudos. Fonte: Dados da Pesquisa

2.1.4.2.4 Lente 4 - Filosofia Enxuta (*Lean*)

Nos últimos anos, o desenvolvimento de *software* tem evoluído principalmente em torno do processo de fabricação, aplicando práticas ágeis a fim de aperfeiçoá-lo (DYBA E DINGSOYR, 2008). Entretanto, as abordagens ágeis têm sido cada vez mais questionadas quando o escopo do projeto é variado (Biffl, et al., 2005). Com o propósito de melhorar os métodos e processos de desenvolvimento de *software*, as empresas começam a observar a abordagem Lean da Manufatura. A aplicação dos conceitos Lean para o contexto de desenvolvimento de *software* não é tão simples uma vez que a sua aplicação depende muito do contexto e dos objetivos do ambiente a ser introduzido (POPPENDIECK E POPPENDIECK, 2007). Contudo, através do ajuste das práticas a se utilizar, melhorias substanciais podem ser atingidas no processo de fabricação (WANG, 2012).

O conceito Lean para desenvolvimento de *software* é um termo novo e não muito difundido na literatura, porém destaques e resultados interessantes na prática já foram e estão sendo observados em vários trabalhos e publicações de revistas e congressos científicos (WANG, 2012). Dessa forma, a lente em questão, objetiva analisar, dentre os 24 artigos selecionados, quais destacam a filosofia Lean nas práticas de desenvolvimento de *softwares* e respectivos estudos científicos. De todos os trabalhos selecionados 61% deles (14 trabalhos) destacam ou citam a filosofia Lean na sua respectiva pesquisa, sendo esses: Torrecilla-Salinas, C. J et. al 2015, Ebert, C. et.al 2012, Da Silva, I. F. et. al 2011,

Petersen, K. et. al 2011, Petersen, K. et. al 2010, Poppendieck, M. et. al 2012, Fogelstrom, N. D. et. al 2010, Middleton, P. et. al 2012, Kettunen, P. et. al 2009, Nerur, S. et. al 2005, Dybå, T. et. al 2008, Chow, T. et. al 2008, Van Waardenburg, G. et. al 2013 e Wang, X. et. al 2012.

Dentre os 14 trabalhos, 6 deles destacam *Lean Software Development*, como uma metodologia ágil, por possuir características semelhantes àquelas já encontradas em modelos ágeis mais conhecidos, como Scrum, XP, entre outros.

Destacamos outros 2 trabalhos dos seguintes autores: Ebert, C. 2012 e Poppendieck, M. 2012. Trata-se de estudos científicos e teóricos, cujos descrevem o conceito do *Lean Software Development*. Poppendieck (2012) descreve o LSD não como uma metodologia da engenharia de *software* tradicional e sim como uma síntese de práticas dos princípios e da filosofia Lean para a construção de *software*.

Os outros 7 trabalhos que mencionam a filosofia Lean, tratam-na como uma evolução dos processos ágeis de desenvolvimento de *software*, onde as lacunas apresentadas pelas práticas ágeis poderiam ser resolvidas com a aplicação de algumas ferramentas do pensamento enxuto. Entretanto, os autores destacam que existem muitos poucos estudos sobre a sua adoção prática, como é o caso do artigo de Wang, X. et. al 2012. O autor destaca que o LSD é considerado como uma “nova era” dos processos de desenvolvimento de software, à sua implantação, tornaria possível a adoção dos preceitos enxutos em projetos ou produtos complexos.

Em seu trabalho, Wang et al. 2012, divide em 6 categorias, a aplicação dos princípios e práticas Lean no desenvolvimento ágil de *software*: (i) Combinação não proposital entre ágil e Lean; (ii) Abordagem Lean aplicada em areas relacionadas ao contexto do negócio; (iii) Utilização do Lean como facilitador para a transição dos processos ágeis; (iv) Uso de elementos Lean como melhoria dos processos ágeis; (v) Do ágil para o Lean; (vi) Sincronismo entre Lean e Ágil.

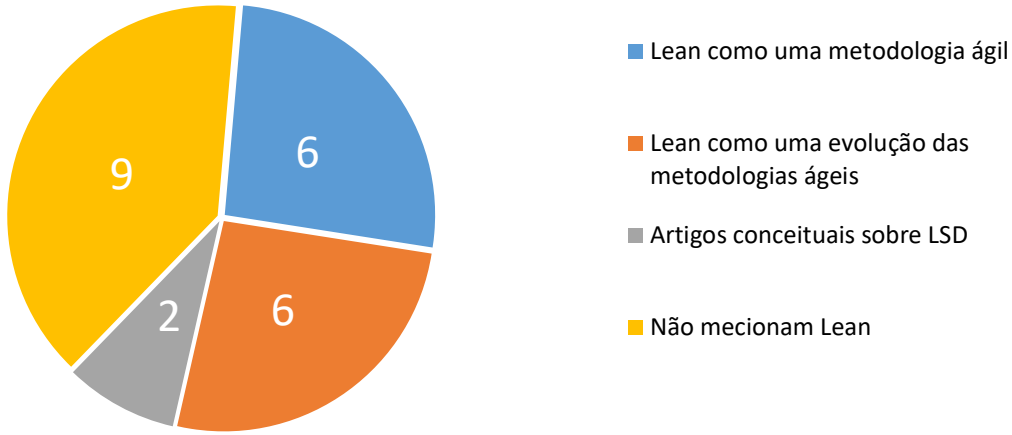

Figura 15 - Lente 4 - Filosofia Enxuta (Lean)

O Quadro 5 ilustra os cinco princípios da filosofia *Lean* e os relaciona com estudos que que expõem *Lean* para desenvolvimento de *software* (LSD) como uma metodologia independente dos conceitos ágeis ou filosofia de gestão do processo de fabricação.

Destaca-se o princípio de identificação de atividades que agregam valor no ponto de vista do cliente, para todos os estudos, o princípio em questão foi incluído pelos autores. Wang 2012, destaca em seu trabalho à importancia em se desenvolver funcionalidades que tenham valor para o cliente final (usuário). Dyba 2008 destaca que o princípio de entregar valor para o cliente, já comumente conhecido dos preceitos ágeis, é o princípio básico de qualquer metodologia ágil de desenvolvimento de *software*.

Quadro 5 - Princípios Lean. Fonte: Dados da Pesquisa

Princípios *Lean*	Wang 2012	Torrecilla-Salinas 2015	Ebert 2012	Da Silva 2011	Fogelstrom 2010	Middleton 2012	Nerur 2005	Chow 2008	Petersen 2010	Petersen 2011	Poppendieck 2012	Waardenburg 2013	Kettunen 2009	Dyba 2008
(i) identificar valor para o cliente;	x	x	x	x	x	x	x	x	x	x	x	x	x	X

- continua

- continuação

(ii) identificar as etapas do fluxo de valor;			x	x		x		x	x	x	x	x	x	
(iii) criar um fluxo sequencial para cada produto;			x			x		x	x	x	x	x	x	x
(iv) estabilizar e fazer com que o fluxo de valor seja puxado pelo cliente;		x	x			x		x	x	x	x	x	x	
(v) garantir a contínua melhoria de todo o processo.					x	x		x			x	x	x	

2.1.5 Lacunas de Pesquisa

O presente trabalho tem interesse em auxiliar a construção de conhecimento sobre Desenvolvimento *Lean* de *Software,* auxiliando, assim, em pesquisas e produções futuras sobre este tema. Levando em conta o interesse dos autores, no entendimento da temática LSD para diferentes contextos, e posteriormente identificá-la como seriam as possíveis soluções para as lacunas encontradas na implantação de metodologias ágeis para desenvolvimento de *software*, têm-se a filiação teórica dos autores deste trabalho.

A análise das lacunas identificadas na pesquisa em questão, pode prover resultados relevantes para a sequência de próximos trabalhos. A partir da observação de cada uma delas, identificou-se uma série de lacunas cabíveis de próximos estudos:

- (i) Redução de desperdício com superprodução de funcionalidades: Um dos desperdícios mais relevantes identificados e destacados na revisão bibliográfica é a análise e construção de funcionalidades que são pouco ou raramente usadas pelos seus clientes. Trata-se do desperdício de

superprodução. Poucos estudos foram encontrados objetivando a redução de tal desperdício;

- (ii) Priorização de demandas: Outra oportunidade para a realização de próximos estudos é a priorização de incrementos em um *software* de grande porte. A má realização dessa atividade ocasiona, além do incremento de funcionalidades não desejadas, o desenvolvimento inacabado de incrementos. Durante a pesquisa, estudos sobre práticas ágeis de priorização de demandas foram identificadas, no entanto, nenhuma delas indicam as particularidades de sistemas de grande porte;
- (iii) Gestão de atividades de desenvolvimento: O desenvolvimento de *software* de grande porte possui particularidades inerentes, principalmente, ao processo de incremento de novas funcionalidades. Esse processo necessita de uma gestão das atividades de desenvolvimento, objetivando ser mais efetivo e reduzir os desperdícios ocasionados no processo de produção. Pesquisas sobre Kanban aplicado ao desenvolvimento de *software* foram encontrados, no entanto, nenhum deles com o propósito de identificar e reduzir tais desperdícios.

Sendo assim, a literatura possui grandes oportunidades para trabalhos futuros a cerca deste tema, considerando especialmente as questões de aplicação do conhecimento teórico em ambiente empresarial, juntamente com a documentação necessária para o registro do mesmo. Como trabalho futuro, recomenda-se estudos empíricos utilizando as práticas descritas dentro do contexto LSD.

2.1.6 Considerações finais

O processo empregado para a formação da análise bibliométrica e análise sistêmica compôs um portfólio bibliográfico de 24 artigos que permitiu analisar: i) o grau de relevância dos periódicos, apresentando o *Journal of System and Software* com maior destaque com relação aos demais, bem como palavras-chave *(Agile, Software Development, Leagile, Scrum e Lean Software Development*) e autores mais relevantes (Petersen K. e Wohlin, C., P.,); ii) e

através da filiação teórica foi possível observar as principais características entre os trabalhos analisados.

Os resultados obtidos ao final da pesquisa, levando em consideração o rigor metodológico da pesquisa, reconhecem a eficiência da metodologia *Proknow-C*, como instrumento de intervenção e auxílio da construção do conhecimento.

Como limitação do presente trabalho, destaca-se a não inclusão de trabalhos que utilizam de outras metodologias de desenvolvimento de *software* diferente das ágeis. Vale ressaltar que elas podem apresentar resultados importantes para o contexto da problemática principal da pesquisa em questão.

Em análise a todo o contexto estudado neste trabalho, é possível afirmar que a grande lacuna presente no meio científico para o tema Desenvolvimento *Lean* de *Software* se apoia, justamente, na deficiência na apresentação de conhecimento acerca de estudos práticos aplicadas na gestão e desenvolvimento de *software*.

3.2 REDUZINDO O DESPERDÍCIO DA SUPERPRODUÇÃO NO DESENVOLVIMENTO DE *SOFTWARE* DE GRANDE PORTE POR MEIO DA PRIORIZAÇÃO DE DEMANDA

A segunda etapa que irá compor a dissertação em questão, trata-se de um estudo de caso a ser realizado em uma empresa desenvolvedora de softwares de grande porte. O respectivo trabalho será submetido ao periódico IEE América Latina, cujo, de acordo com a classificação do ano de 2004 do Qualis, qualificou-o como B2 na área de Engenharias III. O objetivo do trabalho em questão será o de cumprir com alguns dos escopos específicos destacados na seção 1.3.2, segue abaixo:

- Priorizar demandas em uma fábrica de desenvolvimento de *software* de grande porte, reduzindo o desperdício com produção de funcionalidades não utilizadas;

3.2.1 Introdução

Quando se fala em sistemas de grande porte, Mishra (2011) refere-se à sistemas complexos que detém de informações que auxiliam no processo de gestão de pequenas, médias e grandes empresas. Perry (2001) destaca quatro principais características que definem um sistema de grande porte: (i) Constante evolução e manutenção do sistema; (ii) Alta complexidade; (iii) Múltiplas dimensões do sistema (Modularidade) e (iv) Distribuição do conhecimento entre os desenvolvedores.

A necessidade de lidar com a constante evolução e manutenção de um *software* complexo, gera uma série de esforços e estudos com o intuito de adotar processos ágeis no desenvolvimento de *software* de grande porte, mitigando assim os riscos em termos de tempo de colocação no mercado. Além disso, o quanto antes um cliente recebera sua versão do *software*, mais cedo, a empresa desenvolvedora irá obter o retorno sobre o investimento realizado no respectivo incremento da aplicação. Entretanto, Mishra (2011), descreve alguns principais desafios na utilização das práticas ágeis no desenvolvimento de softwares de grande porte, são esses: (i) Custo com a realização de testes contínuos; (ii) Esforço de manutenção com um aumento no número de versões; (iii) Despesas de gerenciamento devido à necessidade de coordenação de equipes diversas; (iv) Dependências entre artefactos de desenvolvimento; (v) Tempo gasto com gerência de requisitos devido à processos complexos; (vi) Listas de prioridades de requisitos difíceis de criar e manter; (vii) Tempos de espera entre as atividades do processo, especialmente entre o desenvolvimento e a especificação de requisitos; (viii) Redução da cobertura de teste devido à escassez de projetos e falta de testes independentes; (ix) Aumento do esforço de gerenciamento de configuração.

Dentre as questões levantadas durante o projeto de desenvolvimento de um *software*, Salinas (2014), destaca as seguintes: Quanto vai custar? Quando ele estará pronto? Quanto esforço será investido? O valor investido será retornado? Quais funcionalidades devem ser desenvolvidas? Respostas às perguntas acima destacadas, são objetos para a concepção de um produto de software de sucesso, entretanto, a omissão à algumas delas, podem causar impactos negativos para o produto final. Se não houver um planejamento acerca

das funcionalidades à serem desenvolvidas em um produto de *software*, recursos entregues raramente ou nunca serão utilizados. Esse resultado representa investimentos feitos que dificilmente serão retornados à organização desenvolvedora, ou seja, desperdício de tempo e recurso (SALINAS, 2014; WANG, 2012; MISHRA, 2011).

Uma pesquisa realizada pelo Standish Group (2010), cuja abrange cerca de 2000 projetos de desenvolvimento de *software*, realizados por cerca de 1000 organizações, destaca que cerca de 64% das funcionalidades desenvolvidas em um sistema de informação nunca ou quase nunca são usadas. A Figura 1 ilustra o resultado da pesquisa.

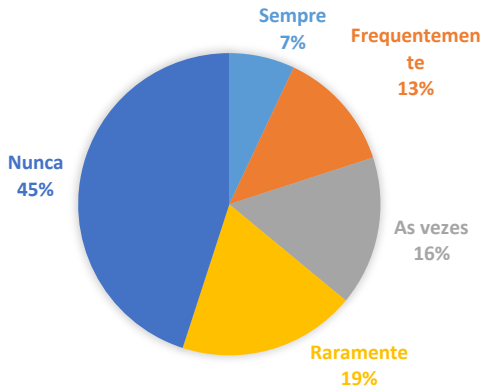

Figura 1 - Standish Group. Fonte: Adaptado de Chaos Report (2010)

Sallinas (2014), em seu estudo sobre estimativa e planejamento ágil de desenvolvimento de *software*, propôs uma nova forma de priorizar e gerenciar o desenvolvimento, baseado em uma perspectiva de valor de negócio. Os resultados apresentados em sua pesquisa, demonstraram que a abordagem é muito apropriada para projetos de desenvolvimento *web*, entretanto, o autor não descarta a adaptação do modelo para projetos de outras modalidades.

Diferentes empresas desenvolvedoras de *software*, utilizam distintas formas para priorizar as demandas de cada um de seus produtos. Seja utilizando alguma técnica de priorização, analisando recomendações do mercado, implementando exigências legais, sugestões de patrocinadores e influenciadores, entre outras. A escolha errada daquilo que será desenvolvido gera um grave desperdício no processo de produção de software, o desperdício de superprodução. (POPPENDIECK, 2003; MISHRA, 2011; PERRY, 2001). O problema de priorização de demandas de desenvolvimento de *software*,

destacado por Salinas (2014), é ainda agravado quando o produto em questão é tratado como um *software* de grande porte, principalmente devido à constante evolução e manutenção sofrida durante o seu ciclo de vida.

Os desafios na adoção de práticas ágeis no desenvolvimento de *softwares* de grande porte, destacados por Mishra (2011), fizeram surgir uma série de estudos com propostas de soluções para algumas das lacunas existentes. Poppendieck (2000) propôs uma nova filosofia de desenvolvimento de *software*, chamada: *Lean Software Development* (LSD). A respectiva metodologia, se propõe à traduzir os conceitos da filosofia *Lean Thinking,* para o ambiente de desenvolvimento de *software*. A filosofia é mapeada em torno de 5 principais princípios: (i) Identificar valor para o cliente; (ii) Identificar todas as etapas do fluxo de valor de cada produto, eliminado possíveis etapas que não agregam valor; (iii) Criar um fluxo sequencial para cada produto; (iv) Estabilizar e fazer com que o fluxo de valor seja puxado pelo cliente, a partir das suas reais necessidades; e (v) Garantir a contínua melhoria de todo o processo, a fim de atingir o estado da perfeição, sem nenhum tipo de desperdício.

Além dos cinco princípios *Lean* anteriormente apresentados, um dos fundamentos do pensamento enxuto é a contínua eliminação de atividades desnecessárias, ou seja, eliminar os desperdícios, usando os esforços em atividades que realmente agregam valor para o cliente Wornack (1991). Desperdício é qualquer atividade executada que não agrega valor ao produto final. Poppendieck (2003) traduz os sete desperdícios da filosofia *Lean* para o contexto de desenvolvimento de *software:* (i) Superprodução: Funcionalidades extras; (ii) Transporte: Transferência de controle; (iii) Estoques: Trabalhos inacabados; (iv) Defeitos: Defeitos em software; (v) Processamento adicional: Reaprendizagem; (vi) Movimentação: Troca de tarefas; (vii) Esperas: Atraso para inciar atividades (POPPENDIECK 2003).

Levando em consideração os assuntos expostos, surgiu então o principal problema desta pesquisa: Qual nova funcionalidade, de um *software* de grande porte, deverá ter o seu desenvolvimento priorizado para que os desperdícios com superprodução sejam minimizados? Para responder à essa pergunta, o presente trabalho tem como objetivo realizar a priorização de demandas em uma fábrica de desenvolvimento de *software* de grande porte,

utilizando métodos de análise multicritério, visado com isso reduzir o desperdício com superprodução.

Esse trabalho busca contribuir com uma forma de priorização de demandas de desenvolvimento de *software* de grande, sendo que críticas e melhorias na priorização poderão ser realizadas. Um estudo de caso em uma empresa desenvolvedora de *softwares* de grande porte foi realizado, coletando métricas sobre a forma de priorização das demandas de desenvolvimento. A próxima seção do trabalho contextualiza cada uma das temáticas apresentadas. Posteriormente é apresentado a metodologia utilizada para obter os resultados, em seguida, na seção 4, são apresentados os resultados referentes ao estudo de caso e por fim é apresentado as considerações finais e sugestões de novos estudos.

3.2.2 Desenvolvimento ágil de *software* (ASD)

Muito tem se falado acerca da adoção de processos ágeis para o desenvolvimento de *software* em todo o mundo. Em 2005, uma pesquisa realizada em empresas de desenvolvimento *software* dos Estados Unidos e Europa, revelou que, 14% das empresas estavam adotando as práticas ágeis e outros 49% estavam cientes dos seus benefícios e planejavam implantá-la (DINGSOYR; DYBA 2009).

As metodologias ágeis de desenvolvimento de *software,* ou do inglês *Agile Software Development (ASD)*, nasceram a partir de uma reação contra os métodos tradicionais, que eram considerados como incapazes de atender às dinâmicas do mercado, Rodriguez (2013). A essência das práticas ágeis foi primeiramente apresentada na década de 90, por meio do Manifesto para Desenvolvimento Ágil de *Software*. Este é o resultado de um estudo de 17 profissionais de *software* que criaram uma filosofia comum, nomeada como *"Agile"* (MISRHA, et. al 2012). Os valores e princípios expressos nesse manifesto constituíram um *framework* para o desenvolvimento ágil de *software*, que se baseia principalmente nos seguintes valores: (i) Indivíduos e interações, ao invés de processos e ferramentas; (ii) *Software* funcionando, ao invés de compreensão de documentação; (iii) Colaboração com o cliente, ao invés de contratos

burocráticos; e (iv) Rápida resposta às mudanças, ao invés de seguir o planejado.

Quadro 1 - Valores do manifesto ágil. Fonte: Adaptado de Misrha (2012)

Valor	Descrição
Indivíduos e interações	O manifesto ágil defende que o mais importante nas relações profissionais é como elas interagem entre si. Os processos e ferramentas são importantes porém não devem substituir as interações humanas.
Software funcionando	Um *software* funcionando e realizando aquilo que o seu cliente espera, vale muito mais que uma série de documentos e manuais.
Colaboração com o cliente	Colaborar com o cliente significa trazê-lo o mais próximo possível do projeto, torná-lo parte do time de desenvolvimento, envolvê-lo em questões e riscos.
Rápidas respostas às mudanças	As mudanças devem ser bem aceitas e tratadas pelo projeto como oportunidades e não somente ameaças.

É com base nos valores destacados no Quadro 1, que as metodologias ágeis de desenvolvimento de *software*, foram concebidas. Algumas delas são: *Scrum, Extreme Programming* (XP), *Crystal*, FDD.

3.2.2.1 Lacunas do desenvolvimento ágil de *software*

Os autores, (DINGSOYR; DYBA, 2009), fizeram uma revisão sistemática da literatura, com o propósito de esclarecer a eficácia na adoção de algumas das metodologias ágeis no desenvolvimento de *software*, bem como as limitações encontradas em cada uma delas. A pesquisa analisou a adoção de 10 metodologias ágeis nas respectivas fases do desenvolvimento de *software*. O resultado foi que, nenhum dos métodos, compreendem todas as etapas. O *Scrum*, por exemplo, abrange principalmente aspectos relacionados à gestão de projetos.

Além da limitação destacada por (DINGSOYR; DYBA, 2009), a adoção das práticas ágeis para o desenvolvimento de *software*, envolvem outros desafios, principalmente, relacionados ao contexto do produto a ser produzido. Um dos valores necessários para a adoção de práticas ágeis é o envolvimento do cliente durante todo o ciclo de desenvolvimento do *software*. Essa prática é

dificilmente aplicada quando o produto em construção, não é feito para atender um único perfil de cliente, característica comum de um *software* de grande porte (CHAU et. AL, 2003; NERUR et. AL, 2005; MISHRA, 2011; DA SILVA et. AL, 2011).

Outra particularidade de um *software* de grande porte é a maneira que são concebidos. Segundo Pernstal (2013) um sistema de grande porte é normalmente construído em subsistemas que posteriormente serão integrados. A construção desses subsistemas pode ser feita por pessoas e equipes distintas, cuja, em alguns casos, estão separadas geograficamente. Essas condições exigem algumas práticas não tão consolidadas em metodologias ágeis, como: (i) Documentação e gerenciamento de requisito, (ii) Gestão visual do andamento do projeto, (iii) Gestão de recursos, (iv) Métricas do produto e da equipe. (BIFFL et. al, 2005).

Por fim, Mishra (2011), ilustra 9 desafios encontrados no desenvolvimento de *softwares* de grande porte, apresentado no Quadro 2.

Quadro 2 - Desafios no desenvolvimento de *software* de grande porte. Fonte: Adaptado de Mishra (2011)

Identificador	Desafio
1	Custo com a realização de testes contínuos
2	Esforço de manutenção com um aumento no número de versões
3	Despesas de gerenciamento devido à necessidade de coordenação de equipes diversas
4	Dependências entre artefactos de desenvolvimento
5	Tempo gasto com gerência de requisitos devido à processos complexos
6	Listas de prioridades de requisitos difíceis de criar e manter
7	Tempos de espera entre as atividades do processo, especialmente entre o desenvolvimento e a especificação de requisitos
8	Redução da cobertura de teste devido à escassez de projetos e falta de testes independentes
9	Aumento do esforço de gerenciamento de configuração

Apesar das limitações destacadas a partir da aplicação das metodologias ágeis em diferentes contextos, os resultados obtidos com a sua aplicação, são notoriamente melhores quando comparados às práticas tradicionais de desenvolvimento de *software*. As metodologias ágeis estão sendo cada vez mais aplicadas no mercado atual, uma vez que proporcionam menores custos de desenvolvimento, qualidade na entrega do produto final, maior produtividade da equipe de desenvolvimento e consecutivamente maior satisfação do cliente (MISHRA et. al, 2011).

De fato, as práticas provaram ser mais eficientes que as tradicionais, especialmente em projetos de tamanho e escopo moderado, uma vez que oferecem flexibilidade e agilidade que ajudam a produzir um sistema de alta qualidade em um período curto de tempo. (DINGSOYR, et. al, 2009; BIFFL, et. al, 2005; KUPIAINEN et. al, 2014).

3.2.3 Metodologia de desenvolvimento ágil: *SCRUM*

O *Scrum* é um *framework* simples utilizado para organizar e auxiliar as equipes a fazer o trabalho de forma mais produtiva e com alta qualidade. O *Scrum* é focado na geração do valor agregado para o cliente, eliminando o desperdiço e dando prioridade a entrega periódica de *software* funcional. Essa entrega periódica decorre da estrutura iterativa do processo, conformada por períodos de duas a quatro semanas chamados *Sprints*, cada um dos quais se encerra com a entrega de uma ou várias partes do final (SUTHERLAND, 2007).

De acordo com Sutherland (2012), o *Scrum* possui três papéis, quatro cerimônias e três artefatos. O *Product Owner* (PO), é o primeiro papel e tem como, principais responsabilidades:

- Definir os requisitos do produto;
- Decidir as datas de implantações;
- Refinar e Priorizar os requisitos de cada entrega ao cliente;
- Aceitar ou Rejeitar os resultados do trabalho.

Ainda que o *framework* não inclua a participação de um gerente de projeto, o papel de líder da equipe é adjudicado ao *Scrum Master* (S.M), cujo tem como principais responsabilidades:

- Atuar como um facilitador assegurando que a equipe esteja produtiva e funcional;
- Removendo barreiras e blindando a equipe de interferências externas;
- Assegurar que o *framework* seja seguido.

O terceiro papel é a própria equipe de desenvolvimento, que tem por responsabilidade a produção do trabalho. Devido à não necessidade da figura

de e um gerente de projetos, no *Scrum*, a própria equipe é responsável por se organizar e cumprir com os objetivos e metas traçados para cada *Sprint* do projeto.

As cerimônias do Scrum são reuniões que ocorrem durante períodos fixos e previamente acordados entre todos os participantes do projeto. A primeira cerimônia do Scrum, denominada, *Sprint Planning*, ocorre antes do início de cada iteração. É durante essa reunião que são apresentadas as próximas funcionalidades à serem incrementadas ao produto final. Todos os dias durante a *Sprint* acontece a reunião diária de alinhamento conhecida como *Daily Scrum Meeting*. Geralmente feita numa sala sem cadeiras para estimular a objetividade da reunião, durante essa cerimônia, cada membro do time responde as seguintes perguntas: a) O que foi feito ontem; b) O que será feito hoje; e c) Quais os problemas que estão sendo enfrentados. A terceira cerimônia requerida pelo *Scrum* é o *Sprint Review Meeting*, que tem por objetivo revisar o que está sendo entregue ao cliente e analisar o funcionamento da equipe durante o *Sprint*. A última cerimônia do Scrum, chamada *Sprint Retrospective,* ocorre sempre, ao término de cada iteração. Tem como objetivo, fomentar em todos os membros do projeto, a necessidade de melhoria contínua. Durante essa reunião, cada membro, destaca pontos fortes identificados durante a *Sprint* que se passou, e pontos à melhorar para a próxima.

Sobre os artefatos produzidos na metodologia *Scrum*, os seguintes são considerados:

- *Product Backlog:* Corresponde à lista de funcionalidades à serem consideradas durante todo o desenvolvimento de um *software*. O *Product Owner,* é responsável por manter esse artefato sempre priorizado com os próximos itens à serem produzidos.
- *Sprint Backlog:* Contém as funcionalidades que estão sendo produzidas dentro da iteração corrente (*Sprint*).
- *Burndow Chart:* Trata-se de um gráfico que expõe o avanço na execução da produção das funcionalidades ao longo da *sprint*. Serve como status do projeto.

3.2.4 Desenvolvimento *Lean* de *software* (LSD)

Na literatura atual, os conceitos “ágeis” e “*Lean*”, quando aplicados no mesmo contexto de pesquisa, são pouco difundidas, e a utilização desses termos é muitas vezes considerada ambígua e inconsistente (CONBOY, 2009). Alguns autores veem os dois termos como nomes diferentes para tratarem a mesma ocorrência, por exemplo, nos estudos de Kettunen (2008) e de Chow (2008), nenhuma distinção significativa é feita entre as duas práticas. Entretanto, outros estudos, como por exemplo, Wang (2012), Pernstål (2013), destacam a prática *Lean* para desenvolvimento de *software* como uma metodologia independente da engenharia de *software*, utilizando uma composição dos princípios ágeis e da filosofia *Lean* para a construção de *softwares* de forma mais eficaz.

Na abordagem enxuta acoplada às práticas ágeis de desenvolvimento de *software*, Wang (2012) sintetizou uma análise de 30 trabalhos em 6 principais categorias. As categorias relatam as diferentes formas que os respectivos autores descrevem as duas práticas, quando aplicadas em conjunto. A maioria dos trabalhos analisados, treze no total, referenciaram às abordagens como complementares, ou seja, a aplicação de técnicas/ferramentas *Lean* com o propósito de melhorar a utilização das metodologias ágeis de desenvolvimento de *software*.

O conceito enxuto *Lean* foi primeiramente identificado na empresa Toyota em meados dos anos 80. Wornack (1990), através da publicação “*The Machine That Changed The World*”, popularizaram o termo, e criou-se então uma nova filosofia, denominada “*Lean Thinking*”. A filosofia é mapeada em torno de 5 principais princípios: (i) Identificar valor para o cliente; (ii) Identificar todas as etapas do fluxo de valor de cada produto, eliminado possíveis etapas que não agregam valor; (iii) Criar um fluxo sequencial para cada produto; (iv) Estabilizar e fazer com que o fluxo de valor seja puxado pelo cliente, a partir das suas reais necessidades; e (v) Garantir a contínua melhoria de todo o processo, a fim de atingir o estado da perfeição, sem nenhum tipo de desperdício.

Fogelström (2010) destaca que os conceitos e práticas da filosofia *Lean* podem ser expandidas para demais áreas além da indústria automotiva.

Foi então que, no início dos anos 2000, Poppendieck (2003) transferiram os princípios e práticas enxutas para o ambiente de desenvolvimento de software e chamou-lhes de "*Lean Software Development*" (LSD) (POPPENDIECK , 2003). Os anteriores 5 princípios da filosofia *Lean*, foram então traduzidos para 7 princípios do desenvolvimento *Lean* de *software*.

Quadro 3 - Princípios do LSD. Fonte: Adaptado de Poppendieck (2003)

Princípio	Descrição
Eliminar desperdício	Em software, entende-se como desperdício, funcionalidades não usadas.
Integrar qualidade	Inspeção do produto para que os defeitos sejam identificados imediatamente após ocorrerem. Quando um defeito é encontrado, deve ser corrigido imediatamente.
Criar conhecimento	É importante ter um processo de desenvolvimento que encoraje o aprendizado contínuo e sistemático.
Adiar comprometimentos	As tomadas de decisões devem ser planejadas, a fim de tomar decisões irreversíveis no último momento possível.
Entregar rápido	Entregar tão rápido ao ponto de não haver mudanças.
Respeitar as pessoas	É essencial a preocupação com pessoas durante o processo de produção.
Otimizar o todo	Uma organização *Lean* otimiza todo o fluxo de valor, do momento que recebe o pedido até que o *software* seja implantado e a necessidade do cliente atendida.

A aplicabilidade das ferramentas *Lean*, junto ao contexto de desenvolvimento de *software*, se resumem a prover formas de atingir um ou mais dos princípios ilustrados no Quadro 3.

Quando traduzido os desperdícios da filosofia *Lean* para o contexto de desenvolvimento de *softwares,* alguns tópicos ganham maior relevância dependendo da categoria do produto à ser produzido. Destaque para o desperdício de superprodução, cujo problema está fortemente relacionado à sistemas de grande porte, devido à constantes adaptações sofridas pelo produto durante todo o seu ciclo de vida. Priorizar de maneira correta aquilo que deverá compor o *software,* tem o potencial de minimizar a ocorrência de tal desperdício. (POPPENDIECK, 2003; MISHRA et. al, 2011).

3.2.5 Priorização de demandas no desenvolvimento de *software*

Um *software* de grande porte possui algumas particularidades, dentre elas a constante manutenção e evolução do respectivo produto. A partir dessa

propriedade alguns desafios são encontrados para gerenciar a sua evolução. O principal deles é a eliminação do desperdício com novas funcionalidades que são pouco ou nem utilizadas. De acordo com Poppendieck (2003) a maior fonte de desperdício no desenvolvimento de *software* são as funcionalidades adicionais. Somente cerca de 20% das funcionalidades de um programa personalizado são regularmente usadas. Funcionalidades extras em um produto de *software* é referente ao desperdício de superprodução. Se a superprodução for considerada o pior dos 7 desperdícios da produção, uma funcionalidade sem valor para o cliente é a pior das perdas para desenvolvimento de *software*. Poppendieck (2003) afirma que funcionalidades adicionais em um *software* acrescentam complexidades ao código, elevando o custo com manutenção e reduzindo a vida útil do programa.

A causa, para o desenvolvimento de funcionalidades não usadas em um *software* de grande porte, está relacionada à má análise e priorização das respectivas demandas. Em gestão de projetos de *software* a priorização e organização de demandas está relacionada às respectivas estimativas, ou seja, de acordo com o tamanho e a relevância, a sua prioridade altera (SALINAS, 2014; MISHRA, 2011).

Muitos modelos de estimativa de funcionalidades de um *software* tem sido propostos durante os últimos anos. Os mesmos são classificados em duas principais categorias: Modelos baseados em algoritmos e modelos não baseados em algoritmos (SALINAS, 2014). Modelos baseados em algoritmos utilizam abordagens matemáticas para calcular o esforço e, consecutivamente, realizar a priorização das demandas. Os modelos não baseados em algoritmos são geralmente mais adequados às práticas ágeis. Estes são mais facilmente adaptáveis às mudanças e possibilitam entregar funcionalidades de valor aos clientes de maneira mais rápida e eficiente. Os principais modelos de estimativa em projetos ágeis são ilustrados nas próximas seções.

3.2.5.1 Jogo de planejamento

É uma técnica proposta pela metodologia *Extreme Programming* (XP), o seu nome original vem de *planning games*. O modelo pressupõe que os clientes têm informações suficientes sobre cada nova funcionalidade a ser

desenvolvida, e o time de desenvolvimento tem a maior parte das informações sobre como implementar cada nova funcionalidade.

Os desenvolvedores estimam a complexidade de desenvolvimento de cada nova funcionalidade, e os clientes priorizam-nas de acordo com a importância relativa de cada uma, perante os demais recursos. Essas duas etapas são repetidas até que todas elas estejam organizadas (MCDAID, 2008; SALINAS, 2014).

3.2.5.2 *Planning poker*

Inicialmente criada para planejar o trabalho de uma pequena parte de um projeto maior, a respectiva técnica foi adaptada para realizar estimativas a nível de projetos. Cada membro da equipe tem um baralho de cartas com um subconjunto de valores (por exemplo 1, 2, 3, 5 e 8) os quais representam pontos à serem atribuídos à cada nova funcionalidade.

O cliente, ou o seu representante, explica cada uma das novas funcionalidades, e após todos os membros terem informações suficientes para estimá-las, cada um mostra uma carta com a sua atribuição de complexidade para o desenvolvimento. Caso haja divergência entre as opiniões a equipe discute os motivos e uma nova rodada é proposta. Esse processo se repete até todos os membros entrarem em um consenso. (COHN, 2005; COHN, 2009; MCDAID, 2008; SALINAS, 2014).

3.2.6 Decisão multicritério

A toma de decisão envolvendo multicritérios, segundo Zeleny (1982), pode ser definida como um esforço para resolver o dilema de critérios conflituosos, cuja presença impede a existência da “solução ótima” e conduz à procura da “solução de melhor compromisso”. A partir dessa definição fica evidente a importância de métodos de apoio à tomadas e decisões. A proposta dos métodos é oferecer, ao decisor, técnicas e ferramentas capazes de torná-lo apto a resolver problemas levando em consideração os mais diversos pontos de vista, muitas vezes contraditórios (VINCKE, 1992).

Uma metodologia de apoio à tomada de decisão multicritério deriva do fato de que, a situação em que se deve decidir (alternativas), não existe apenas um objetivo, e sim vários pontos de vistas conflitantes. Por isso o

processo de decisão deve ser orientado por um método multicritério, o qual apoiará o decisor na otimização das alternativas. A metodologia em si visa auxiliar no processo de escolher, ordenar ou classificar as alternativas (TROJAN, 2012).

Atualmente existem vários métodos multicritérios os quais são identificados como: aditivos (MAUT e AHP) e os de sobreclassificação (*ELECTRE* e *PROMETHEE*). A adoção desses modelos é justificada por argumentos ditados pela natureza do problema a analisar, do contexto considerado e da estrutura de preferência do decisor. (VINCKE, 1992; ALMEIDA e COSTA, 2003).

De acordo com Carignano (2004) os tipos de problemas de decisão multicritério, a ser considerado para a escolha do método, são divididos nos seguintes grupos:

- Tipo α (Pα): Selecionar a “melhor” alternativa ou “melhores” alternativas;
- Tipo β (*P*β): Classificar as alternativas, estabelecendo uma pré-ordem incompleta, ou seja, simplesmente distinguir as alternativas “boas” das alternativas “ruins”;
- Tipo γ(*P*γ): Gerar uma pré-ordem completa, ou seja, simplesmente ordernar as alternativas;
- Tipo δ(*P*δ): Elaborar uma descrição das alternativas.

Aspectos ligados às preferências do decisor são geralmente considerados para a escolha de um método. Muitas vezes, a falta de intimidade com outros métodos, implica ao decisor a escolha de uma determinada metodologia, o qual nem sempre é a mais apropriada para a situação.

De acordo com Almeida e Costa (2003) a aplicação de qualquer método de análise multicritério pressupõe a necessidade de se estabelecer quais objetivos o decisor pretende alcançar.

3.2.7 Metodologia

Este estudo trata de uma pesquisa de natureza aplicada com visões subjetivas (dados qualitativos) e objetivas (dados quantitativos) (CAUCHICK, 2012). Devido ao caráter empírico do trabalho em questão, o procedimento utilizado na pesquisa é um estudo de caso.

A Figura 2 ilustra as principais etapas e seus respectivos produtos utilizados para o desenvolvimento deste estudo.

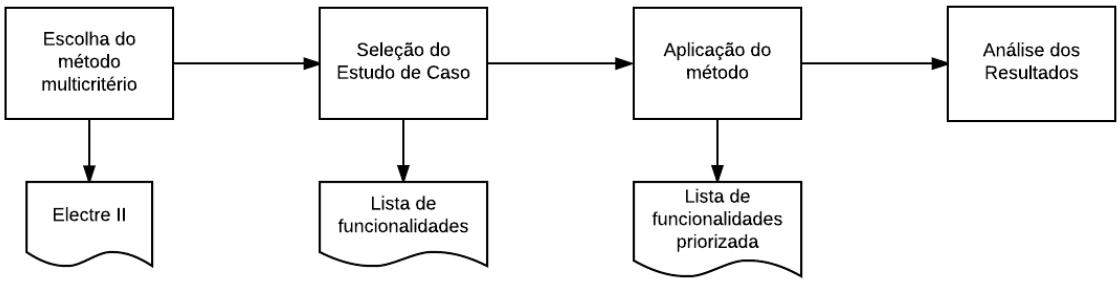

Figura 2 - Etapas para execução do trabalho. Fonte: Dados da pesquisa

A primeira etapa foi a escolha do método multicritério mais adequado para ser empregado. Após a seleção do método, foi selecionado o processo para estudo de caso, coleta e análise dos dados. Como resultado, foi obtido uma lista com as funcionalidades do produto, algumas já desenvolvidas pela empresa, outras ainda sem priorização. A última etapa passo foi a aplicação do método multicritério na lista de funcionalidades no processo estudado. Assim sendo, obteve-se uma lista priorizada de acordo com os critérios estabelecidos pelo método. Por fim, uma análise dos resultados foi realizada a fim de comparar a priorização alcançada com o método com aquela anteriormente aplicada pela empresa, objeto de estudo.

Cada uma das etapas ilustradas na Figura 2 é apresentada de forma detalhada nas próximas seções.

3.2.7.1 Escolha do método multicritério

A escolha do método multicritério para auxílio à tomada de decisão depende diretamente da natureza do problema a ser analisado. Para o propósito do trabalho em questão, o contexto está relacionado à ordenação, classificação ou ranqueamento das demandas à serem desenvolvidas em um *software* de grande porte, assim sendo, a problemática a ser considerada é a do Tipo γ(*P*γ):

Gerar uma pré-ordem completa, ordenando cada uma das alternativas, conforme ilustrado na Figura 3.

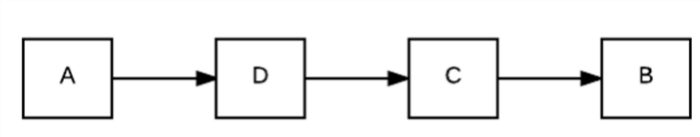

Figura 3 - Problemática de ordenação completa. Fonte: Adaptado de Frega (2008)

Para o sucesso do produto faz-se necessário escolher o momento certo para o desenvolvimento de respectivas funcionalidades, tendo o propósito de entregar valor para o cliente e diminuir o risco de entregar funcionalidades que serão poucas vezes ou nunca utilizadas, diminuindo assim o desperdício, como funcionalidades adicionais, conforme destacado por Poppendieck (2003),.

A escolha do momento certo para o desenvolvimento dos incrementos de um *software* também irá auxiliar no processo de balanceamento do fluxo de produção das funcionalidades do produto. Com isso estima-se reduzir o tempo de ociosidade em cada estação de trabalho. “Esperar pela disponibilidade de pessoas que estão trabalhando em outras áreas é uma grande causa do desperdício proveniente do atraso” Poppendiceck (2003).

Considerando a problemática apresentada, do Tipo γ(*P*γ), ou seja, ordenação de cada uma das alternativas, os esforços do respectivo trabalho se concentra na utilização do método de sobreclassificação *ELECTRE* II. A escolha deste método se justifica pelo fato de ser um método não compensatório, possibilitando a adoção de critérios de natureza distintos, como por exemplo, critérios qualitativos e quantitativos e que serão aplicados neste trabalho. Além disso o *ELECTRE* II é um método clássico e de fácil adoção, quando comparado com outros métodos de sobreclassificação.

3.2.7.2 *ELECTRE* II

Uma das famílias mais conhecidas de métodos de subordinação é a família *ELECTRE* (*Elimination Et ChoiceTRaidusaint la Realitè*). Atualmente composta dos métodos: *ELECTRE*, *ELECTRE* II, *ELECTRE* III, *ELECTRE* IV, *ELECTRE IS* e *ELECTRE TRI* (VINCKE, 1992; ROY, 1971).

Os Métodos *ELECTRE* se caracterizam por utilizar o conceito de superação, subordinação, superclassificação, prevalência ou, até mesmo,

dominação. Como princípio, nestes métodos consideram-se como dominadas as alternativas que "perdem" para as demais (ou são piores que as demais) em um maior número de critérios. No presente trabalho o método utilizado foi o *ELECTRE* II, proposto em ROY (1971) e também reportado em VINCKE (1992). Este método é caracterizado por tratar de problemas específicos de ordenação, ou seja, dado um conjunto A de alternativas, o *ELECTRE* II ordena-as, considerando o desempenho de A a um conjunto de critérios B, e assim por diante.

O respectivo método tem como meta ordenar ações, da melhor para a pior, por meio dos conceitos de Índice de Concordância – C(a,b) e Índice de Discordância – D(a,b). A ideia principal do método é escolher as alternativas que satisfaçam a maioria dos critérios e que não ultrapassem um determinado nível de descontentamento aceito por parte do decisor. Para possibilitar a análise de discordância e concordância são definidos valores de referência: Limiar de Concordância (p) e Limiar de Discordância (q). Estes são pertencentes a um intervalo de variação entre 0 e 1. Esses valores indicam que a concordância desejada deve ser superior ou igual a *p*, e que a discordância deve ser menor ou igual a *q*.

Por fim, o procedimento resulta em dois estágios distintos: O primeiro define a ordenação progressiva e o segundo a ordenação regressiva. As duas são diferentes, porém próximas. O decisor poderá escolher a média entre elas, ou então redefinir o problema e reaplicar o método (VINCKE, 1992; ROY, 1971).

3.2.8 Estudo de caso

O objeto selecionado para aplicação do estudo de caso é uma empresa desenvolvedora de *softwares* de grande porte especializada em soluções para gestão de supermercados, materiais de construção e restaurantes. A organização tem sua matriz situada no sudoeste do estado do Paraná. A empresa tem como missão: Melhorar a gestão e os resultados de seus clientes. A corporação está presente em todos os estados do Brasil e conta atualmente com uma carteira de mais de 3 mil clientes e mais de 100 mil usuários ativos em cada uma de suas soluções. Atualmente a empresa possui cerca de 350 colaboradores.

Para análise e aplicação do processo de multicritério o departamento selecionado foi a Fábrica de *software* de novos produtos. Durante o emprego do estudo de caso, a fábrica trabalhava na criação de um produto para operacionalização de vendas em supermercados de pequeno porte, referenciado no contexto desse trabalho como SGS.

O SGS já havia sido implantado no primeiro cliente e estava em período de maturação para, posteriormente, ser colocado à venda junto aos demais produtos comercializados pela empresa. Esse período de maturação do produto é importante afim de coletar dados para a sequência e evolução do *software*. “Para novos produtos, sempre há um período de amadurecimento antes de colocá-lo à venda. É assim que coletamos informações necessárias para sermos mais assertivos durante a construção do *software*”, destacou o Gerente de Produto.

A Fábrica de Novos Produtos utiliza das metodologias ágeis de desenvolvimento de *software* para concepção e elaboração dos seus produtos, mais especificamente a metodologia ágil *Scrum*. Dessa forma o SGS possui uma lista com todas as funcionalidades do produto, trata-se do *Product Backlog.* No momento em que os dados foram levantados a lista era composta de 105 funcionalidades: 71 já implementadas e outras 34 não.

A empresa mantém a lista de funcionalidades em uma plataforma *online* chamada IBM Jazz. Nessa ferramenta, além da lista, foi possível coletar maiores detalhes de cada funcionalidade: requisitos, complexidade, status (desenvolvido ou não), prioridade e histórico de alterações. Além da coleta dos dados na ferramenta *web*, outras informações foram obtidas a partir de entrevistas com gestores e envolvidos na solução. Trata-se da priorização de algumas funcionalidades que foram realizadas usando critérios subjetivos. Essa falta de rigor na priorização das funcionalidades gerou, em algumas entregas, funcionalidades pouco utilizadas e sem valor para o produto como destaca o *Product Owner* (PO): “Em algumas iterações, priorizamos e entregamos funcionalidades que nossos clientes não estavam preparados para usar, precisamos de mais outras *Sprint’s* para adaptar o *software* novamente”.

3.2.9 Aplicação do método multicritério ELECTRE II para priorização do *Product Backlog*

Durante o planejamento de cada iteração, o PO apresenta para o time de desenvolvimento quais são as funcionalidades a serem consideradas na próxima *Sprint*. Trata-se dos itens mais prioritários do *Product Backlog*. A priorização dos itens a serem desenvolvidos é feita durante toda a execução do projeto. O PO mantém o *Product Backlog* constantemente priorizado de acordo com uma série de fatores. Entretanto, esses fatores não são constantes e nem estáveis.

Pelo fato de não haver um padrão para priorização dos itens, o PO do produto SGS, muitas vezes se depara com dúvidas para saber quais itens devem ser priorizados: “Atualmente nossa priorização é muito subjetiva. Por hora consideramos a opinião dos clientes, outra a opinião da equipe técnica e algumas vezes a opinião dos diretores da empresa”. Essa incerteza na priorização das próximas demandas tem causado problemas nas entregas do produto: “Aconteceu casos de entregarmos funcionalidades que o nosso cliente não precisava. Poderíamos ter priorizado outros recursos, entregando maior valor para nosso cliente e produto”, destacou o PO do SGS.

Com base nas problemáticas destacadas pelo *Product Owner* em iterações passadas, foi aplicado o método multicritério *ELECTRE* II no *Product Backlog* do SGS, proporcionando assim a classificação de cada funcionalidade (alternativas) da mais relevante à menos importante de acordo com critérios a serem apresentados.

3.2.9.1 Definição dos critérios

A escolha dos critérios foi efetuada com o objetivo de classificar o *Product Backlog* com os critérios que representem os reais interessados na solução. De acordo com o PO, os interessados na solução são: Clientes, Diretores da empresa, Patrocinadores do projeto e Time de desenvolvimento. O decisor assumido, que definiu cada um dos critérios, é o PO do produto SGS, cuja responsabilidade se adéqua ao papel exercido dentro do *framework* SCRUM: Refinar e Priorizar o *Product Backlog* (SUTHERLAND, 2012). O *Product Owner*, utilizou do seu domínio de negócio e experiências passadas no

desenvolvimento de soluções para supermercados para a escolha de cada um dos critérios.

3.2.9.1.1 Critério 1 (C1) – importância para o negócio do cliente

Trata-se de um critério qualitativo que busca identificar a real importância de uma alternativa, para o contexto do cliente piloto da solução. Por ocasião do levantamento, um único cliente piloto utilizava o SGS. Para cada alternativa é atribuído um número que representa o grau de relevância da mesma. O PO, em comum acordo com o cliente do SGS, impõe uma escala de importância para cada item. Essa escala varia entre: Essencial, Importante e Desejável. Sendo que a escala maior, essencial, indica que o cliente está impossibilitado de usar determinada parte do sistema pela falta da referente funcionalidade. Importante, quer dizer que o cliente possui uma outra forma de usar o recurso. Entretanto, faz-se necessário tornar o seu uso mais prático por meio da implementação da funcionalidade. Desejável, quer dizer que o cliente não sofre nenhum impacto negativo pelo fato do sistema não possuir a funcionalidade. Porém, a implementação dela traria benefícios interessantes para o produto e sua utilização futura. Esse critério pretende responder a seguinte questão: “Quanto valor esse recurso traz para sua organização?”. A técnica utilizada para obtenção dos respectivos valores, chamada de Análise de Pontos de Valor, propõe a atribuição dentro de uma escala ordinal para cada funcionalidade. Os valores propostos são: 10000, 5000 e 1000 os quais refletem respectivamente: Essencial, Importante e Desejável (HIGHSMITH, 2009).

A Tabela 1 ilustra a escala de importância para o negócio do cliente.

Tabela 1 - Critério importância para o negócio do cliente. Fonte: Dados da pesquisa

Relevância	Valor
Essencial	10000
Importante	5000
Desejável	1000

3.2.9.1.2 Critério 2 (C2) – complexidade

Esse também é um critério qualitativo que reflete a complexidade em analisar, desenvolver e testar cada nova funcionalidade. O valor de complexidade é atribuído a cada alternativa por meio de estimativas realizada por todos os membros do time. Para cada item do *Product Backlog* é atribuído um número que indica o quão complexo é para analisar, desenvolver e testar a respectiva funcionalidade. A complexidade não foi tratada como um critério de minimização devido à uma prática da equipe. Esta busca priorizar o desenvolvimento de funcionalidades complexas com o intuito de conhecer melhor a solução e minimizar possíveis impedimentos técnicos.

Para atribuir o valor de complexidade de cada item o time utiliza da técnica de estimativa, apresentada na seção 2, chamada *Planning Poker*. Os valores das cartas de baralho representam a complexidade a ser definida para cada funcionalidade. Durante a cerimônia de planejamento de cada iteração (*Sprint Planning*) é estimada a complexidade de cada item que irá compor a próxima *Sprint*. Um número é adicionado ao respectivo item representando sua complexidade. Além desse momento, há um dia na *Sprint* reservado para o time estimar os demais itens do *Product Backlog*. Como o *Product Backlog* é constantemente alterado, faz-se necessário sempre manter os itens que o compõe estimados com sua real dificuldade.

A complexidade de cada item é utilizada pelo time também para saber quantas novas funcionalidades cabem na próxima *Sprint*. Há uma média de valor que consideram como a velocidade por iteração, atualizada ao término de cada *Sprint*. O cálculo para se chegar a essa média é a soma de todos os valores entregues em cada uma das iterações, dividido pelo número de ciclos já executados, ver Equação 1.

$$X = \frac{\sum(p^{+})}{(SP\ *)} \qquad (1)$$

Na qual:

- $\boldsymbol{p}^{+}$corresponde à soma dos pontos entregues em todas as iterações;
- $\boldsymbol{SP}*$ corresponde ao número de iterações já desenvolvidas e entregues.

No momento do presente estudo já haviam sido entregues 548 pontos, em um total de 12 *Sprint's* ou iterações. Assim a, velocidade do time estava com uma capacidade de entrega média de 45,6 *story points* por *Sprint*.

3.2.9.1.3 Critério 3 (C3) – valor agregado de mercado

É um critério quantitativo que indica o valor de negócio que foi adicionado ao produto final a partir da implementação de tal funcionalidade. A perspectiva da empresa com relação ao SGS é colocá-lo à venda junto aos demais produtos do seu portfólio. Sendo assim, diferentes clientes utilizarão o *software*. Dessa forma, faz-se importante construir funcionalidades que possam não ter valor de negócio para o cliente piloto atual, porém, pensando no produto final que será posteriormente comercializado em escala. O incremento tornará o SGS mais competitivo e com maior valor de mercado. Essa análise é feita atualmente em acordo com os patrocinadores, gerentes e diretores da organização. Cada um atribui um valor de mercado para cada novo incremento, e por fim, uma média dos valores é considerada em cada funcionalidade.

Uma reunião é feita em intervalos regulares, geralmente a cada bimestre, onde o *Product Owner*, apresenta aos gestores e patrocinadores do projeto os itens do *Product Backlog* que foram inseridos no período. Depois de apresentado, cada envolvido recebe um valor fictício que representa uma unidade monetária qualquer. O esforço deles é então distribuir o montante recebido entre cada uma das novas funcionalidades do *Product Backlog*. O resultado final é uma média dos valores atribuídos a cada funcionalidade por parte de cada envolvido. Em alguns casos determinados envolvidos possuem um peso de decisão maior que outros. Esse peso é considerado ao valor atribuído à respectiva funcionalidade. A Equação 2 é utilizada para chegar no valor agregado de cada funcionalidade.

$$X = \frac{\sum (V^{+}.\mathrm{P})}{(E\ *)} \quad (2)$$

Na qual:

- $\boldsymbol{V^{+}}$ corresponde aos valores atribuídos à cada funcionalidade por parte de cada envolvido;
- $\mathbf{P}$ é o peso que cada envolvido possui. Em alguns casos, patrocinadores ou diretores, possuem um peso maior comparado a outros participantes, para tanto, o valor por ele atribuído é maior que dos demais;
- $\boldsymbol{E}\ *$ corresponde à quantidade de envolvidos no processo de atribuição de valores para cada funcionalidade.

Como o *Product Backlog* é um artefato que sofre mudanças no decorrer do projeto, essas reuniões são feitas em intervalos correspondentes às respectivas mudanças, ou seja, quando o PO identifica que existem mudanças suficientes para uma nova rodada de atribuição de valor de mercado ao produto. Ele convoca cada um dos envolvidos e realiza uma nova rodada de atribuição de valor de negócio aos novos itens no *backlog*. Durante o estudo de caso, duas reuniões já haviam sido feitas com envolvidos distintos.

3.2.9.1.4 Critério 4 (C4) – exigência legal

Devido ao fato do SGS ser um sistema de automação comercial existem algumas restrições impostas pela legislação, sejam estas estaduais ou federais. Quando alguma nova funcionalidade faz referência à alguma lei, a mesma deve ser imediatamente considerada para ser implementada. As leis estaduais precisam ser consideradas devido ao fato da empresa ter abrangência em todos os estados do país. Devido ao fato do produto ter somente alcance nacional somente leis brasileiras são implementadas.

Assim sendo, trata-se de um critério qualitativo de escala verbal, ou seja, os valores possíveis são: Verdadeiro ou Falso.

3.2.9.2 Peso dos critérios

O peso adotado para cada um dos critérios anteriormente descritos foi estabelecido pelo decisor, ou seja, pelo *Product Owner* do produto SGS, em

comum acordo com os gestores da empresa. Depois de definir os respectivos pesos dos critérios, os mesmos foram normalizados conforme ilustrado na Tabela 2.

Tabela 2 - Peso dos critérios. Fonte: Dados da pesquisa

	Critério	Peso Atribuído	Peso normalizado
C1	Importância para o negócio do cliente	15	0,15
C2	Complexidade	25	0,25
C3	Valor agregado ao produto	20	0,20
C4	Exigência legal	40	0,40

3.2.9.3 Alternativas

As alternativas consideradas envolvem todos os itens que constituíam o *Product Backlog* do *software* SGS. No momento do estudo de caso, o *Product Backlog* continha 105 funcionalidades divididas entre os seguintes status: 71 já implementadas e integradas ao produto e 34 ainda não prontas. Com o propósito de manter o sigilo das informações, o nome de cada alternativa foi trocado pelo acrônimo da palavra: Alternativa (Alt), seguido de um número sequencial único, por exemplo: Alt 1, Alt 2, e assim por diante. A lista com as funcionalidades foi organizada alfabeticamente (A-Z), e por fim, cada item teve o seu nome substituído pela sua abreviação.

O propósito de não desconsiderar da análise as funcionalidades já entregues, foi o de possibilitar comparar a priorização resultante da aplicação do método multicritério com as diferentes priorizações anteriormente feitas pela empresa. É importante destacar que a forma utilizada para priorizar as funcionalidades, antes do presente trabalho, era estritamente relativa, ou seja, por hora dava-se prioridade àquelas que o cliente sugeria, outras àquelas que eram exigências legais junto às sugestões dos diretores e envolvidos.

3.2.10 Apresentação dos resultados

Para cada uma das 105 alternativas (itens do *Product Backlog*), analisando documentos do projeto e em conjunto com o PO do produto SGS, os

valores dos critérios foram atribuídos. A partir da valorização, os mesmos foram normalizados, colocando-os em uma escala proporcional de 0 a 100, conforme ilustra a Tabela 3.

Tabela 3 - Critérios normalizados. Fonte: Dados da pesquisa

C1		C2		C3		C4	
Valor	Normalizado	Valor	Normalizado	Valor	Normalizado	Valor	Normalizado
1000	30	1	10	1000 a 5000	30	S	100
5000	60	2	20	5001 a 10000	60	N	0
10000	100	3	30	10001 ou maior	100		
		5	45				
		8	60				
		13	75				
		20	80				
		40	100				

Com os valores dos critérios normalizados, cada alternativa, frente a cada um dos critérios, foi valorizada de acordo com a respectiva importância. A atividade de atribuição dos valores foi realizada a partir de análise de documentos produzidos durante o projeto nas respectivas ferramentas de gestão de requisitos, além de entrevistas realizadas com o PO do SGS. O Apêndice A ilustra cada alternativa com seus respectivos valores normalizados.

Após a normalização dos critérios e pesos foram calculados os índices de concordância e discordância, conforme previamente prescrito pelo método *ELECTRE* II. O resultado foi a projeção de duas matrizes quadradas (105 x 105). Amostras de ambas as matrizes estão ilustradas nos Apêndices B e C desse trabalho.

Para poder analisar essas matrizes foram definidos os limiares de concordância (p) e de discordância (q). Os valores foram definidos a partir de uma média aritmética simples, entre os valores de ambas as matrizes (Matriz de Concordância e Matriz de Discordância), sendo que o índice de concordância (p) é igual a 0,6 e o de discordância (q) 0,4. Os respectivos atributos foram usados para a comparação das alternativas. Diz-se que uma alternativa “aSb” (a subordina b) quando são satisfeitas às condições de concordância e

discordância. Os valores revelam o rigor utilizado para admitir que uma alternativa domina a outra.

Tendo o conjunto de alternativas criadas, seus critérios e pesos normalizados, os limiares de concordância e discordância calculados, o próximo passo foi estabelecer as relações de sobreclassificação e incomparabilidade, para que posteriormente seja possível realizar a última etapa do método: ordenação e ranqueamento das alternativas. O procedimento de ordenação é feito em duas partes: Ordenação Descendente (da melhor para a pior alternativa) e, Ordenação Ascendente (da pior para a melhor alternativa). A classificação final das alternativas é obtida através da mediana das classificações alcançadas em cada estágio.

A ordenação final proposta pelo método *ELECTRE* II é ilustrado no Quadro 4, organizada, da esquerda para a direita, a partir da alternativa melhor ordenada, ou seja, aquela que mais domina e é menos dominada pelas demais. A priorização de execução seriam as funcionalidades 36, 87, 93, 94 e assim por diante.

Quadro 4 - Resultado de ordenação *ELECTRE* II. Fonte: Dados da pesquisa

36	87	93	94	40	88	12	4	5	39	46	47	48	98	6	26	45	104	37	49
14	100	27	86	96	11	41	101	90	89	66	19	21	50	43	15	38	57	85	70
23	71	51	62	83	58	82	84	103	1	2	91	18	44	7	20	28	29	72	105
92	95	17	67	3	32	59	68	69	80	99	33	74	55	56	75	30	65	54	73
52	53	63	34	81	64	76	102	10	24	35	22	31	97	13	60	78	8	77	9
16	25	61	42	79															

3.2.10.1 Análise dos resultados

Conforme já apresentado, o *Product Backlog,* era composto de 105 funcionalidades, dividas entre incrementos já incorporados ao produto (71) e outras ainda não desenvolvidas e nem planejadas (34).

Devido à natureza incremental da metodologia de desenvolvimento de *software* utilizado pela empresa, entregas foram planejadas e feitas durante todo o processo de produção do SGS, outras ainda estão sendo planejadas para

serem feitas no decorrer do ciclo de vida do produto. Cada entrega realizada e planejada a ocorrer é chamada de *Release*. De acordo com Sommervile (2011), uma *Release* é uma nova versão de um *software*, a qual passa por todos os procedimentos e atividades envolvidas no processo de entrega de um produto de *software*.

Até o momento da presente pesquisa, duas *releases* já haviam sido planejadas e entregues ao cliente piloto. Ambas contemplaram um total de 71 funcionalidades entregues. A Tabela 4 ilustra as funcionalidades do SGS priorizadas anteriormente, sem a utilização de nenhum método específico. Estão tabuladas as funcionalidades que já foram planejadas, desenvolvidas e integradas ao produto final (*Release 1 e Release 2*), e aquelas ainda não distribuídas em nenhuma iteração (não planejadas). Para efeito de comparação, a Tabela 5 exibe como seria o planejamento das *Releases* do SGS a partir da aplicação do *ELECTRE* II. Os itens destacados na Tabela 5, correspondem aqueles que o método priorizou para as mesmas *Releases* que a priorização realizada pela equipe do SGS.

Confrontando a priorização obtida a partir da aplicação do método multicritério com aquela já adotada pela equipe do SGS, na primeira *Release*, 23 funcionalidades (48,9%), foram priorizadas iguais, comparando o método, com a prática aplicada pela equipe. Já na segunda *Release,* das 24 funcionalidades desenvolvidas e entregues, observa-se que 7 delas tiveram a mesma importância, ou seja, 29% de semelhança entre ambas as práticas de priorização. Por fim, do total de funcionalidades não planejadas (36), o método ELECTRE II priorizou 7 delas (20%) com a mesma baixa relevância. Assim sendo, das 36 funcionalidades ditas como pouco relevantes, 29 delas já foram desenvolvidas e entregues nas *Releases* 1 ou 2, ou seja, 80% daquilo que foi priorizado como não importante já foi desenvolvido e entregue ao cliente.

Comparando aquilo que já foi desenvolvido e entregue em alguma *Release,* com a priorização do método ELECTRE II, a sequência do desenvolvimento dos próximos incrementos do SGS, seriam as seguintes: 33, 102, 97, 60, 8, 61, 79 e o restante em ordem aleatória, sendo que, as demais já deveriam ter sido planejadas e desenvolvidas.

Tabela 4 - Priorização de funcionalidades sem aplicação do método. Fonte: Dados da pesquisa

Release 1															
58	63	64	65	62	66	67	68	69	70	71	72	73	74	75	76
77	78	50	51	52	53	34	43	44	45	46	47	48	49	55	87
35	37	38	39	41	30	31	32	40	54	56	57	82	100	104	
Release 2															
24	25	26	36	42	83	23	16	17	19	22	7	103	10	1	2
3	4	5	9	13	27	28	81								
Não planejadas															
6	8	11	12	14	15	18	20	21	29	33	59	60	61	79	
80	84	85	86	88	89	90	91	92	93	94	95	96	97	98	
99	101	102	105												

Tabela 5 - Simulação de priorização com ELECTRE II. Fonte: Dados da pesquisa

Release 1															
36	**87**	93	94	**40**	88	12	4	5	**39**	**46**	**47**	**48**	98	6	26
45	**104**	**37**	**49**	14	**100**	27	86	96	11	**41**	101	90	89	**66**	19
21	**50**	**43**	15	**38**	**57**	85	**70**	23	**71**	**51**	**62**	83	**58**	**82**	
Release 2															
84	**103**	**1**	**2**	91	18	44	**7**	20	**28**	29	72	105	92	95	**17**
67	**3**	32	59	68	69	80	99								
Não planejadas															
33	74	55	56	75	30	65	54	73	52	53	63	34	81	64	
76	**102**	10	24	35	22	31	**97**	13	**60**	78	**8**	77	9	16	
25	**61**	42	**79**												

Sobre as funcionalidades já entregues, usuários do SGS foram questionados se tinham conhecimento de cada uma das 71 do sistema. Eles deveriam responder sim ou não. Como o sistema estava em processo de maturação em um único cliente, somente 3 usuários foram indagados. Todos eles eram usuários diretos da aplicação, ou seja, em suas atividades de trabalho

estavam em contato diário com o SGS. Das 71 funcionalidades desenvolvidas e entregues, 46 delas foram destacadas como conhecidas pelos usuários do sistema, ou seja, cerca de 65%. As demais foram descritas como não conhecidas pelos usuários do SGS.

Pode-se comprovar que a falta de um método que considerasse diferentes critérios para a priorização, ocasionou em alguns casos o desperdício com a produção de funcionalidades pouco importantes para o produto e seus usuários, desperdício esse, que poderia ser reduzido a partir da utilização de um método para priorização das demandas de desenvolvimento.

A Tabela 6 ilustra as funcionalidades não planejadas, priorizadas de acordo com o ELECTRE II, e destaca aquelas que já foram desenvolvidas e incorporadas ao produto, em alguma das duas *Releases.* Observa-se que a maioria delas (80%), já tiveram o seu desenvolvimento priorizado em entregas anteriores, ou seja, 80% delas foram incrementadas ao produto, no momento errado. Esse esforço poderia ter sido reduzido a partir da utilização do ELECTRE II.

Tabela 6 - Funcionalidades não planejadas pelo ELECTRE II. Fonte: Dados da pesquisa

Funcionalidades não planejadas														
33	**74**	**55**	**56**	**75**	**30**	**65**	**54**	**73**	**52**	**53**	**63**	**34**	**81**	**64**
76	102	**10**	**24**	**35**	**22**	**31**	97	**13**	60	**78**	8	**77**	**9**	**16**
25	**61**	**42**	**79**											

3.2.11 Considerações finais

Como objetivo do trabalho, procurou-se a priorização de demandas de produção de *software* de grande porte, utilizando a aplicação de um método multicritério, buscando a redução de desperdícios com superprodução, ou seja, reduzindo o desenvolvimento de funcionalidades que não serão utilizadas. Pelos resultados encontrados pode-se verificar que o método proposto tem o potencial de redução esperado.

Analisando os resultados obtidos a partir da aplicação do método multicritério *ELECTRE* II, pôde-se observar uma baixa semelhança com a

priorização utilizada pela empresa. Entretanto, vale ressaltar que pelos critérios da própria organização, foram priorizados o desenvolvimento de funcionalidades que o cliente não utilizou, causando uma série de desperdícios. O tempo despendido com o desenvolvimento de uma funcionalidade não utilizada poderia ser melhor aplicado, sendo direcionado para funcionalidades mais relevantes ou atividades que gerassem um retorno para a empresa. A falta de critérios, para a correta priorização de próximas demandas de desenvolvimento de *software*, em alguns momentos ocasionou o planejamento e entrega de funcionalidades que não foram bem aceitas pelo cliente da solução, constatação do Product *Owner* do SGS.

Estudos desta natureza tornam-se cada vez mais importantes. A partir dos dados apresentados no início deste trabalho, e confirmados pelos resultados, ilustram a grande quantidade de funcionalidades desenvolvidas e entregues sem a real necessidade por parte dos clientes e envolvidos na solução, ocasionando cada vez mais desperdício no desenvolvimento de *software*.

Como trabalho futuro sugere-se o desenvolvimento de funcionalidades em um sistema de grande porte utilizando o resultado da priorização de demandas obtidos a partir da aplicação do método de multicritério *ELECTRE* II. Com o acompanhamento dos resultados obtidos, e a análise com os envolvidos, será possível uma avaliação completa e aferição do método.

Apêndice A - Lista de Alternativas. Fonte: Dados da Pesquisa

	C1	C2	C3	C4
Alt 1	60	80	60	0
Alt 2	60	80	60	0
Alt 3	60	45	60	0
Alt 4	100	60	100	100
Alt 5	100	60	100	100
Alt 6	100	45	100	100
Alt 7	60	75	60	0
Alt 8	30	45	30	0
Alt 9	30	30	30	0
Alt 10	30	75	30	0
Alt 11	100	60	100	0
Alt 12	100	75	100	100
Alt 13	30	60	30	0
Alt 14	100	75	100	0
Alt 15	100	45	100	0
Alt 16	30	30	30	0
Alt 17	30	45	30	100
Alt 18	60	30	60	100
Alt 19	60	60	60	100
Alt 20	30	45	30	100
Alt 21	60	60	60	100
Alt 22	30	15	30	100
Alt 23	30	100	30	100
Alt 24	30	75	30	0
Alt 25	30	30	30	0
Alt 26	100	45	100	100
Alt 27	100	75	100	0
Alt 28	100	75	100	100
Alt 29	100	75	100	100
Alt 30	60	45	60	0
Alt 31	60	15	60	0
Alt 32	60	45	60	0
Alt 33	60	30	60	0
Alt 34	60	30	60	0
Alt 35	30	75	30	0

	C1	C2	C3	C4
Alt 36	100	100	100	100
Alt 37	100	30	100	100
Alt 38	100	45	100	0
Alt 39	100	60	100	100
Alt 40	100	75	100	100
Alt 41	100	60	100	0
Alt 42	30	15	30	0
Alt 43	100	15	100	100
Alt 44	60	30	60	100
Alt 45	100	45	100	100
Alt 46	100	60	100	100
Alt 47	100	60	100	100
Alt 48	100	60	100	100
Alt 49	100	30	100	100
Alt 50	60	60	60	100
Alt 51	60	45	60	100
Alt 52	60	30	60	0
Alt 53	60	30	60	0
Alt 54	100	30	100	0
Alt 55	100	15	100	0
Alt 56	100	15	100	0
Alt 57	100	45	100	0
Alt 58	100	30	100	0
Alt 59	60	45	60	0
Alt 60	30	60	30	0
Alt 61	30	30	30	0
Alt 62	60	45	60	100
Alt 63	60	30	60	0
Alt 64	60	30	60	0
Alt 65	60	15	60	0
Alt 66	100	60	100	0
Alt 67	100	45	100	0
Alt 68	60	45	60	0
Alt 69	60	45	60	0
Alt 70	30	45	30	100

	C1	C2	C3	C4
Alt 71	100	75	100	0
Alt 72	100	15	100	0
Alt 73	100	30	100	0
Alt 74	60	30	60	0
Alt 75	100	15	100	0
Alt 76	30	100	30	0
Alt 77	30	45	30	0
Alt 78	30	60	30	0
Alt 79	30	15	30	0
Alt 80	60	45	60	0
Alt 81	60	15	60	0
Alt 82	60	100	60	0
Alt 83	60	45	60	100
Alt 84	100	30	100	0
Alt 85	100	45	100	0
Alt 86	100	75	100	0
Alt 87	100	100	100	100
Alt 88	100	75	100	100
Alt 89	60	60	60	100
Alt 90	60	100	60	100
Alt 91	60	80	60	0
Alt 92	60	60	60	0
Alt 93	100	100	100	100
Alt 94	100	100	100	100
Alt 95	60	60	60	0
Alt 96	100	75	100	0
Alt 97	60	15	60	0
Alt 98	100	60	100	100
Alt 99	60	45	60	0
Alt 100	100	80	100	0
Alt 101	100	60	100	0
Alt 102	60	45	60	0
Alt 103	100	30	100	0
Alt 104	100	45	100	100
Alt 105	60	45	60	0

Apêndice B – Amostra da Matriz de Concordância. Fonte: Dados da Pesquisa

	1	**2**	**3**	**4**	**…**	**100**	**101**	**102**	**103**	**104**	**105**
1	0	0,5	0,625	0,25	…	0,325	0,45	0,625	0,45	0,25	0,625
2	0,5	0	0,625	0,25	…	0,325	0,45	0,625	0,45	0,25	0,625
3	0,475	0,375	0	0	…	0,2	0,2	0,5	0,45	0,125	0,5
4	0,75	0,75	1	0	…	0,575	0,7	1	0,825	0,625	1
…	…	…	…	…	…	…	…	…	…	…	…
100	0,675	0,675	0,8	0,425	…		0,625	0,8	0,625	0,425	0,8
101	0,55	0,55	0,8	0,3	…	0,375		0,8	0,625	0,425	0,8
102	0,375	0,375	0,5	0	…	0,2	0,2		0,45	0,125	0,5
103	0,55	0,55	0,55	0,175	…	0,375	0,375	0,55		0,175	0,55
104	0,75	0,75	0,875	0,375	…	0,575	0,575	0,875	0,825		0,875
105	0,75	0,75	1	0	…	0,4	0,4	1	0,65	0,25	

Apêndice C – Amostra da Matriz de Discordância. Fonte: Dados da Pesquisa

	1	**2**	**3**	**4**	**…**	**100**	**101**	**102**	**103**	**104**	**105**
1	0	0	0	1	…	0,4	0,4	0	0,4	1	0
2	0	0	0	1	…	0,4	0,4	0	0,4	1	0
3	0,35	0,35	0	1	…	0,4	0,4	0	0,4	1	0
4	0,2	0,2	-0,15	0	…	0,2	0	-0,15	0	0	-0,15
…	…	…	…	…	…	…	…	…	…	…	…
100	0	0	0	1	…	0	0	0	0	1	0
101	0,2	0,2	0	1	…	0,2	0	0	0	1	0
102	0,6	0,6	0,6	1	…	1	1	0,6	1	1	0,6
103	0,5	0,5	0,15	1	…	0,5	0,3	0,15	0	1	0,15
104	0,35	0,35	0	0,15	…	0,35	0,15	0	0	0	0
105	0,35	0,35	0	1	…	0,4	0,4	0	0,4	1	0

3.3 KANBAN COMO FORMA DE REDUÇÃO DE DESPERDÍCIO NO PROCESSO DE PRODUÇÃO DE *SOFTWARE* DE GRANDE PORTE

A última etapa que irá compor a dissertação em questão, trata-se da aplicação da ferramenta Kanban durante o desenvolvimento de um sistema de grande porte. O respectivo trabalho será submetido ao periódico *The Journal of System and Software*, cujo, de acordo com a classificação do ano de 2004 do Qualis, qualificou-o como A2 na área de Engenharias III. O objetivo do trabalho em questão foi o de cumprir com os seguintes objetivos específicos:

- Melhorar o fluxo das atividades, priorizando demandas, e buscando uma maior estabilidade durante o desenvolvimento de softwares, reduzindo o desperdício com funcionalidades inacabadas e como consequência, propor uma forma de gerenciamento das atividades de desenvolvimento de um sistema de grande porte.

3.3.1 Introdução

A necessidade de lidar com as contínuas mudanças no processo de produção de um *software* é ainda mais evidente, quando se trata da concepção de sistemas de grande porte. Um *software* de grande porte possui propriedades que o definem: (i) Constante evolução e manutenção do sistema; (ii) Alta complexidade; (iii) Múltiplas dimensões do sistema (Modularidade) e (iv) Distribuição do conhecimento entre os desenvolvedores. A precisão de lidar com cada uma das especificidades de um *software* de grande porte, gerou uma série de estudos com o intuito de mitigar os riscos em volta da produção de sistemas dessa categoria (PERRY, 2001).

Desde meados dos anos 2000, metodologias ágeis de desenvolvimento de *software*, ajudam empresas a melhorar a qualidade de seus produtos e satisfazer seus clientes. Os métodos ágeis aumentam a capacidade das organizações de *software* em responder às mudanças e dinamismo do mercado. Por mais benefícios identificados a partir da implementação das metodologias ágeis para a fabricação de *softwares*, sua adoção para o desenvolvimento de sistemas de grande porte, possuem alguns desafios: (i) Custo com a realização de testes contínuos; (ii) Esforço de manutenção com um aumento no número de versões; (iii) Despesas de gerenciamento devido à necessidade de coordenação de equipes diversas; (iv) Dependências entre artefatos de desenvolvimento; (v) Tempo gasto com gerência de requisitos devido à processos complexos; (vi) Listas de prioridades de requisitos

difíceis de criar e manter; (vii) Tempos de espera entre as atividades do processo, especialmente entre o desenvolvimento e a especificação de requisitos; (viii) Redução da cobertura de teste devido à escassez de projetos e falta de testes independentes; (ix) Aumento do esforço de gerenciamento de configuração (MISHRA, 2011; SUOMALAINEN, 2015).

Alternativamente a agilidade, organizações de diferentes setores, adotaram o desenvolvimento *Lean*, almejando alcançar um fluxo de produção contínuo e consistente, reduzindo desperdícios em processos e aumentando o valor para o cliente final. O desenvolvimento *Lean*, significa criar uma organização tolerante às mudanças que possa ter sucesso em tempos de mudanças e incertezas. Ambos os conceitos, agilidade e *Lean*, foram introduzidos no setor de produção de *software* para ajudar empresas, em ambientes de negócio turbulentos, a alcançar prazos de entrega cada vez mais curtos (CHARETTE, 2003; MIDDLETON et al., 2005; POPPENDIECK, 2003; WALLEY, 2008; PETERSEN, 2010).

Devido às constantes mudanças que ocorrem em ambientes de desenvolvimento de *software* de grande porte, muitas corporações buscam adotar as práticas ágeis em conjunto com o desenvolvimento enxuto, devido à capacidade que ambas as filosofias possuem em adequar-se a atmosferas instáveis e proporcionar uma substancial redução de custos e melhorias na qualidade dos produtos entregues (RADNOR, 2008; BELLOMO 2013). Dessa forma surge uma nova filosofia de desenvolvimento de *software*, chamada: *Lean Software Development* (LSD) (POPPENDIECK, 2003). A metodologia, se propõe, principalmente, à traduzir os conceitos da filosofia *Lean Thinking,* para o ambiente de desenvolvimento de *software*, mapeada em torno de 5 principais princípios: (i) Identificar valor para o cliente; (ii) Identificar todas as etapas do fluxo de valor de cada produto, eliminado possíveis etapas que não agregam valor; (iii) Criar um fluxo sequencial para cada produto; (iv) Estabilizar e fazer com que o fluxo de valor seja puxado pelo cliente, a partir das suas reais necessidades; e (v) Garantir a contínua melhoria de todo o processo, a fim de atingir o estado da perfeição, sem nenhum tipo de desperdício. Além dos princípios, um dos fundamentos do pensamento enxuto é a contínua eliminação de atividades desnecessárias, ou seja, eliminar os desperdícios, cujos são definidos como qualquer atividade executada que não agrega valor ao produto final (WORNACK, 1991).

A partir do momento que o mercado de produção de *software* entrou em uma fase de alta competição, com mudanças constantes, prazos curtos e exigências por produtos feitos de forma rápida e eficiente, minimizar a ocorrência dos desperdícios, torna-se um diferencial competitivo para qualquer organização (POPPENDIECK, 2003; COCKBURN, 2001; BECK, 2001). Uma das formas de reduzir desperdício é produzir apenas por demanda. Essa prática, tende a eliminar o desperdício com excesso na produção (superprodução) e acúmulo de subprodutos inacabados (estoques). (ABDELHAMID, 2004; POPPENDIECK, 2003; OHNO, 1988; HOWELL, 1999).

Com base no que fora apresentado, surge então o problema desta pesquisa: Como gerenciar as atividades de desenvolvimento de *software* de grande porte para que desperdícios com estoque e produtos inacabados sejam minimizados? Para responder à essa pergunta, o presente trabalho tem como objetivo aplicar o uso do Kanban em uma fábrica de desenvolvimento de *software* de grande porte, visando reduzir o desperdício com trabalhos inacabados e excesso na produção. O uso do Kanban está relacionado a um dispositivo de sinalização (geralmente um cartão físico em um envelope de papel), que instrui o movimento de peças ou tarefas em um sistema de produção, produzindo somente o necessário.

O presente trabalho fornece evidências de um estudo de caso único, cujos dados foram coletados por meio de entrevistas, observações e análise de documentos produzidos durante o processo de produção de um *software* de grande porte.

3.3.2 Desenvolvimento ágil de *software* de grande porte

Os métodos ágeis evoluíram como uma nova abordagem para o desenvolvimento de *software*. As práticas foram implementadas com sucesso em diversos projetos de pequeno e médio porte, conforme destacado por alguns autores em seus respectivos trabalhos (VANHANEN, 2003; FUQUA, 2003; RASMUSSEN, 2003; SCHUH, 2001; GRENNING, 2001; POOLE, 2001; MURRU et al., 2003; SHARP, 2004; KARLSTROM, 2006).

As metodologias ágeis concentram-se em simplificar e melhorar o processo de construção de *software*, tornando os clientes, os desenvolvedores e o produto final, os produtos mais importantes em todo o processo (AGILE MANIFESTO, 2001). Além disso, o desenvolvimento ágil de *software* oferece uma solução viável quando o produto a ser desenvolvido tem requisitos difusos ou em alteração, sendo capaz de

lidar melhor com ambientes instáveis e de constantes mudanças. Os métodos ágeis provaram ter maior flexibilidade, comparado com o desenvolvimento tradicional, sendo usado para entregar *software* em um tempo mais curto. O Quadro 1 compara os valores formulados pelo manifesto com atividades das práticas tradicionais. Na primeira coluna são destacados os processos com maior relevância no desenvolvimento ágil, e na segunda as atividades onde a ênfase é mais comum nos processos tradicionais de desenvolvimento de *software*. Juntas, as linhas da tabela, representam os valores do Manifesto Ágil (2001).

Quadro 1 - Valores ágeis. Fonte: Adaptado de Karlstrom (2006)

Desenvolvimento Ágil	**Desenvolvimento Tradicional**
Indivíduos e interações mais que	Processos e ferramentas
Software funcionando mais que	Extensa documentação
Colaboração com o cliente mais que	Negociação de contrato
Responder às mudanças mais que	Seguir um plano

Embora os métodos ágeis sejam efetivos em alguns contextos, conforme ilustrados nos parágrafos anteriores, os produtos de *software* grandes e complexos, geralmente requerem maior disciplina e gestão das atividades que compõe o ciclo de desenvolvimento da solução. O planejamento ágil é um processo relativamente informal e a aplicabilidade desses métodos em ambientes complexos é considerada desafiadora. A implementação incremental de práticas ágeis no desenvolvimento de sistemas complexos, leva a benefícios em uma parte do processo e ao mesmo tempo levanta sérios problemas em outra parte. Por exemplo, o uso de equipes pequenas e auto gerenciáveis aumenta o controle sobre o projeto, mas leva a questões no nível de gerenciamento e coordenação de múltiplas equipes (SOUNDARARAJAN, 2009; PETERSEN et al. 2009).

Por mais efetivo que as práticas ágeis têm se destacado em projetos de produção de *software*, é notório que para domínios de aplicações específicos, faz-se necessário o complemento de alguns processos com práticas mais efetivas. É com base nessa afirmação que alguns estudos referenciam outras abordagens em complemento às práticas ágeis. Wang (2012) acopla técnicas e/ou ferramentas *Lean* com o propósito de melhorar a utilização das metodologias ágeis de desenvolvimento de *software.*

3.3.3 Desenvolvimento *Lean* de *software* (LSD)

Se o desenvolvimento de *software* não é feito como em uma fábrica de produção, como aplicar os princípios da produção enxuta para esse contexto? Esse questionamento, feito por Poppendieck (2012), teve a seguinte resposta em seu trabalho: *Lean* é considerado um conjunto de princípios em vez de práticas. A sua aplicação no desenvolvimento de produtos de *software* faz total sentido e pode levar a melhorias de processo e qualidade. A partir desse conceito, no início dos anos 2000, Mary e Tom Poppendieck (2012) transferiram os princípios da filosofia *Lean* para o ambiente de desenvolvimento de *software* e chamou-lhes de *Lean Software Development* (LSD).

A primeira tradução dos conceitos *Lean* para o contexto de desenvolvimento de *software*, foi a interpretação dos princípios da filosofia enxuta, apresentados no Quadro 2.

Quadro 2 - Princípios do LSD. Fonte: Adaptado de Poppendieck (2012)

Princípio	Descrição
Eliminar desperdício	Em *software*, entende-se como desperdício, funcionalidades não usadas, produtos inacabados, entre outras.
Integrar qualidade	Inspeção do produto para que os defeitos sejam identificados imediatamente após ocorrerem. Quando um defeito é encontrado, deve ser corrigido imediatamente.
Criar conhecimento	É importante ter um processo de desenvolvimento que encoraje o aprendizado contínuo e sistemático.
Adiar comprometimentos	As tomadas de decisões devem ser planejadas, a fim de tomar decisões irreversíveis no último momento possível.
Entregar rápido	Entregar tão rápido ao ponto de não haver mudanças.
Respeitar as pessoas	É essencial a preocupação com pessoas durante o processo de produção.
Otimizar o todo	Uma organização *Lean* otimiza todo o fluxo de valor, do momento que recebe o pedido até que o *software* seja implantado e a necessidade do cliente atendida.

A aplicabilidade das ferramentas *Lean*, junto ao contexto de desenvolvimento de *software*, se resumem a prover formas de atingir um ou mais dos princípios ilustrados no Quadro 2, principalmente, aquele que se refere à eliminação de desperdício. Desperdício é qualquer coisa feita que não adiciona valor ao cliente ou qualquer atraso que o impeça de obter valor quando desejado. Os sete desperdícios da filosofia *Lean* foram traduzidos para o contexto de desenvolvimento

de *software*, conforme ilustrado no Quadro 3. Na coluna à esquerda, é apresentado os desperdícios já conhecidos da filosofia *Lean,* já aqueles ilustrados na coluna da direita, referem-se à tradução proposta por Poppendieck (2003).

Quadro 3 - Tradução dos 7 desperdícios. Fonte: Adaptado de Poppendieck (2003)

Produção	**Desenvolvimento de Software**
Superprodução	Funcionalidades extras: Do mesmo modo que a superprodução é o pior dos 7 desperdícios da produção, uma funcionalidade sem valor para o cliente, é o pior dos desperdícios para desenvolvimento de *software*.
Transporte	Transferência de controle: A grande questão aqui, está em como minimizar a perda quando há transferência de conhecimento tácito entre colaboradores.
Estoques	Trabalho inacabado: O objetivo é partir do início do trabalho para um código integrado, testado, documentado e entregue, tudo isso, em um fluxo rápido e único.
Defeitos	Defeitos: Quanto mais cedo e mais frequente forem realizados os testes, menos a chance de defeitos serem encontrados pelos clientes em ambiente de produção.
Processamento adicional	Reaprendizagem: Redescobrir algo que sabíamos, porém, esquecemos é a definição de “retrabalho” no desenvolvimento de *software*.
Movimentação	Troca de tarefas: Executar múltiplas tarefas ao mesmo tempo, ao invés de executar uma a uma, entregando o seu valor o quanto antes.
Esperas	Atrasos: Esperar a disponibilidade de pessoas que estão trabalhando em outras áreas é um grande desperdício proveniente do atraso.

As técnicas e ferramentas da filosofia *Lean* buscam auxiliara execução de atividades relacionadas ao processo de produção na indústria de manufatura, sempre relacionadas aos princípios do desenvolvimento enxuto: (i) Identificar valor para o cliente; (ii) Identificar todas as etapas do fluxo de valor de cada produto, eliminado possíveis etapas que não agregam valor; (iii) Criar um fluxo sequencial para cada produto; (iv) Estabilizar e fazer com que o fluxo de valor seja puxado pelo cliente, a partir das suas reais necessidades; e (v) Garantir a contínua melhoria de todo o processo a fim de atingir o estado da perfeição, sem nenhum tipo de desperdício. Algumas ferramentas ou técnicas frequentemente utilizadas são:

- *Value Stream Mapping* (VSM): Esta ferramenta permite aglomerar todas as ações que acrescentam valor ao produto e todas as que não acrescentam, mostrando o percurso dos materiais desde o pedido até à entrega do produto final ao cliente (ROTHER, SHOOK, 2003);

- Gestão Visual: Ferramenta que disponibiliza informação em formato visual, com a finalidade de fornecer subsídio relacionados às atividades básicas diárias, estado das máquinas, instruções de manutenção, entre outras (LAZARIN, 2008);
- Trabalho Padronizado: Esta ferramenta é um sistema detalhado e visual que passa pela documentação prévia das etapas do processo, de forma a garantir que todos os operários sigam mesmo procedimento. Algumas vantagens desta ferramenta são: a redução de custos; o fato de favorecer na qualidade dos produtos com a redução dos defeitos; o aumento da produtividade, do tempo produtivo das máquinas; e a minimização de falhas através da organização do trabalho (BENETTI et al., 2007; PINTO, 2009);
- 5S: Representam uma ferramenta que procura reduzir os desperdícios e melhorar o desempenho das pessoas e dos processos. Concebe mudanças ao nível dos hábitos e do comportamento tanto da chefia como dos operários, tornando o ambiente de trabalho agradável, seguro e produtivo (SILVA, FRANCISCO, & THOMAZ, 2008).

Não diferente da filosofia *Lean*, o LSD se propõe à aplicação de ferramentas do desenvolvimento enxuto no contexto de produção de *software*, objetivando apoiar a concretização dos princípios *Lean* para desenvolvimento de sistemas de informação. O Kanban já difundido na manufatura, quando aplicado no contexto de desenvolvimento de *software*, apresenta resultados importantes como melhoria na qualidade das entregas, melhoria na comunicação e coordenação das equipes, diminuição de defeitos, entre outros ganhos (ABRAHAMSSON et al. 2010; AHMAD et al., 2013).

3.3.4 KANBAN

As abordagens *Lean* e Kanban foram introduzidas na manufatura japonesa em meados dos anos 50 por Taiichi Ohno.. Acevedo (2001) afirma que Kanban é uma ferramenta de gestão visual que apoia o processo de produção, auxiliando na criação de um fluxo contínuo baseando-se em metodologias de auto-gestão (SERNA, 2015). O princípio da ferramenta é prover que as atividades de processamento e montagem sejam desencadeadas somente a partir de sinais de demanda nas próximas etapas

do processo, criando uma linha de produção “puxada” e controlada (CHAI, 2008; ABRAHAMSSON, 2010; BECKER, 1988).

A aplicação do Kanban no contexto de desenvolvimento de *software* foi realizada originalmente em 2004 por Anderson (2010), quando uma pequena equipe de TI da Microsoft estava obtendo resultados insatisfatórios, e a implementação do método possibilitou melhorias significantes como consequência. A ferramenta Kanban, quando aplicada em um ambiente de desenvolvimento de *software,* permite com que a equipe visualize o fluxo de trabalho e limite as atividades em progresso em cada estágio do fluxo de trabalho. O quadro Kanban fornece visibilidade no processo de desenvolvimento de *software* pois mostra o trabalho atribuído a cada desenvolvedor, além de deixar claro as prioridades e destacar possíveis sobrecargas no fluxo de produção. A proposta de limitar as atividades em progresso é fazer com que cada membro da equipe se concentre apenas no desenvolvimento daquilo que fora planejado, focalizando seu esforço em poucas atividades ao mesmo tempo, produzindo assim um fluxo contínuo de funcionalidades liberadas para os clientes. O Quadro 4 ilustra os princípios do Kanban para o desenvolvimento de software (KNIBERG, 2012; ANDERSON 2010).

Quadro 4 - Princípios do Kanban. Fonte: Adaptado de Ahmad (2013)

	Princípios do Kanban
1	Visualizar o fluxo de trabalhos
2	Limitar o trabalho em andamento
3	Gerenciar o fluxo
4	Tornar as políticas de processo explícitas
5	Melhorar de forma colaborativa

Em um sistema Kanban, quando aplicado ao desenvolvimento de *software*, o fluxo de valor é mapeado por meio de um quadro, preferencialmente físico em uma parede. Para cada etapa do fluxo de trabalho são criadas colunas no respectivo quadro. Os cartões Kanban são colocados nas colunas representando o estado atual do trabalho. Quando a atividade cumpre as políticas especificadas para estar completa, o cartão é movido para a próxima coluna Ao longo do tempo cada cartão se move da esquerda para a direita em direção ao final do quadro. Para cada coluna é atribuída uma quantidade máxima de cartões que poderão estar em determinada

etapa do fluxo. As etapas do fluxo referem-se às atividades que agregam valor para o cliente ou o produto durante o processo de desenvolvimento (POPPENDIECK, 2012).

É esperada a produção de resultados positivos a partir da aplicação do Kanban em processos de desenvolvimento de *software*. A possibilidade de visualizar o fluxo de trabalho torna a equipe responsável por gerenciar suas atividades durante o fluxo, prevendo possíveis problemas relacionados às entregas para o cliente, assim sendo, ações preditivas podem ser gerenciadas. A limitação do trabalho em andamento, de acordo com a capacidade da equipe, equilibra a demanda de acordo com o rendimento do trabalho produzido pelos membros do time de desenvolvimento, entregando produtos de maior qualidade e melhorando o seu desempenho. O fluxo contínuo de produção de funcionalidades ajuda a reduzir o tempo de execução, levando à lançamentos regulares que ajudam na relação de confiança entre o cliente e a organização desenvolvedora (ANDERSON, 2010; AHMAD, 2013).

3.3.5 Metodologia

Este estudo trata-se de uma pesquisa de natureza aplicada com visões subjetivas (CAUCHICK, 2012). Devido ao caráter empírico do trabalho em questão, o procedimento utilizado na pesquisa é um estudo de caso.

A Figura 1 ilustra os passos e procedimentos executados durante à concepção do respectivo trabalho.

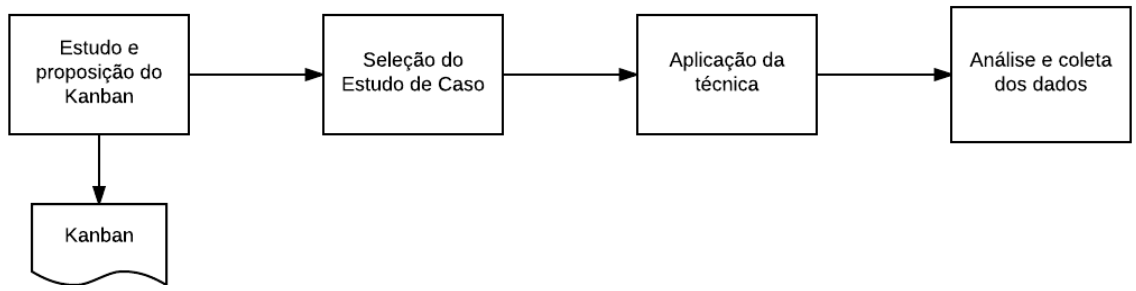

Figura 1 - Etapas para a execução do trabalho. Fontes: Dados da pesquisa

A primeira etapa, trata-se de um estudo sobre a utilização de quadros *Kanban* no contexto de desenvolvimento de *software,* objetivando a criação de um quadro baseado em estudos teóricos, cujo posteriormente será aplicado na prática. A proposição do quadro foi realizada com base nos trabalhos de Ladas (2009) e Anderson (2010), respectivamente: *Scrumban: Essays on Kanban Systems for Lean software development e Kanban: Successful Evolutionary Change for Your Technology Business.*

Depois que a proposição do quadro foi realizada, a próxima etapa foi a seleção do objeto para estudo do caso. O estudo foi realizado em um departamento de desenvolvimento de novos produtos. O quadro foi aplicado na equipe responsável por desenvolver e manter um produto de gestão de frentes de lojas de um supermercado. A equipe é composta por 8 membros, divididos entre 5 desenvolvedores e 3 testadores.

Após a seleção do estudo de caso, a próxima etapa foi a aplicação do quadro Kanban anteriormente concebido. A aplicação do estudo foi realizada durante 3 *Sprints*, cujas têm duração de 15 dias cada. Assim sendo, durante 45 dias, a equipe utilizou a nova proposição do quadro *Kanban*. Anteriormente, a gestão das tarefas em execução era feita em uma ferramenta online, chamada *IBM Jazz*. Nela eram cadastradas cada uma das funcionalidades e suas respectivas tarefas, onde cada colaborador era responsável por acessar a ferramenta e gerenciar, diariamente, o status de execução de cada.

Por fim, a análise e coleta dos dados foram feitas. Para a coleta dos dados, além do uso da ferramenta para obtenção de dados, foram realizadas entrevistas semiestruturadas com gestores e envolvidos no desenvolvimento da solução. Foram entrevistados os membros da equipe e o gerente do projeto em questão, cujo responde diretamente aos diretores da empresa.

3.3.6 Resultados

3.3.7 Estudo de caso

O objeto selecionado para a aplicação do estudo de caso é uma empresa desenvolvedora de *softwares* de grande porte, especialista em soluções para gestão de supermercados e materiais de construção. Com a matriz situada no sudoeste do estado do Paraná, a corporação conta com uma carteira de aproximadamente 3 mil clientes e cerca de 100 mil usuários ativos em cada uma de suas soluções.

O estudo de caso, ocorreu em uma das fábricas de produção de *software* da corporação, mais especificamente no setor de Novos Produtos. Este setor é responsável por analisar e desenvolver soluções inovadoras com o intuito de manter a empresa constantemente alinhada às tendências tecnológicas nos setores de atuação da corporação. A aplicação deste trabalho foi realizada especificamente na concepção e produção de um sistema para gestão de vendas em supermercados de

pequeno e médio porte, sistema esse, referenciado no contexto do trabalho como SGVS.

O SGVS é um *software* de gestão de vendas na parte de atendimento de supermercados. Criado para gerenciar as transações efetuadas nos pontos de venda de um supermercado, o SGS é um sistema novo. No momento da concepção deste trabalho estava em período de maturação para, posteriormente, ser colocado à venda junto aos demais produtos comercializados pela empresa. Esse período de maturação se faz importante afim de coletar dados para a sequência e evolução do sistema. “Para novos produtos, sempre há um período de amadurecimento antes de colocá-lo à venda. É assim que coletamos informações necessárias para sermos mais assertivos durante a construção do *software*. “, destacou o Gerente da Fábrica de Novos Produtos.

A Fábrica de novos produtos utiliza de metodologias e práticas ágeis para a concepção e elaboração dos seus produtos, mais especificamente a metodologia ágil chamada *Scrum*. Assim sendo, o desenvolvimento do SGVS ocorre de maneira iterativa e incremental. Cada ciclo de produção do sistema é composto de uma lista priorizada de novas funcionalidades. O tempo de execução de cada ciclo é de 15 dias, o qual échamados de *Sprint*. No momento deste estudo, 19 ciclos já haviam sido executados nos SGVS, incrementando um total de 86 funcionalidades. A empresa mantém as *Sprints* e as funcionalidades desenvolvidas em uma plataforma *online* chamada IBM Jazz. Nessa plataforma, foi possível coletar detalhes sobre as iterações e as funcionalidades que as compunham.

3.3.7.2 Concepção do quadro KANBAN

Kanban é definido como uma ferramenta ou uma técnica que objetiva representar uma parte da capacidade produtiva de um pedaço ou um departamento responsável pela produção de um produto qualquer. É uma forma de gerenciar projetos com mudanças mínimas nas atividades operacionais estabelecidas pelo time de desenvolvimento. Cada uma das atividades que compõe o projeto, são chamadas de cartões. Quando aplicado o conceito de Kanban, para o contexto de desenvolvimento de *software*, a técnica faz com que a equipe consiga visualizar o fluxo de trabalho e permita limitar o máximo de atividades em progresso em cada estágio do fluxo de trabalho. O uso do Kanban como ferramenta de gestão de projetos, é realizado a partir da publicação dos cartões em um quadro ilustrado em uma parede

física. No quadro, as tarefas de desenvolvimento são representadas por cartões, e os status são indicados por cada coluna que compõe o quadro. Quando a atividade está completa, o cartão é movido para a próxima coluna até dar-se como concluída ou pronta. O quadro é composto por cada uma das atividades necessárias para entregar as funcionalidades priorizadas para a referente *Sprint* (LADAS, 2009; HIRANABE, 2007).

Cada *Sprint* possui o seu respectivo quadro *Kanban*, o qual é projetado durante toda reunião de planejamento cujas ocorrem à cada 15 dias. Ao término da execução da iteração o quadro é desfeito com o intuito de prepara-lo para a próxima *Sprint*. As funcionalidades que serão desenvolvidas em cada ciclo são previamente identificadas, usando técnicas de priorização de demandas. Durante as cerimônias de planejamento, o time é responsável por estimar o esforço necessário para desenvolver e entregar cada uma das funcionalidades priorizadas, além de elicitar atividades necessárias para concluí-las. Cada funcionalidade e suas atividades são transpostas para cartões físicos e posteriormente adicionadas ao quadro Kanban. Cartões de funcionalidades são compostos por um nome e sua referente estimativa. Cartões de atividades são formados por uma descrição que possibilite a posterior relação com a funcionalidade. Durante a execução da *Sprint*, ambos os cartões (funcionalidades e atividades) são movidos entre cada uma das colunas do quadro (POPPENDIECK, 2003; LADAS, 2009; HIRANABE, 2007; KNIBERG, 2012; ANDERSON, 2010).

O quadro Kanban é formado por colunas que ilustram o status de execução de cada atividade e/ou funcionalidade. Ladas (2009), destaca que o status de cada atividade, difere de cada sistema, devido a particularidades determinadas pelas necessidades do projeto ou habilidades específicas da equipe de desenvolvimento, por exemplo. Para a concepção do quadro e posterior aplicação do estudo de caso, as seguintes colunas foram determinadas:

- A fazer: Refere-se ao primeiro status das funcionalidades e atividades. Enquanto nenhuma atividade e ou funcionalidade foi iniciada;
- (ii) Desenvolvendo: Atividades relacionadas ao processo de codificação, quando iniciadas, estarão posicionadas na respectiva coluna. Com relação às funcionalidades, enquanto houverem atividades, em

andamento, relacionadas ao desenvolvimento, a mesma permanecerá no status em questão;

- (iii) A testar: Assim que todas as atividades relacionadas à codificação de tal funcionalidade terminar, a mesma deverá mover-se para a coluna em questão. Enquanto as atividades pertinentes ao processo de testes não forem iniciadas, a funcionalidade permanecerá na respectiva coluna;
- (iv) Testando: Atividades relacionadas ao processo de testes, quando iniciadas, estarão posicionadas nessa coluna. Com relação às funcionalidades, enquanto houverem atividades, em andamento, relacionadas a testes, a mesma permanecerá no status em questão;
- (v) Pronto: Quando atividades forem concluídas, as mesmas deverão ser movidas para o status em questão. Para as funcionalidades, elas se enquadrarão na coluna em questão somente quando todas as atividades relacionadas a ela estiverem concluídas.

Durante a execução das atividades de testes é possível que defeitos sejam identificados. Os defeitos também são inseridos no quadro Kanban, e a sua navegabilidade entre as colunas acontece da mesma forma que as atividades. Enquanto estiver em processo de correção o cartão estará na coluna “Desenvolvendo”. Quando pronto para testar, o cartão será movido para o status: “A testar” e enquanto estiver em teste, estará posicionado na coluna: “Testando”. Quando inserido no quadro, o cartão do tipo defeito, será composto de uma descrição que o identifique e o relacione com a funcionalidade que o originou. A Figura 2 ilustra um exemplo de um quadro Kanban aplicado no desenvolvimento de *software*.

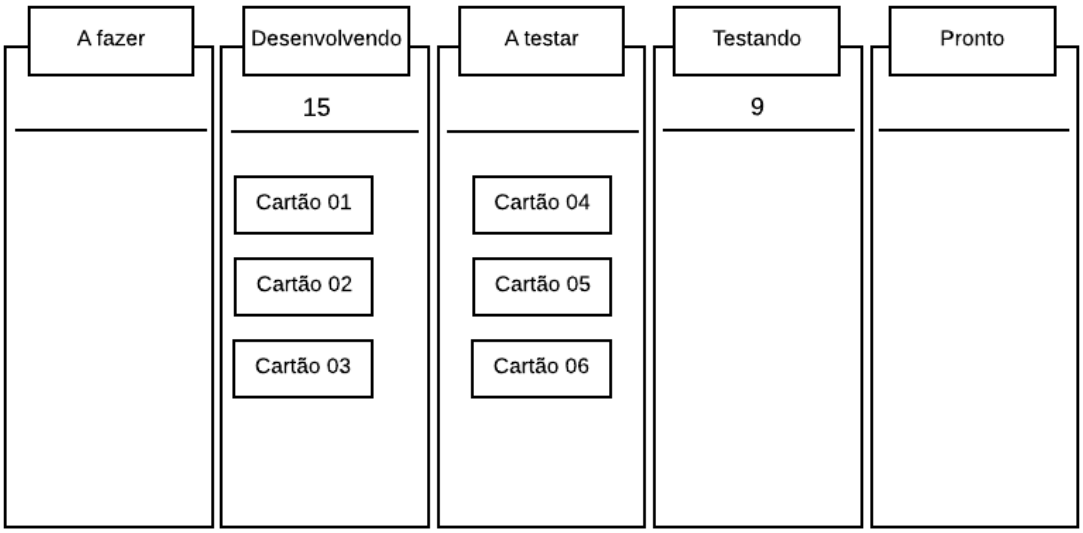

Figura 2 - Exemplo de quadro Kanban. Fonte: Adaptado de Ladas (2009)

Quando se refere a propriedade que o quadro Kanban possui de limitar o número máximo de atividades em progresso, Ladas (2009) justifica sua afirmação dizendo que desenvolvedores possuem a habilidade de trabalhar apenas em uma única atividade ao mesmo tempo, e mudar a execução para outra atividade ou recurso causará atrasos e afetará negativamente sua concentração e desempenho. Entretanto, isso não significa que os desenvolvedores deverão limitar-se à execução de apenas uma atividade por vez devido ao fato do processo de desenvolvimento de *software* ser um trabalho complexo. Algumas tarefas levam mais e outras menos tempo para serem concluídas, e em geral existem atrasos e dependências entre as atividades. O importante é limitar as tarefas atribuídas entre um número razoável que não produza o caos. O autor sugere um número máximo entre 3 e 5 atividades por vez para cada indivíduo.

Para o estudo de caso em questão, o quadro Kanban foi proposto e aplicado durante a execução de 3 ciclos de desenvolvimento: *Sprint* A, *Sprint* B e *Sprint* C, as quais foram compostas de: 10, 11 e 10 funcionalidades respectivamente, e tiveram a duração de 45 dias. As atividades pertinentes a cada funcionalidade serão apresentadas nas próximas seções. Com relação à limitação de atividades em desenvolvimento, foi considerado um máximo de 3 tarefas por vez para cada recurso, sugestão de Ladas (2009). A limitação se refere às atividades e não às funcionalidades, devido ao fato de elas serem objetos de execução prática. Dessa forma, um limite máximo de atividades no status de “Desenvolvendo” é de 15, devido

ao fato da equipe de desenvolvedores ser composta de 5 membros. Com relação à coluna de “Testando”, o limite de atividades é de 9 em virtude de a equipe de testadores ser composta de 3 membros.

3.3.7.3 Execução *Sprint* A

Durante a cerimônia de planejamento da *Sprint* A, o time de desenvolvimento, em comum acordo com os gestores do projeto do SGVS, projetaram o desenvolvimento e integração de 10 novas funcionalidades, as quais já haviam sido previamente priorizadas. No decorrer da reunião o time definiu quais seriam as atividades necessárias para concluir a entrega de cada uma das 10 funcionalidades previamente definidas. O Quadro 5 ilustra as funcionalidades e atividades relacionadas. É importante ressaltar que para garantir o sigilo das informações, o nome de cada funcionalidade foi substituído pelo acrônimo da palavra “Funcionalidade” seguido de um número sequencial único, por exemplo: Func 1, Func 2, Func 3 e assim sucessivamente, organizados de acordo com os itens mais prioritários. A descrição das atividades permaneceu as mesmas, entretanto a relação com o nome da funcionalidade que ela referência, foi alterada para a nova atribuição de nome.

Em geral, as atividades relacionadas com as funcionalidades referem-se a cada uma das principais disciplinas tangenciadas ao processo de desenvolvimento de *software*: Desenvolvimento e Teste. Em cenários específicos, fez-se necessário o desenvolvimento de testes automatizados. Trata-se de atividades executadas por testadores, objetivando auxiliar no processo de garantia de qualidade. Outras especificidades são atividades de arquitetura. Nessa, desenvolvedores projetam um modelo arquitetural antes de iniciar a codificação de tal funcionalidade, objetivando assim uma melhor qualidade nos códigos construídos.

Quadro 5 - Funcionalidades e Atividades *Sprint* A. Fonte: Dados da Pesquisa

Funcionalidade	**Atividade**
Func 1	Arquitetura Func 1
	Desenvolver Func 1
	Testar Func 1
	Automatizar Func 1
Func 2	Desenvolver Func 2
	Testar Func 2

- continua

- continuação

Func 3	Desenvolver Func 3
	Testar Func 3
Func 4	Desenvolver Func 4
	Testar Func 4
Func 5	Desenvolver Func 5
	Testar Func 5
Func 6	Desenvolver Func 6
	Testar Func 6
	Automatizar Func 6
Func 7	Desenvolver Func 7
	Testar Func 7
Func 8	Desenvolver Func 8
	Testar Func 8
Func 9	Desenvolver Func 9
	Testar Func 9
Func 10	Desenvolver Func 10
	Testar Func 10

Cada funcionalidade e suas respectivas atividades foram transcritas em cartões adesivos e posteriormente adicionadas ao quadro Kanban, o qual estava fixado em uma parede de fácil visualização do time e demais interessados nos resultados do projeto. Inicialmente todos os cartões foram atribuídos ao status de “A Fazer”. Em geral, a primeira atividade iniciada de cada funcionalidade é a de desenvolvimento, excepcionalmente quando há a necessidade de projetar um modelo arquitetural antes de iniciar a codificação. Foi o caso da Funcionalidade Func 1. Para esse caso, a primeira atividade iniciada foi: Arquitetura Func 1. Quando concluída, a próxima iniciada foi a atividade de desenvolvimento: “Desenvolver Func 1”. Depois de pronta, a funcionalidade (Func 1) foi movida do status “Desenvolvendo” para a coluna “A Testar”, até que a primeira atividade de teste fosse iniciada. Assim que a funcionalidade foi movida para o status “A Testar”, membros da equipe de teste, iniciaram ambas as atividades: “Testar Func 1” e “Automatizar Func 1”. Junto com as respectivas atividades, a funcionalidade foi movida também para a coluna “Testando”, até que todos os testes fossem encerrados. Somente assim tal incremento de *software* foi dado como pronto. Paralelo ao andamento de tal funcionalidade, os próximos 4

incrementos foram também iniciados, um para cada desenvolvedor. Assim que as atividades de desenvolvimento foram sendo encerradas, os teste e automações (quando necessário) também eram iniciadas.

O fluxo das atividades e funcionalidades não se repetiu de forma completa para as 10 funcionalidades planejadas para a *Sprint* A, o que ocasionou na não entrega dos incrementos previamente projetados. Aquelas não concluídas, foram planejadas para a próxima Sprint. Somente 6 itens foram entregues com sucesso no final da iteração (Func 1, Func 2, Func 3, Func 4, Func 5 e Func 6). Durante a execução da *Sprint*, defeitos foram sendo sinalizados no quadro Kanban, os quais eram sempre tratados com o mais alto nível de prioridade, ou seja, quando um testador encontrava um defeito referente a uma funcionalidade, o programador responsável pelo desenvolvimento da mesma, interrompia a execução de quaisquer atividades para priorizar a correção de tal defeito. Na *Sprint* A, foram encontrados e corrigidos um total de 63 defeitos. Comparado à quantidade de funcionalidades planejadas para a Sprint, o número de defeitos encontrados é considerado alto, em média 6 defeitos para cada uma. O tempo gasto corrigindo tais defeitos, é um tempo que poderia ter sido despejado na execução de atividades planejadas para a Sprint.

Ao término da *Sprint* A, durante a reunião de retrospectiva onde se discute sobre os principais resultados da iteração, o time foi indagado por membros da gestão acerca das causas que levaram ao não cumprimento do planejado. Diversas afirmações foram dadas como possíveis causas do atraso. Destaque dado para: (i) Número alto de defeitos; (ii) Complexidade subestimada de algumas funcionalidades; (iii) Sobrecarga de atividade para teste.

Como melhorias para a próxima *Sprint* as seguintes ações foram sugeridas: Atribuição de cores distintas para cartões do tipo funcionalidade, atividade e defeito. A proposta visava melhorar a visualização e gestão do fluxo de atividades, bem como deixar facilmente explícito o status do andamento de cada iteração. Foi sugerido também a criação de testes automatizados para um número maior de funcionalidades. Para a *Sprint* em questão, das 10 funcionalidades planejadas, somente para duas haviam sido previstas atividades de automação. Com essa proposta, procura-se melhorar o processo de teste e identificação de defeitos.

3.3.7.4 Execução *Sprint* B

Na reunião de planejamento da *Sprint* B, projetou-se o desenvolvimento e integração de 11 funcionalidades, sendo que quatro delas se tratavam de vestígios da *Sprint* anterior inacabada. Ainda no decorrer da reunião o time definiu quais seriam as atividades necessárias para concluir a entrega de cada uma das 11 funcionalidades previamente definidas. O Quadro 6 ilustra as funcionalidades e atividades relacionadas.

Quadro 6 - Funcionalidades e Atividades *Sprint* B. Fonte: Dados da Pesquisa

Funcionalidade	**Atividade**
Func 1	Testar Func 1
	Automatizar Func 1
Func 2	Testar Func 2
	Automatizar Func 2
Func 3	Testar Func 3
	Automatizar Func 3
Func 4	Desenvolver Func 4
	Testar Func 4
	Automatizar Func 4
Func 5	Desenvolver Func 5
	Testar Func 5
	Automatizar Func 5
Func 6	Desenvolver Func 6
	Testar Func 6
	Automatizar Func 6
Func 7	Desenvolver Func 7
	Testar Func 7
	Automatizar Func 7
Func 8	Desenvolver Func 8
	Testar Func 8
Func 9	Desenvolver Func 9
	Testar Func 9
Func 10	Desenvolver Func 10
	Testar Func 10
Func 11	Desenvolver Func 11
	Testar Func 11
	Automatizar Func 11

Cada funcionalidade e respectivas atividades foram transcritas em cartões adesivos de cores distintas (atendendo sugestões da *Sprint* anterior) e adicionadas ao quadro Kanban. Inicialmente, todos os cartões foram atribuídos ao status de “A Fazer”. As 4 primeiras funcionalidades, foram vestígios da *Sprint* A e dessa forma algumas atividades relacionadas a elas já haviam sido concluídas anteriormente. É o caso da “Func 1”, “Func 2” e “Func 3”, as quais já tiveram suas atividades de desenvolvimento executadas e prontas na iteração passada. Assim sendo, as 3 primeiras funcionalidades e respectivas atividades de testes iniciaram na coluna “A Testar”. Os recursos de teste iniciaram a execução das atividades e as movera, na sequência, para o status “Testando”. Ainda sobre essas primeiras funcionalidades, conforme sugerido na retrospectiva da *Sprint* A, foram criadas atividades de automação de testes para cada uma. A maior parte das funcionalidades planejadas para a iteração em questão contemplaram a execução de atividades de automação de teste. Oito das onze funcionalidades tiveram seus testes automatizados. Os 3 incrementos que não contemplam a execução de testes automatizados, referem-se à integração com equipamentos físicos, os quais impossibilitam a execução de tais testes.

Das 11 funcionalidades planejadas para a *Sprint* B, 8 foram entregues com sucesso no final da iteração (Func 1, Func 2, Func 3, Func 4, Func 5 e Func 6, Func 7 e Func 8). Além dos incrementos entregues na *Sprint* B, foram encontrados e corrigidos um total de 45 defeitos. Durante a reunião de retrospectiva, mais uma vez, o time foi indagado por membros da gestão, acerca das causas que levaram ao não cumprimento do planejado. Os possíveis motivos foram destacados pelo time: (i) Defeitos de funcionalidades já entregues anteriormente; (ii) Não observação do status da iteração durante a *Sprint*.

Sobre os defeitos corrigidos, os quais foram originados a partir do desenvolvimento de funcionalidades planejadas em outras *Sprints*, ficou evidente que se tratava do reflexo da criação dos testes automatizados na iteração corrente, ou seja, os testes descobriram defeitos originados em outras *Sprints*, entretanto, devido a não terem sido automatizados antes, não foi possível identificá-los. A sua correção foi vista com melhores olhos a partir dessa análise.

A atribuição de cores distintas para cartões do tipo funcionalidade, atividade e defeito, possibilitou uma análise rápida e preditiva durante a execução da *Sprint*.

Faltando 3 dias para o encerramento da iteração, o gerente do projeto do SGVS visualizou no quadro Kanban que ainda haviam 3 funcionalidades no status “Desenvolvendo”. De forma clara, foi possível prever que as mesmas não seriam concluídas e integradas a tempo, levando em consideração que ainda precisariam ser automatizadas e testadas, sendo que, outras 5 atividades ainda estavam no status “Testando”.

Como melhoria para a próxima *Sprint* foi sugerido que se faça a determinação de uma data limite para o desenvolvimento de funcionalidades. Com essa proposta pretende-se, diminuir o gargalo na fase de teste perto do final da *Sprint*. Além disso, foi sugerido que as reuniões diárias, realizadas com todos os membros do time objetivando melhorar a comunicação e eliminar empecilhos, ocorra em frente ao quadro Kanban. Essa proposta pretende fazer com que o time melhor identifique e gerencie o fluxo de suas atividades.

3.3.7.5 Execução *Sprint* C

No planejamento da *Sprint* C, o time, em comum acordo com os gestores do projeto do SGVS, propuseram o desenvolvimento e integração de 10 funcionalidades, sendo que 3 delas se tratavam de vestígio da *Sprint* passada. O time definiu também quais seriam as atividades necessárias para concluir a entrega de cada uma das 10 funcionalidades previamente definidas. O Quadro 7 ilustra as funcionalidades e atividades relacionadas.

Quadro 7 - Funcionalidades e Atividades *Sprint* C. Fonte: Dados da Pesquisa

Funcionalidade	**Atividade**
Func 1	Testar Func 1
Func 2	Testar Func 2
Func 3	Desenvolver Func 3
	Testar Func 3
	Automatizar Func 3
Func 4	Desenvolver Func 4
	Testar Func 4
	Automatizar Func 4
Func 5	Desenvolver Func 5
	Testar Func 5
	Automatizar Func 5

- continua

- continuação

Func 6	Desenvolver Func 6
	Testar Func 6
	Automatizar Func 6
Func 7	Desenvolver Func 7
	Testar Func 7
	Automatizar Func 7
Func 8	Desenvolver Func 8
	Testar Func 8
	Automatizar Func 8
Func 9	Desenvolver Func 9
	Testar Func 9
	Automatizar Func 9
Func 10	Desenvolver Func 10
	Testar Func 10
	Automatizar Func 10

Cada funcionalidade e respectivas atividades foram adicionadas ao quadro Kanban. Inicialmente, todos os cartões foram atribuídos ao status de “A Fazer”. As 3 primeiras funcionalidades foram as iniciadas na *Sprint* B e dessa forma algumas atividades relacionadas a elas já haviam sido concluídas anteriormente. É o caso da “Func 1” e “Func 2”, as quais já tiveram suas atividades de desenvolvimento executadas e prontas na iteração anterior. Assim, as primeiras funcionalidades e respectivas atividades de testes iniciaram na coluna “A Testar”, até que os recursos de teste iniciassem a execução das atividades e as movessem para o status “Testando”. Conforme sugerido na iteração anterior, durante o planejamento da *Sprint* em questão, todo o time em consenso definiu uma data limite para encerrar a execução de atividades de desenvolvimento tendo como propósito prover tempo suficiente para a execução de todas as atividades de testes. O ponto de corte para o desenvolvimento ficou estabelecido em 3 dias antes do encerramento da referente *Sprint*.

Das 10 funcionalidades planejadas para a *Sprint* C, todas elas foram entregues com sucesso no final da iteração. Além dos incrementos entregues foram encontrados e corrigidos um total de 22 defeitos. Sobre a entrega de tudo aquilo que fora planejado, além da considerável diminuição dos defeitos, o time atribuiu as

seguintes possíveis causas: (i) Automatização de grande parte dos testes; (ii) Respeito e definição de um ponto de corte de desenvolvimento; (iii) Auto gerência do fluxo das atividades; (iv) Colaboração entre os membros para o cumprimento do planejado.

Sobre a utilização de uma data limite para execução das atividades relacionadas ao processo de codificação, o time observou um ganho considerável. Foi possível observar que a equipe de teste não ficou sobrecarregada com as atividades da *Sprint*. Outra análise realizada a partir da implantação de um ponto de corte de desenvolvimento foi que, integrantes do time de desenvolvimento auxiliaram, quando necessário, na execução de atividades relacionadas ao teste, principalmente nas atividades de automação dos mesmos.

3.3.8 Análise dos resultados

A partir das análises realizadas pelo próprio time, durante a cerimonia de retrospectiva, realizada ao término de execução de cada Sprint, ficou evidente que os membros, foram aprendendo a trabalhar com o quadro Kanban, e com o passar das iterações, utilizando ele para auxiliar na gestão de suas atividades diárias e prever possíveis atrasos.

Ao término da execução da primeira Sprint, o time observou um número muito alto de defeitos encontrados na própria iteração. Com base nisso, foi sugerido que os defeitos fizessem parte do quadro Kanban com uma cor diferente das demais atividades que compõe o quadro, auxiliando na gestão do fluxo de atividades. O ganho obtido a partir da distinção de atividades e defeitos no Kanban foi identificado na Sprint B, onde foi destacado uma maior rapidez em realizar análises preditivas a cerca do andamento da iteração.

Na Sprint B, o número de defeitos encontrados foi menor que a anterior (Sprint A), reflexo também, de uma melhora na distribuição visual dos cartões que refletem atividades de desenvolvimento, testes e defeitos. Ainda nessa iteração, os membros do time evidenciaram que as atividades de desenvolvimento estavam ficando concluídas muito próximo do término da Sprint, o que gerava um gargalo na disciplina de teste. Essa análise foi possível de ser feita durante a execução da Sprint devido a gestão visual proporcionada pelo quadro Kanban. Sobre essa problemática,

foi proposto a criação de uma data limite para o desenvolvimento de funcionalidades, propiciando uma melhora no fluxo de entrega de funcionalidades.

Na última Sprint analisada, todas as funcionalidades planejadas foram concluídas durante a iteração, ganho esse, proporcionado, devido à diminuição considerada de defeitos. Nessa Sprint, foram encontrados 22 defeitos. Assim sendo, o tempo que anteriormente era gasto com a resolução de defeitos, foi despejado no desenvolvimento e entrega das funcionalidades planejadas. Outra evidência que proporcionou o sucesso da execução da Sprint foi a gerência do fluxo das atividades. Todo dia, durante a execução da iteração, a reunião diária ocorria em frente ao quadro Kanban, o que proporcionou o auxílio da execução de todas as atividades pertencentes à Sprint. Em casos onde identificado um atraso na entrega de alguma atividade, os membros do time se ajudavam na execução de tal tarefa, com o intuito de cumprir com o planejado. O Quadro 8 sintetiza os ganhos identificados a partir da utilização de tal quadro Kanban.

Quadro 8 - Melhorias no processo com a implementação do Kanban. Fonte: Dados da pesquisa

Atributos do processo	Sprint identificada	Ganhos proporcionados
Cartões Kanban com cores diferentes para tarefas e defeitos	B e C	Análise rápida e preditiva sobre o status da *Sprint*
Reuniões diárias em frente ao quadro Kanban	C	Auto gerência do fluxo das atividades (Multitarefa)
Limitar o número de itens em progresso	A, B e C	Identificação prévia do status da *Sprint*
Quadro Kanban visível para todos os envolvidos	A, B e C	Acompanhamento do status da *Sprint*

3.3.9 Considerações finais

Como objetivo do trabalho, foi procurado aplicar o uso do *Kanban* em uma fábrica de desenvolvimento de *software* de grande porte, objetivando reduzir o desperdício com trabalhos inacabados e excesso na produção. Foi procurado respostas para o seguinte problema de pesquisa: Como gerenciar as atividades de desenvolvimento de *software* de grande porte para que desperdícios com estoque e produtos inacabados sejam minimizados?

A partir da projeção e aplicação da ferramenta durante o desenvolvimento de novas funcionalidades para um sistema de grande porte, pode-se observar ganhos relevantes, principalmente relacionados à redução de desperdícios com as atividades

de desenvolvimento de *software*. Alguns desperdícios foram consideravelmente minimizados, como no caso da diminuição dos defeitos produzidos durante as *Sprints*. A melhoria somente foi possível de ser feita a partir do momento que ficou evidente o alto índice de defeitos que estavam sendo produzidos em cada iteração. A gestão visual, implementada com o sistema Kanban, permitiu a visualização do problema e um planejamento foi realizado objetivando minimizar a situação recorrente. Outras melhorias constatadas foram relacionadas à gestão do fluxo de trabalho. A partir do momento que ficou evidenciado que estavam sendo planejadas atividades que não eram entregues dentro do período de tempo traçado, melhorias foram projetadas objetivando a diminuição de trabalhos inacabados e projeção de um fluxo contínuo de produção. A partir da melhoria contínua, na última *Sprint* analisada, nenhuma atividade planejada deixou de ser integrada ao produto final bem como as falhas foram sensivelmente reduzidas.

Resultado relevante que merece ser destacado em tal trabalho, foi a adaptação da ferramenta Kanban, comumente aplicada e conhecida no contexto da Engenharia de Produção e sistemas produtivos, para o cenário de desenvolvimento de *software*, assim sendo, o respectivo resultado poderá ser replicado em trabalhos futuros.

Como trabalho futuro, sugere-se um estudo sobre a aplicação de técnicas de melhoria contínua com o intuito de melhorar o processo de produção de *software*. Esse estudo é relevante, levando em consideração que grande parte dos resultados positivos obtidos nesse trabalho, somente foi possível a partir de propostas e ideias coletadas em cerimônias e reuniões de retrospectivas.

Estudos desta natureza tornam-se cada vez mais importantes a partir do alto índice de competitividade identificado no mercado de produção de *software* de grande porte. A identificação e eliminação de resíduos durante o processo de produção de sistemas, tende a ser cada vez mais necessário visando a sobrevivência em um mercado cada vez mais acirrado.

4 CONCLUSÕES FINAIS

O presente trabalho se propôs a reduzir alguns desperdícios identificados durante o processo de desenvolvimento de sistemas de grande porte, os quais possuem propriedades onde as metodologias ágeis, por si só, não oferecem respostas eficientes às determinadas particularidades. Inicialmente, procurou-se identificar o estado da arte de tal problemática, a fim de identificar as reais lacunas de pesquisa, além de habituar-se com os autores e periódicos mais relevantes do contexto em questão. A partir de uma revisão da literatura estruturada e sistemática ficou evidente a existência de lacunas no desenvolvimento de sistemas de grande porte, onde foi identificado trabalhos que citavam a importância da filosofia *Lean* aplicada às práticas ágeis, como resposta às problemáticas encontradas no desenvolvimento de *softwares* desse contexto. Outro resultado relevante obtido a partir dessa primeira etapa, foi a construção de um portfólio bibliográfico, cujo guiou grande parte dos referenciais dos demais trabalhos.

Com a problemática da dissertação identificada como a necessidade de reduzir desperdícios identificados em sistemas de grande porte com o propósito de torna-lo mais eficiente, o próximo passo foi selecionar e identificar alguns dos desperdícios identificados, (resultado da primeira etapa) e propor uma forma de reduzir o seu impacto. O propósito foi a redução de um dos desperdícios ditos como mais relevante na literatura: O problema com a superprodução de funcionalidades, ou seja, o desperdício com o desenvolvimento de funcionalidades que não agregam valor para o produto ou cliente. A proposta foi a realização de um estudo de caso em uma empresa desenvolvedora de *softwares* de grande porte, objetivando identificar a problemática na prática e coletar dados necessários para a proposição de um método para minimizar a sua ocorrência. Foram coletados dados sobre o desenvolvimento de um sistema de gestão de frentes de lojas de um supermercado e comparado com o resultado da aplicação de um método multicritério. O trabalho evidenciou que o cliente do *software* desconhecia cerca de 35% das funcionalidades já desenvolvidas e integradas ao produto final, ou seja, a priorização de tais funcionalidades fora feita de forma equivocada, o tempo despendido com o desenvolvimento de tais, poderiam ter sido melhor aplicados em outras atividades que agregariam maior valor ao produto final.

Com as demandas do desenvolvimento de um sistema de grande porte já priorizadas, o próximo passo foi a gestão das atividades inerentes ao processo de desenvolvimento de *software* de grande porte. Tal processo se tornou imperativo devido à necessidade, identificada na literatura e na prática, de reduzir o desperdício com o desenvolvimento inacabado de funcionalidades. Além da proposta em reduzir tal desperdício, criou-se um fluxo de desenvolvimento puxado e contínuo. O trabalho se resumiu à projeção de um quadro Kanban, e colocá-lo em prática por meio de um estudo de caso em uma empresa desenvolvedora de sistemas de grande porte. Depois de 45 dias em prática, bons resultados foram observados: Rápida resposta às mudanças identificadas durante a execução de uma Sprint; Auto gerência do fluxo das atividades inerentes ao processo de produção; Entre outras. Uma consequência obtida com a implantação de tal ferramenta, foi a diminuição no número de defeitos encontrados durante a execução de uma iteração.

Por fim, as evidencias coletadas a partir da aplicação de ferramentas e técnicas *Lean* durante o desenvolvimento de sistemas de grande porte, se fizeram eficazes e auxiliaram no propósito final de entregar e desenvolver produtos de software de maneira mais eficaz. Destaca-se como lacunas do respectivo trabalho, a realização de múltiplos estudos de caso objetivando comprovar tai teorias destacadas nessa pesquisa.

5 REFERÊNCIAS

ABDELHAMID, T. Lean Production Paradigms in the Housing Industry. NSF Housing Research Agenda Workshop. Florida, EUA. 2004.

AHMAD, M. O. MARKKULA, J. OVIO, M. Kanban in software development: A systematic literature review. Euromicro Conference Series on Software Engineering and Advanced Applications. 2013.

ALMEIDA, A. T. Gestão de manutenção na direção da competitividade. Editora Universitária UFPE. 2003.

ALMEIDA, A. T.; COSTA, A. P. C. S. Aplicações multicritério de apoio à decisão. Recife UFPE. 2003.

ANDERSON, D. J. Kanban: Successful Evolutionary Change for Your Technology Business. Sequim, WA: Blue Hole Press. 2010.

BAPTISTA, G. L. VANALLE, R. M. SALLES, J. A. A. A Software Development Process Model Integrated to a Performance Measurement System. IEEE Latin America Transactions. 2015

BASKERVILLE, R. RAMESH, B. LEVINE, L. PRIES-HEJE, J h-speed software development practices: What works, what doesnt. IT Professional. 2006

BECK, K. Extreme Programming Explained. Addison Wesley, 2009.

BECK, K. eXtreme Programming Explained: Embrace Change. Addison-Wesley, San Francisco. 2000

BELLOMO, S., NORD, R.L. and OZKAYA, I. A study of enabling factors for rapid fielding combined practices to balance speed and stability, 35th International Conference on Software Engineering (ICSE), pp.982–991. 2013.

BENETTI, H. P., FILHO, J. I., SILIPRANDI, E. M., & SAURIN, T. A. Padronização do trabalho em uma fábrica de artefatos de cimento. XXVII Encontro Nacional de Engenharia de Produção. 2007.

BIFFL, S., AURUM, A., GRÜNBACHER, P., & BOEHM, B. Value Based Software Engineering. Springer. 2005.

BORTOLUZZI, S.C. et al. Avaliação de desempenho em redes de pequenas e médias empresas: Estado da arte para as delimitações postas pelo pesquisador. Estratégia & Negócios. V. 04, n. 02, p. 202-222, jun/dez. 2011.

CARIGNANO, C. G. Tomada de decisões em cenários complexos: introdução aos métodos discretos de apoio à decisão. São Paulo: Thompson Learning. 2005

CAUCHICK et al. Metodologia da pesquisa em engenharia de produção e gestão de operações. Rio de Janeiro: Elsevier. 2010.

CHAI, L. E-based inter-enterprise supply chain Kanban for demand and order fulfilment management, Emerging Technologies and Factory Automation. IEEE International Conference, pp.33-35. 2008.

CHARETTE, R.N. Challenging the Fundamental Notions of Software Development, Cutter Consortium, Executive Rep, p.4, USA. 2003.

CHAU, T., MAURER, F., MELNIK, G. Knowledge Sharing: Agile Methods vs. Tayloristic Methods. WETICE 03 Proceedings of the Twelfth International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises. IEEE Computer Society. 2003.

CHOW, T. CAO, D. B. A survey study of critical success factors in agile software projects. Journal of Systems and Software: 961-971. 2008.

COAD, P., Palmer, S. Feature-Driven Development. Prentice Hall, Englewood Cliffs, 2002.

COCKBURN, A. Agile Software Development. Addison-Wesley, UK. 2002

COCKBURN, A. Crystal Clear: A Human-Powered Software Development Methodology for Small Teams. Addison-Wesley, 2001.

COCKBURN, A.; HIGHSMITH, J. Agile Software Development: The Business of Innovation. IEEE Computer. Setembro, 2001.

COHEN, D., LINDVALL, M., COSTA, P. Agile Software Development, A DACS State-of-the-art report. Maryland: Fraunhofer center for experimental software engineering Maryland. 2003.

COHN, M. Agile Estimating and Planning, Addison-Wesley, NJ, 2005.

COHN, M. Succeeding with Agile Using Scrum, Addison-Wesley, Boston, 2009.

CONBOY K., Agility from First Principles: Reconstructing the Concept of Agility in Information Systems Development, Information Systems Research. 2009.

COOPER, Kendra, TIAN, Kun. Agile and Software Product Line Methods: Are They So Different?. APLE 1st International Workshop on Agile Product Line Engineering. 2006.

DA SILVA, I. F. DA MOTA S. N., P. A. OLEARY, P. DE ALMEIDA, E. S. DE LEMOS MEIRA, S. R. Agile software product lines: A systematic mapping study. Software - Practice and Experience: 899-920. 2011.

DINGSOYR T., S. Nerur, V. Balijepally, et al., A decade of agile methodologies: Towards explaining agile software development, Journal of Systems and Software. 2012.

DOMINGOS, Mainart; SANTOS, Ciro M. Desenvolvimento de Software: Processos Ágeis ou Tradicionais? Uma visão crítica. Teófilo Otoni, 2010.

DYBÅ, T. DINGSØYR, T. Empirical studies of agile software development: A systematic review. Information and Software Technology: 9-10. 2008.

EBERT, C. ABRAHAMSSON, P. OZA, N. Lean software development. IEEE Software: 22-25. 2012.

ENSSLIN, L., ENSSLIN, S. R., LACERDA, R. T. O. , TASCA, J. E. Processo de seleção de portfólio bibliográfico: Processo técnico com patente de registro pendente junto ao INPI. Brasil, 2010.

FOGELSTRÖM, N. D. SVAHNBERG, T. G. M. OLSSON, P. The impact of agile principles on market-driven software product development. Journal of Software Maintenance and Evolution: 53-80. 2010.

FREGA, J. F. Conflitos e Incertezas na Tomada de Decisão Gerencial de Custos: um Novo Olhar Sobre a Ampliação dos Limites da Racionalidade. 2008.

FUQUA AM, HAMMER JM. Embracing change: an XP experience report. Extreme programming and agile processes in software engineering, 4th International Conference, XP. Genova, Italy. 2003.

GRENNING J. Launching extreme programming at a process intensive company. IEEE Software 18(6):27–33. 2001.

HANSSON, C. DITTRICH, Y. GUSTAFSSON, B. ZARNAK, S. How agile are industrial software development practices?. Journal of Systems and Software: 1295-1311. 2005.

HIGHSMITH, J. Agile Software Development Ecosystems. Addison-Wesley Longman Publishing Co., Inc. 2002.

HIRANABE, K. Applied to software development: From agile to lean. 2007.

HOWELL, G. What is Lean Construction? 7th annual conference of IGLC, Califórnia, EUA, 1999.

JUNIOR, E. D. B., ENSSLIN, L., ENSSLIN, R. S. Proposta de processo para seleção, bibliometria e revisão sistêmica de artigos sobre a avaliação de desempenho na cadeia de suprimentos. Revista Produção Online, Florianópolis, SC, v.12, n. 4, p. 876-903, out./dez. 2012

KARLSTROM, D. Integrating agile software development into stage-gate managed product development. Empire Software Eng. 11: 203–225. 2006.

KETTUNEN, P. Adopting key lessons from agile manufacturing to agile software product development-A comparative study. Technovation: 408-422. 2009.

KNIBERG. H. How to make the most of both. 2012.

KUPIAINEN, E., MÄNTYLÄ V. M., ITKONEN J. Using metrics in Agile and Lean Software Development – A systematic literature review of industrial studies. Information and Software Technology. 2015.

LAANTI M. Agile methods in large-scale software development organisations. Applicability and model for adoption. University of Oulu. 2012.

LADAS, C. Scrumban: Essays on Kanban Systems for Lean software development. Seattle, WA, USA: Modus Cooperandi Press. 2009.

LARMAN C & Vodde B. Scaling Lean & agile development: Thinking and organisational tools for large-scale Scrum. Addison-Wesley Professional. 2008.

LAZARIN, D. F. Implementação de um sistema de gerenciamento visual em um ambiente de alta diversificação e baixo volume de produtos. 2008.

LEAL, F. Um diagnóstico do processo de atendimento a clientes em uma agência bancária através de mapeamento do processo e simulação computacional. Itajubá, MG, 2003.

LUNDAHL, L. On the choice of thesis format and on writing the kappa of a thesis of publications. 2015.

MARAFON, A. D., ENSSLIN, L., LACERDA, R. T., & ENSSLIN, S. R. Avaliação de desempenho na gestão de P&D – revisão sistêmica literária. P&D em Engenharia de Produção, 10(2), pp. 171-194, 2012.

MARCONI, M. A.; LAKATOS E. M. Fundamentos de Metodologia Científica. 6. Ed. São Paulo: Atlas. 2007.

MCDAID, K. Agile release planning: dealing with uncertainty in development time and business value, in: ECBS, Proceedings of the 15th, Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems. 2008.

MIDDLETON, P. JOYCE, D. Lean software management: BBC worldwide case study. IEEE Transactions on Engineering Management: 20-32. 2012.

MIDDLETON, P., FLAXEL, A., COOKSON, A. Lean software management case study: Timberline Inc. Proceedings of the 6th International Conference on Extreme Programming and Agile Processes in Software Engineering (XP), pp.1–9. 2005.

MISHRA, D. MISHRA, A. Complex software project development: Agile methods adoption. Journal of Software Maintenance and Evolution: 549-564. 2008.

MISHRA, D. MISHRA, A. Complex software project development: Agile methods adoption. Journal of Software Maintenance and Evolution: 549-564. 2011.

MISRA, S. C. KUMAR, V. KUMAR, U. Identifying some important success factors in adopting agile software development practices. Journal of Systems and Software: 1869-1890. 2009.

MOZZATO, A. R; GRZYBOVSKI, D. Análise sistêmica como Técnica de Análise de Dados Qualitativos no Campo da Administração: Potencial e Desafios. Revista de Administração Contemporânea, Curitiba, v. 15, n. 4, pp. 731-747, Jul./Ago. 2011.

MURRU O., DEIAS R., MUGHEDDUE G. Assessing XP at a European internet company. IEEE Software 20(3):37–42. 2003.

MUSAT, D., & RODRÍGUEZ, P. Value Stream Mapping integration in Software Product Lines. Proceedings of the 11th International Conference on Product Focused Software (pp. 110-111). New York: Association of Computing Machinery. 2010.

NERUR, S. MAHAPATRA, R. MANGALARAJ, G. Challenges of migrating to agile methodologies. Communications of the ACM: 72-78. 2005.

NIVOIT, J. B. Henri. Issues in Strategic Management of Large-Scale Software Product Line Development. Composite Information Systems Laboratory (CISL). Massachusetts Institute of Technology Cambridge. 2013.

OHNO, T. Toyota Production System: Beyond Large-Scale Production. Productivity Press, 1988.

PERNSTÅL J. FELDT R. GORSCHEK T. The Lean gap: A review of Lean approaches to large-scale software systems development. Journal of Systems and Software: 2797 – 2821. 2013.

PERRY, E. D. Parallel Changes in Large-Scale Software Development: An Observational Case Study. ACM Transactions on Software Engineering and Methodology, Vol. 10, No. 3, July 2001, Pages 308–337. 2001.

PETERSEN, K. WOHLIN, C. Measuring the flow in Lean software development. Software - Practice and Experience: 975-996. 2011.

PETERSEN, K. WOHLIN, C. Software process improvement through the Lean Measurement (SPI-LEAM) method. Journal of Systems and Software: 1275-1287. 2010.

PINHEIRO F, P, A. Abordagem MCDA como ferramenta de mudança de paradigma no planejamento dos recursos hídricos. Universidade Estadual de Campinas. Faculdade de Engenharia Civil, Arquitetura e Urbanismo. 2008.

POOLE C, HUISMAN JW. Using extreme programming in a maintenance environment. IEEE Software 18(6):42–50. 2001.

POPPENDIECK, M. CUSUMANO, M. A. Lean software development: A tutorial. IEEE Software: 26-32. 2012

POPPENDIECK, M. T. POPPENDIECK, Lean Software Development: An Agile Toolkit (The Agile Software Development Series). Addison-Wesley Professional, 2003

POPPENDIECK, M., & POPPENDIECK, T. Implementing Lean Software Development, from concept to cash. Boston: Pearson Education. 2007

POPPENDIECK, M., POPPENDIECK, T. Implementing Lean Software Development from Concept to Cash. Addison Wesley Professional. 2006

PRESSMAN, Roger S. Engenharia de Software. 5 ed. Porto Alegre: Bookman, 2002

QUMER, A., & HENDERSON-SELLERS, B. An evaluation of the degree of agility in six agile methods and its applicability for method engineering. Information and Software Technology, 50(3), 280-295. 2008.

RADNOR, Z., WALLEY, P. Learning to walk before we try to run: adapting lean for the public sector, Public Money and Management, Vol. 28, No. 1, pp.13–20. 2008.

RASMUSSEN J. Introducing XP into greenfield projects: lessons learned. IEEE Software 20(3):21–28. 2003.

RODRIGUEZ, F. C. Scrum e Agile em Projetos Guia Completo. Rio de Janeiro: Brasport. 2013.

RODRIGUEZ, P. Combining Lean thinking and agile software development: How do software-intensive companies use them in practice? University of Oulu Graduate School. 2013.

ROTHER, M., & SHOOK, J. Learning to see - Value Stream Mapping to Create Value and Eliminate Muda. Cambridge, Massachusetts: Lean Enterprise Institute. 2003.

ROY, B.; BERTIER, P. La méthode ELECTRE II. Paris: SEMA-METRA, 1971.

SALINAS, T. C. J, SEDENÕ, J., ESCALONA, M., MEJÍAS, M. Estimating, planning and managing Agile Web development projects under a value-based perspective. Information and Software Technology. 2014.

SCHUH P. Recovery, redemption, and extreme programming. IEEE Software 18(6): 34–40. 2001.

SCHWABER, K., Beedle, A. Agile Software Development with SCRUM. Prentice-Hall, 2002.

SERNA, M. D. A, ZAPATA, L. F. C, CORTES, J. A. Z. Mejoramiento de procesos de manufactura utilizando Kanban. Revista Ingenierías Universidad de Medellín. 2015.

SERRADOR, P. PINTO, J. K. Does Agile work? - A quantitative analysis of agile project success. International Journal of Project Management: 1040 – 1051. 2015.

SHARP H., ROBINSON H. An ethnographic study of XP practice. EMPIRICAL SOFTWARE ENGINEERING 9(4):353–375. 2004.

SHARP, H. ROBINSON, H. PETRE, M. The role of physical artefacts in agile software development: Two complementary perspectives. Interacting with Computers: 108 – 116. 2009.

SIDKY A, ARTHUR J. Determining the applicability of agile practices to mission and life-critical systems. Proceedings of the 31st IEEE Software Engineering Workshop. IEEE Computer Society. 2007.

SOMMERVILLE, I. Engenharia de Software. 9. ed. São Paulo: Pearson Prentice Hall. 2013.

SOUNDARARAJAN S, ARTHUR JD. A soft-structured agile framework for larger scale systems development. Proceedings of the 2009 16th Annual IEEE international Conference and Workshop on the Engineering of Computer Based Systems (14–16 April 2009). ECBS. IEEE Computer Society: Washington DC, 2009; 187–195. 2009.

STANDISH GROUP. Modernization: Clearing a pathway to success. The Standish Group International, Inc. 2010.

SUOMALAINEN, T. KUUSELA, R. TIHINEN, M. Continuous planning: An important aspect of agile and Lean development. International Journal of Agile Systems and Management: 132-162. 2015.

SUTHERLAND J. Scrum metrics for hyper productive teams: how they fly like fighter aircraft, in: HICSS, Proceedings of the 45th Hawaii International Conference on System Science. 2012.

TASCA, J. E., ENSSLIN, L., ENSSLIN, S. R., ALVES, M. B. M., An approach for selecting a theoretical framework for the evaluation of training programs. Journal of European Industrial Training, 2010.

THOMAZ, M. S., SILVA, N. P., FRANCISCO, A. C. A implantação do 5S na Divisão de Controle de Qualidade de uma Empresa Distribuidora de Energia do Sul do País: um estudo de caso. 2008.

TORRECILLA-SALINAS, C. J. SEDEÑO, J. ESCALONA, M. J. MEJÍAS, M. Estimating planning and managing Agile Web development projects under a value-based perspective. Information and Software Technology: 124-144. 2015.

TROJAN, F. Modelos Multicritérios para apoiar decisões na gestão da manutenção de redes de distribuição de água para a redução de custos e perdas. 2012.

VALLON, R. MÜLLER-WERNHART, M. SCHRAMM, W. GRECHENIG, T. Combination of agile and Lean in software development: Concept and realization. Informatik-Spektrum: 28-35. 2014.

VAN WAARDENBURG, G. VAN VLIET, H. When agile meets the enterprise. Information and Software Technology: 2154-2171. 2013.

VANHANEN J, KAHKONEN T. Practical experiences of agility in the telecom industry. Extreme programming and agile processes in software engineering, 4th International Conference, XP. Genova, Italy. 2003.

VILKKI K. When agile is not enough. In: Lean Enterprise Software and Systems. 2010.

VINCKE, P. Multicriteria Decision-Aid. John Wiley & Sons Ltd. 1982. ZELENY, M. Multiple Criteria Decision Making. New York: MacGraw-Hill. 1982.

WANG, X. CONBOY, K. CAWLEY, O. Leagile software development: An experience report analysis of the application of Lean approaches in agile software development. Journal of Systems and Software: 1287-1299. 2012.

WOMACK, D. T. JONES. D. Roos, The Machine That Changed the World : The Story of Lean Production. Harper Perennial. 1991.

WOMACK, D. T. JONES. D. Roos, The Machine That Changed the World : The Story of Lean Production. Harper Perennial, 1996.

WOMACK, J. P. & JONES, D.T. A Mentalidade Enxuta nas Empresas: Elimine o Desperdício e crie riqueza. 4 ed. Rio de Janeiro: Campus, 1991

YIN, R. K. Estudo de Caso – Planejamento e Métodos. 3. Ed. São Paulo: Bookman, 2005.

ZHI, J. GAROUSI-YUSIFOLU, V. SUN, B. GAROUSI, G. SHAHNEWAZ, S. RUHE, G. Cost, benefits and quality of software development documentation: A systematic mapping. Journal of System. 2015.

Apêndice A – Apresentação e resumo da etapa 01. Fonte: Dados da Pesquisa

PROCESSO ENXUTO PARA DESENVOLVIMENTO DE *SOFTWARES* DE GRANDE PORTE: UM MAPEAMENTO DA LITERATURA

Pedro Henrique de Alencar, Marcelo Gonçalves Trentin, Sandro Cesar Bortoluzz

(Universidade Tecnológica Federal do Paraná, Programa de Pós-Graduação em Engenharia de Produção e Sistemas. Via do Conhecimento, Km 1 CEP 85503-390 - Pato Branco - PR – Brasil. pedromachado@alunos.utfpr.edu.br)

Resumo: Devido ao avanço tecnológico dos dias atuais, empresas têm procurado aperfeiçoar cada dia mais seus processos e fluxos de desenvolvimento de *software*, eliminando possíveis desperdícios com o principal propósito de melhorar a qualidade de seus produtos finais. Uma metodologia comumente utilizada com essa finalidade são as práticas ágeis. Embora bons resultados já tenham sido observados a partir da sua implantação, algumas de suas abordagens têm sido cada vez mais questionadas quando o assunto é documentação, métricas, entre outras características essenciais para *softwares* de grande porte. Algumas ferramentas do ambiente de manufatura são especialistas em eliminar desperdícios melhorando assim as entregas dos produtos. Trata-se do pensamento enxuto, ou *Lean Thinking* cuja aplicação já é uma realidade nas indústrias em geral. Devido aos resultados positivos obtidos a partir da aplicação da filosofia *Lean* na manufatura, outros segmentos de negócio também estão implementando algumas das suas ferramentas, é o caso do *Lean* para desenvolvimento de *software*, ou *Lean Software Development* (LSD), o qual tem sido considerado como uma forma de ultrapassar as limitações dos processos ágeis para o contexto de desenvolvimento de sistemas de grande porte. Neste contexto, o presente estudo tem como propósito fazer uma análise bibliométrica e sistêmica de um portfólio bibliográfico alinhado ao tema de processo enxuto para o desenvolvimento de *software* de grande porte. O método utilizado para a construção e proposição do portfólio bibliográfico e posteriores pesquisas bibliométricas e sistêmicas foi o *Proknow-C, Knowledge Development Process-Constructivist* (ENSSLIN et al., 2010). Como resultado, os principais trabalhos compuseram o portfólio bibliográfico, evidenciando assim os periódicos, autores e palavras chave mais relevantes à temática em questão. Pôde-se evidenciar o periódico de maior destaque . A maior parte dos trabalhos que compuseram o portfólio bibliográfico, fizeram referência à algum tipo de estudo teórico (61%). Por fim, observou-se que o tema LSD ainda é um conceito recente. Alguns autores o definem como uma

metodologia ágil, outros como uma prática distinta dos demais modelos de desenvolvimento de *software*. Dessa forma, uma lacuna identificada nas pesquisas é a importância da realização de mais estudos empíricos que apliquem o LSD em contextos distintos, a fim de ilustrar os resultados obtidos a partir da sua utilização.

Palavras Chave: Análise sistêmica; Processo de desenvolvimento de *software*; *Lean Thinking*; *Software* de grande porte*; Lean Software Development*.

Apêndice B – Apresentação e resumo da etapa 02. Fonte: Dados da Pesquisa

REDUZINDO O DESPERDÍCIO DA SUPERPRODUÇÃO NO DESENVOLVIMENTO DE *SOFTWARE* DE GRANDE PORTE POR MEIO DA PRIORIZAÇÃO DE DEMANDA

Pedro Henrique de Alencar, Marcelo Gonçalves Trentin

(Universidade Tecnológica Federal do Paraná, Programa de Pós-Graduação em Engenharia de Produção e Sistemas. Via do Conhecimento, Km 1 CEP 85503-390 - Pato Branco - PR – Brasil. pedromachado@alunos.utfpr.edu.br)

Resumo:

A escolha das funcionalidades certas para o desenvolvimento e/ou incremento de um *software* de grande porte, é essencial para o sucesso na evolução e entrega de um sistema de informação, levando em consideração as específicas características que um produto dessa categoria possui. A atividade de escolher quais serão os próximos incrementos de um produto de *software* não é uma tarefa fácil. Muitas empresas erram no momento de selecionar as próximas evoluções dos seus produtos, e acabam entregando funcionalidades que nunca ou raramente serão utilizadas. Assim, os índices de insatisfação crescem por parte dos clientes e demais envolvidos. Este estudo propõe uma forma de priorizar as próximas demandas a serem consideradas para a evolução de um *software* de grande porte. Foi utilizado para isso o método de apoio à decisão multicritério Electre II. O objetivo do estudo é o de prover uma ordenação de funcionalidades a fim de identificar aquelas mais indicadas para serem consideradas nas próximas evoluções de um *software* de grande porte. Trata-se de priorizar as demandas, com o intuito de reduzir o desperdício com superprodução, com base na filosofia do *Lean Software Development.* Realizou-se um estudo de caso em um sistema de gestão de vendas de supermercados e lojas de varejo, que no momento avaliado estava em evolução. Foi realizada uma análise comparando os resultados da priorização, antes da aplicação do multicritério com aqueles obtidos depois. Os resultados preliminares ilustraram que 52% das funcionalidades priorizadas e desenvolvidas antes da aplicação do método foram julgadas como relevantes pelo método de apoio à decisão. Observa-se uma oportunidade de tornar o processo de priorização de demandas ainda mais efetivo.

Palavras Chave: *Electre* II. Multicritério. *Software* de grande porte. *Lean Software Development.*

Apêndice C – Apresentação e resumo da etapa 03. Fonte: Dados da Pesquisa

KANBAN COMO FORMA DE REDUÇÃO DE DESPERDÍCIO NO PROCESSO DE PRODUÇÃO DE *SOFTWARE* DE GRANDE PORTE

Pedro Henrique de Alencar, Marcelo Gonçalves Trentin
(Universidade Tecnológica Federal do Paraná, Programa de Pós-Graduação em Engenharia de Produção e Sistemas. Via do Conhecimento, Km 1 CEP 85503-390 - Pato Branco - PR – Brasil. pedromachado@alunos.utfpr.edu.br)

Resumo:

O sucesso de um projeto de *software* de grande porte depende de muitos fatores, entretanto, o mais comum dos motivos da falha de um projeto dessa magnitude é a falta ou a má gestão do processo de produção. Não importa o tamanho da equipe, ou a forma como os requisitos são documentados, se não há uma gestão eficiente do processo de produção, certamente o projeto falhará. Estudos emergentes sobre a gestão *Lean* de projetos de *software* de grande porte estão sendo realizados objetivando melhorar a gestão e os resultados dos projetos. Propõe-se a aplicação da ferramenta Kanban, no contexto de desenvolvimento de *software* de grande porte. O objetivo é prover uma forma de gerenciar o fluxo de entrada e saída das atividades pertinentes ao processo de produção de um *software* de grande porte, criando um fluxo balanceado e reduzindo o desperdício com superprodução de funcionalidades. Este trabalho trata-se de um estudo de caso em um sistema de gestão de vendas de supermercados e lojas de varejo que estava em evolução. Foi realizado a proposição de um quadro Kanban físico e colocado avaliado durante 45 dias. Depois do da implantação os resultados coletados foram analisados e mostraram um desempenho superior ao modelo digital, utilizado pela empresa antes do estudo em questão. Resultados ilustraram uma diminuição no número de funcionalidades inacabadas além da redução na quantidade de defeitos produzidos durante os ciclos de desenvolvimento do *software*. Isso somente foi possível a partir da gestão visual dos processos e atividades praticadas durante todo o fluxo de produção.

Palavras Chave: *Lean*. Kanban. *Software* de grande porte. *Agile*. Melhoria Contínua.