Universidade Federal Rural de Pernambuco

Departamento de Estatística e Informática

Bacharelado em Sistemas de Informação

Conjunto de Métricas para Suporte aos Princípios do Desenvolvimento Ágil de Software

Bruna Regina de Almeida Chagas

Recife

Março de 2014

Bruna Regina de Almeida Chagas

Conjunto de Métricas para Suporte aos Princípios do Desenvolvimento Ágil de Software

Orientadora: Teresa Maciel

Monografia apresentada ao Curso de Bacharelado em Sistemas de Informação da Universidade Federal Rural de Pernambuco, como requisito parcial para obtenção do título de Bacharel em Sistemas de Informação.

Recife

Março de 2014

À Deus,
Aos meus amados pais, Alba e José
À minha família
Ao meu namorado, Daniel
À minha orientadora, Teresa Marciel
Aos meus amigos.

Agradecimentos

Agradeço imensamente à minha mãe, Alba Valéria, que com toda simplicidade, amor e dedicação sempre me mostrou que seria através “dos estudos”que eu seria “alguém na vida”.

Ao meu pai, José Joaquim, que me mostrou o quanto a dedicação ao trabalho é importante.

Ao meu namorado, Daniel Barreto, pelo incentivo e palavras de conforto nos momentos mais difíceis.

À minha orientadora, Teresa Marciel, por toda a dedicação e apoio não apenas neste trabalho de conclusão de curso, mas também durante todas as suas aulas.

À minha avó, Daluz e a minha tia Ana que sempre acreditaram no meu potencial e pelos seus incentivos.

À minha sogra, Izabel, por mostrar todo o seu apoio.

À todos os meus amigos que se mostraram na torcida para que eu conseguisse realizar os meus sonhos.

Aos professores que fizeram parte da minha história, para aqueles que me apoiaram, guiaram e me incentivaram.

Resumo

O desenvolvimento ágil de software vem ganhando cada vez mais espaço nas organizações e se mostrado uma forma mais eficiente que as metodologias tradicionais. O *Lean* também ajuda a eliminar desperdícios e a focar no cliente. Ambas trazem uma nova visão no desenvolvimento de software e mostram-se como alternativas para que as empresas se tornem mais eficazes.

As métricas apresentam-se como um meio para entender o que acontece durante o desenvolvimento e também para a melhoria do projeto em qualquer etapa em que ele se encontre. Elas ajudam na visualização de possíveis problemas existentes e podem indicar futuras tendências, boas ou ruins, em cada fase do desenvolvimento. Mas, para isso, elas precisam estar alinhadas ao princípios ágeis propostos no Manifesto Ágil.

A finalidade do projeto é realizar um mapeamento entre as métricas ágeis, dentre elas as adotadas pelo *Lean*, para dar suporte aos doze princípios do Manifesto Ágil. Este estudo visa ajudar empresas e estudiosos da área a encontrar nas métricas o embasamento de que aqueles pontos mostrados no princípio estão realmente de acordo com o esperado. Em outras palavras, propiciar aos envolvidos com o desenvolvimento ágil de software o alinhamento dos princípios utilizando métricas para monitorar seus processos e o desenvolvimento propriamente dito.

Palavras Chave: *Lean*, Métricas Ágeis, *Lean* Métricas, Desenvolvimento Ágil, Princípios Ágeis

Abstract

The agile software development has been increasing in terms of space in the organizations and has been being a more efficient way than the traditional methodologies. Lean also helps eliminating wastes and focusing on the client. Both methodologies bring a new view of software development and are alternatives for the companies to become more effective.

The metrics are a means of understanding what happens during the development and also for the improvement of the project in any stage of it. The metrics also help visualizing the possible existent problems and can indicate future good or bad tendencies in each phase of the development. However, for tthis purpose, the metrics need to be aligned with the agile principles proposed by the Agile Manifesto.

The aim of the project is to perform a mapping among the agile metrics, including the ones adopted by Lean, in order to support the twelve principles of Agile Manifesto. This study aims at helping companies and scholars of the area to find in the metrics the basis that those points found in the beginning are really according to what is expected. In other words, to provide to the ones involded with the agile software development the alignment of the principles by using metrics to monitor their processes and the development itself.

Key Words: Lean, Agile Metrics, Lean Metrics, Agile Development, Agile Principles.

Sumário

Lista de Tabelas

Lista de Figuras

Lista de Siglas

ACM Association for Computing Machinery

CAPES Coordenação de Aperfeiçoamento de Pessoal de Nível Superior

CoV Coefficient of Variation

IEEE Instituto de Engenheiros Eletricistas e Eletrônicos

WIP Work in Progress

LSD Lean Software Development

CoV Coefficient of Variation

EBV Earned Business Value

AgileEVM Agile Earned Value Management System

ACPF Average Cost Per Function

SDI System Design Instability Metric

PSU Project Size Unit

NPV Net Present Value

IRR Internal Rate of Return

ROI Return on Investment

BTS Build-to-Schedule

OEE Overall Equipment Efficiency

LOC Lines of Code

KLOC Kilo Lines of Code

FTT First-Time-Through

RTF Running Tested Features

PO Product Owner

Capítulo 1

Introdução

Este trabalho de conclusão de curso apresentará um conjunto de métricas para apoiar os princípios do desenvolvimento ágil de software, propostos no Manifesto Ágil [14]. Para cada princípio será indicada(s) métrica(s) para suporte a sua definição. Neste capítulo serão expostos aspectos para contextualização da pesquisa realizada e a motivação para a realização da mesma. Outros pontos mostrados foram a questão de pesquisa, os objetivos, as contribuições e por último a estruturação do trabalho.

1.1 Contextualização

Os Métodos Ágeis promovem um processo empírico para o desenvolvimento de software. Essa abordagem exige um ciclo constante de inspeção, adaptação e melhoria. Descobrir formas eficazes de avaliar o processo e a equipe de desenvolvimento não é uma tarefa simples [43].

Em meio ao surgimento do termo engenharia de software, surgiram também inúmeros modelos, métodos e artefatos em alternativa ao desenvolvimento de software tradicional entre elas, as métricas, que vêem ganhando uma maior proporção e maturidade junto aos times de desenvolvimento, principalmente em relação à qualidade, abordagem que cada vez mais

desperta o interesse das empresas em busca da eficiência e maturidade.

Atualmente o desenvolvimento ágil de software vem ganhando destaque, vendo que o desenvolvimento tradicional se mostra um grande obstáculo para responder a mudanças, sejam elas alterações nos requisitos ou até mesmo o escopo por completo. O desenvolvimento ágil compreende várias metodologias ou *frameworks*, como o *Scrum*, XP (*eXtreme Programming*), *Lean Software Development*, *Kanban*, entre outros.

Percebe-se uma crescente utilização de metodologias ágeis por profissionais da área de desenvolvimento de softwares, afinal gerenciar as constantes mudanças que ocorrem durante um projeto não é uma tarefa trivial e as metodologias ágeis auxiliam na adaptabilidade a mudança nos projetos. Uma série de metodologias começaram a ser usadas, e a partir deste cenário os principais usuários destas metodologias e importantes nomes do desenvolvimento de software (Kent Beck, Mike Beedle, Arie Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland e Dave Thomas) se reuniram para formalizar estas metodologias na estação de esqui nos Estados Unidos, em Fevereiro de 2001 [17]. Em 2001 o nome métodos ágeis foi formalizado e o resultado desta reunião foi o acordo ágil, e a criação de todos os pontos crucias para o desenvolvimento ágil de software, incluindo os princípios do mesmo [14].

1.2 Motivação

Durante a terceira Conferência Internacional sobre *eXtreme Programming* ocorrida na Itália em 2002, Jim Johnson, então presidente do Standish Group, expôs, que em um software personalizado, cerca de 45% das funcionalidades implementadas nunca são utilizadas e 19% são raramente utilizadas [26]. Tom e Mary Poppendieck (2006), atestam que os dados apesentados por Jim Johnson, apesar de assustadores, tem se comprovado em levantamentos realizados com clientes e parceiros.

No decorrer de um projeto de desenvolvimento de software, as funcionalidades em excesso

são incluídas pelo cliente ou empresa desenvolvedora como forma de garantir que todas as necessidades dos clientes serão atendidas. Porém, desenvolver mais que o necessário resulta em um esforço desnecessário, logo, percebe-se o quão comum são os desperdícios em um processo de desenvolvimento de software [40]. Ou seja, Qualquer atividade que não gera valor ao cliente é um foco de desperdício e deve ser eliminado [45].

Segundo, [15], a maioria das organizações e equipes que adotaram *Agile* como o principal método para a entrega de software conhecem muito bem o Manifesto Ágil e os doze princípios ágeis. Entretanto, eles não devem ter tido um esforço consciente para medir seu nível de adesão aos princípios ágeis.

Para Hartmann [20], uma boa métrica ou diagnóstico ágil deve afirmar e reforçar os princípios Lean e ágil. Ela precisa focar no que é o valor para o cliente e nas características que fortaleçam os princípios ágeis. Em um nível organizacional, as métricas usadas para avalisar a legibilidade da organização na transição ágil, para ajudar a planejar as melhorias de processos ágeis e quantificar as decisões estratégicas sobre valores ágeis [19].

Um dos argumentos de motivação e justificativa para este trabalho, é que muitas empresas aderiram ou pensam em adotar as metodologias ágeis, mas não sabem como medir e monitorar se estes princípios são realmente transparentes. Outro ponto crucial é o fato que para adotar as práticas ágeis é necessário abraçar o manifesto ágil juntamente com os doze princípios ágeis, e as métricas mostram-se como auxiliares para coletar os dados, analisá-los e consequentemente conseguir efetivamente atingir o conjunto de objetivos que devem ser recém definidos.

Desta forma, o trabalho contribui em ajudar as empresas e estudiosos da área a conseguir medir e monitorar, através das métricas, os doze princípios ágeis propostos no manifesto. Serão coletadas métricas ágeis representativas e será realizado um mapeamento entre essas métricas e os princípios do desenvolvimento ágil de software.

1.2.1 Questão de pesquisa

Este trabalho de conclusão de curso busca responder a seguinte questão:

Dentre as métricas mais adotadas pela abordagem ágil, quais podem suportar a institucionalização dos doze princípios estabelecidos no Manifesto Ágil?

Tabela 1.1: Pergunta a ser respondida ao final do trabalho de conclusão do curso

Essas métricas citadas na pergunta serão selecionadas no decorrer do projeto, através inicialmente do método de revisão sistemática, que será abordado no capítulo 4. Outras métricas, que também farão parte deste conjunto, serão elegidas por estarem em destaque em livros, como o de Anderson [3] ou por serem consideradas fundamentais na literatura.

1.3 Objetivos

Direcionado pela questão de pesquisa estabelecida, o objetivo deste trabalho é investigar e selecionar um conjunto de métricas de desenvolvimento ágil de software, incluindo as métricas Lean, e mostrar que as mesmas dão suporte aos doze princípios ágeis. Através da análise, estudo e aprendizado com as métricas ágeis, demostrar que através da medição podemos tornar os doze princípios mais visíveis dentro da empresa.

O trabalho inicialmente irá reunir um conjunto de métricas representativas no meio ágil, para que consiga delimitar a quantidade de métricas que possam existir. Em seguida, será realizado um mapeamento entre métricas e princípios mostrando a conectividade entre eles.

1.4 Metodologia de Trabalho

Para que este projeto de conclusão de curso fosse organizado e trabalhado na melhor forma possível, ele foi dividido em seis fases principais, que serão descritas a seguir:

- **1ª Etapa - Pesquisa Exploratória** Nessa fase foi realizada uma pesquisa bibliográfica exploratória para definição do tema. Executando uma revisão relevante sobre

as metodologias ágeis e buscando referências para uma melhor contextualização e além disso confirmar a importância do mesmo e a possibilidade de contribuição para a área.

- **2ª Etapa - Definição da diretriz** Esta etapa foi de extrema importância para o trabalho. Foi nela, que foram apontadas e analisadas os objetivos e questões ou hipóteses da pesquisa, como também o escopo do mesmo.

- **3ª Etapa - Revisão da Literatura** Nesta fase foi realizada uma revisão em cima de métricas que apoiem os princípios do desenvolvimento ágil de software. Com ela, foram identificadas as métricas ágeis, e do Lean. Baseada na revisão sistemática proposta no artigo [28], que segundo a autora é um meio de identificar, avaliar e interpretar todas as pesquisas disponíveis relevantes para uma questão de pesquisa específica, ou área temática, ou fenômeno de interesse. Ainda nesta fase, foi realizada uma tabela indicando quais foram as métricas mais citadas pelos autores.

- **4ª Etapa - Análise dos dados coletados** Utilizando uma revisão da literatura baseada na revisão sistemática o resultado foi um conjunto de referências relevantes sobre o tema abordado. Com este aparato foi possível identificar quais são as métricas ágeis mais referenciadas, e depois foram analisadas quais foram as mais relevantes para o trabalho, afinal algumas destas métricas acabam tendo o mesmo significado ou sendo mostradas em um outro gráfico. Outro ponto, desta fase foi buscar métricas que mesmo não citadas ou tendo poucas menções, elas continham na literatura considerada primordial no universo Ágil. Em resumo, nesta fase foram selecionadas as métricas que foram utilizadas neste projeto de conclusão de curso.

- **5ª Etapa - Mapeamento** Aqui foi trabalhado a correspondência entre os princípios de desenvolvimento ágil e as métricas ágeis. Para cada princípio foi realizado um estudo para apontar as métricas, dentre as selecionadas, diretamente relacionadas com o princípio, e que assim possam prover suporte a sua institucionalização.

- **6ª Etapa - Conclusão** Nesta etapa contemplou o encerramento do projeto, nela foram respondidas as questões ou hipóteses do mesmo como também os trabalhos futuros que podem ser realizados seguindo esse mesmo direcionamento tratado no TCC.

1.5 Contribuições obtidas

A principal contribuição trazida por esta pesquisa é a realização de um mapeamento, analisando cada principio ágil e atribuindo ao mesmo métricas, que foram encontradas durante toda a trajetória deste projeto, que possam apoiar na sua adoção. E também ajudar trabalhos de organizações e estudiosos da área que desejem utilizar as métricas como suporte na adoção dos princípios. Outro ponto, como o trabalho precisou reunir métricas para realizar o mapeamento, consequentemente existiu uma fase apenas para agrupá-las, o que também pode auxiliar outras pesquisas. Servindo assim como base para outros trabalhos.

1.6 Organização do trabalho

Para que o projeto tenha uma estrutura desejável e consistente, o escopo está organizado em seis capítulos, são elas:

- **Capítulo 1:** Esta parte abrange a introdução que incluí a contextualização, a motivação, o objetivo, as contribuições que esta pesquisa fornecerá às organizações e para a academia, e, por fim, a organização do documento.
- **Capítulo 2:** Está exposto uma revisão bibliográfica que contempla uma introdução sobre o desenvolvimento ágil, o manifesto e a metodologia Scrum. Outros pontos abordados foram o Lean, LSD e o Kanban.
- **Capítulo 3:** Será apresentado um conjunto de métricas para a realização do mapeamento proposto.
- **Capítulo 4:** Neste ponto serão definidas e selecionadas as métricas para o mapeamento.
- **Capítulo 5:** Será apresentado o mapeamento entre os princípios e as métricas que dão suporte aos mesmos.

- **Capítulo 6:** Finalizando, encontra-se a conclusão do projeto. Tendo como pontos as principais contribuições trazidas pelo projeto, os trabalhos futuros e também as considerações finais.

Capítulo 2

Fundamentação Teórica

Neste capítulo, será apresentado uma revisão bibliográfica do tema abordado. O objetivo é mostrar a base do estudo e conceitos básicos do desenvolvimento ágil de software, inclusive a metodologia Scrum, Lean, inserindo também o LSD e Kanban. Outro ponto abordado será a importância de utilizar métricas no mundo ágil.

2.1 Desenvolvimento Ágil de Software

2.1.1 Manifesto Ágil

As ideias do desenvolvimento ágil ganharam força durante a década de 1990 e foram cristalizadas em 2001, através do manifesto ágil, realizado por um grupo experiente de profissionais da área de desenvolvimento de software [41].

E com esses ideais este grupo de dezessete pessoas destacaram as diferenças entre às abordagens tradicionais e esta nova abordagem. Eles consolidaram seu ideal de forma que mesmo que haja valor nos itens da direita o desenvolvimento ágil valoriza mais os itens da esquerda [14]:

- "**Indivíduos e interação entre eles mais que** processos e ferramentas"

As pessoas fazem a diferença em um time, e elas precisam ser valorizadas. Os processos e ferramentas são importantes, porém o saber trabalhar em equipe é um diferencial, a interação entre os envolvidos precisa ser prezado.

- "**Software em funcionamento mais que** documentação abrangente"

  A documentação é necessária e relevante, pois ajudam as pessoas a entenderem o que está sendo desenvolvido e caso haja novos envolvidos eles podem recorrer à ela. Porém, o sistema em funcionamento é a evidência do que o projeto anda bem, ou seja, no final o produto desenvolvido é o que realmente conta.

- "**Colaboração com o cliente mais que** negociação de contratos"

  No final do projeto a satisfação do cliente é um dos pontos cruciais de sucesso do mesmo. E é ele que pode dizer o que é esperado, apesar que normalmente ele não sabe explicar exatamente o que quer, podendo até mudar de ideia ao longo do tempo conforme vê o software funcionando [34]. Através do contato mais direto, ou seja, havendo a colaboração constante com cliente é mais fácil entender suas necessidades e conseguir ajustá-las a medida que o produto é desenvolvido. O contrato é importante afinal é necessário a definição de responsabilidades, direitos e deveres de ambas as partes, mas após o acordo o cliente ainda precisa continuar presente e fazendo parte do time.

- "**Responder a mudanças mais que** seguir um plano"

  Neste ponto temos que a mudança deve ser acolhida e os envolvidos precisam estar abertos à estas modificações. Muitas vezes, pode existir um apego ao plano inicial e isso pode prejudicar o potencial do projeto. A ocorrência de mudanças faz parte da realidade da área de negócios e elas podem acontecer por inúmeras razões: as regras e leis sofrem alterações, as pessoas mudam de ideia ou tecnologia evoluiu [34]. O projeto deve ter um plano, porém ele precisa ser flexível para acolher essas mudanças.

Seguindo esses valores, foram definidos doze princípios que têm como objetivo: a garantia da satisfação do cliente entregando rapidamente e continuamente softwares simples e funcionais,

com rápida adaptação às mudanças, através de indivíduos motivados e da colaboração de pessoas que entendam do negócio [14]. Os doze princípios do Manifesto Ágil [14] são:

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

2.1.2 Metodologias Ágeis

O termo "Metodologias Ágeis" tornou-se popular em 2001 quando dezessete especialistas em processos de desenvolvimento de software representando os métodos Scrum, Extreme Programming (XP) e outros, estabeleceram princípios comuns compartilhados por todos esses métodos. Em outras palavras, este termo se popularizou devido ao Manifesto Ágil.

As metodologias ágeis têm sido apontadas como uma alternativa às abordagens tradicionais para o desenvolvimento de software. Uma importante característica das metodologias ágeis é que elas são adaptativas e não preditivas. Enquanto abordagens preditivas encaram a mudança como um problema a ser evitado e um caro e indesejado desvio do planejado, métodos adaptativos acolhem a mudança como uma oportunidade de melhoria [23]. Outra visão diferenciada das metodologias ágeis é que a sua abordagens são orientadas a pessoas, não a processos [39].

De acordo com Fowler [17], a metodologia ágil não é uma anti-metodologia, na verdade, o que foi proposto foi a restauração da credibilidade da palavra. O que foi almejado foi restabelecer um equilíbrio: abraçando a modelagem, mas não apenas para apresentar alguns diagramas em um repositório corporativo empoeirado. Abraçar a documentação, mas não desperdiçar resmas de papel em tomos nunca mantidos e raramente utilizados.

Em resumo, o desenvolvimento ágil de software é incremental (pequenas entregas do software, com ciclos rápidos), cooperativo (clientes e desenvolvedores trabalhando constantemente juntos com uma comunicação eficiente e próxima), íntegro (o método sozinho é fácil para aprender e modificar), e adaptativo (capaz de realizar mudanças tardias) [39].

2.1.3 Scrum

Ken Schwaber e Jeff Sutherland, foram os idealizadores do Scrum. Eles a definiram como um processo iterativo e incremental, conforme a Figura 2.1

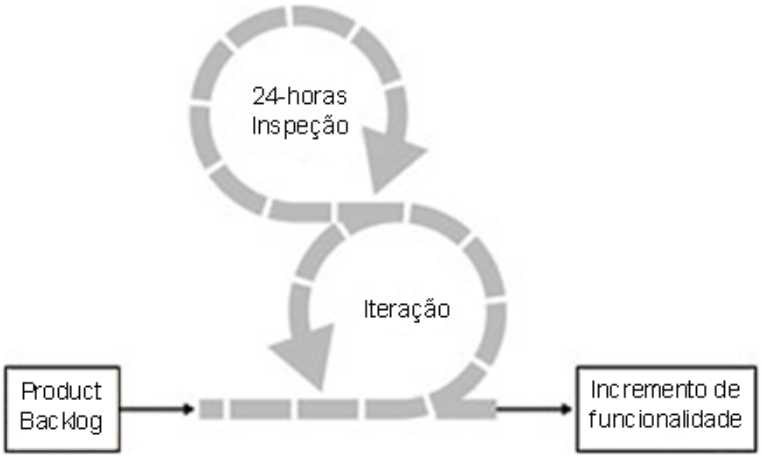

Figura 2.1: Representação do processo iterativo e incremental do Scrum [44]

O círculo mais abaixo representa uma iteração de desenvolvimento das atividades que ocorrem uma após a outra. A saída de cada iteração é um incremento do produto. O círculo mais acima representa a inspeção diária que ocorre durante a iteração, onde cada membro do time inspeciona o trabalho uns dos outros e faz adaptações que sejam necessárias [44].

Segundo Schwaber [44], todo projeto que utiliza Scrum começa com uma visão do produto que será desenvolvido, mais simples do que é o esperado no final. O *Product Owner*, então cria e prioriza o *Product Backlog* que é, definido basicamente como uma lista de requisitos, estórias, coisas que o cliente deseja, descritas utilizando a terminologia do cliente [29]. Após esta fase, estes itens chamados de *Backlog Items*, são divididos em *Releases*. Nas*Releases* é esperado que o que foi planejamento e priorizado e agrupado no *Product Backlog* sofram mudanças no momento que o projeto começa, o que pode refletir mudanças nas regras e requisitos de negócios.

As estórias do usuário ou *User Stories*, são cartões que representam os requerimentos dos clientes. Como ela pode ser lida por qualquer pessoa do time é importante que seja legível para todos os envolvidos [6]. Cohn [6], descreve que para criar boas estórias o foco precisa ser em seis atributos, são eles:

- Independentes;

- Negociáveis;
- Estimáveis;
- Pequenas;
- Testáveis;
- Valiosas para os usuários e/ou clientes;

Todo o trabalho realizado pelo o time, denominado *Scrum Team*, é feito em ciclos ou iterações (que duram. no máximo, 30 dias), chamadas *Sprints*. No Scrum *Scrum Team* precisa desenvolver um incremento do produto, que precisa potencialmente entregável para o cliente, conhecido como *Product Owner*. Cada uma das *Sprints* começam com uma reunião denominada *Sprint Planning Meeting*, onde o PO e o time entram em consenso sobre o que será trabalhado na iteração. Nesta etapa, o PO apresenta os itens de maior prioridade, e o time, de acordo com sua capacidade de produção, seleciona os itens que serão entregues no final da iteração [44].

Todos os dias, o *Scrum Team* realiza uma reunião com uma duração máxima de 15 minutos e, mais comumente, feitas em pé, chamada *Daily Scrum*. Nestas reuniões, os envolvidos respondem três perguntas essenciais: O que você tem feito neste projeto desde a última *Daily Scrum Meeting*? O que você planeja fazer neste projeto de agora até a próxima *Daily Scrum Meeting*? Quais impedimentos estão no caminho? [44]. Ela serve, justamente, para sincronizar o trabalho de toda a equipe.

No final cada Sprint, é realizada a *Sprint Review Meeting*, onde todos os envolvidos no projeto se reúnem com a finalidade de apresentar ao PO o produto desenvolvido ao logo da iteração. Tendo cumprido esta fase, o time realiza uma retrospectiva, chamada de *Sprint Retrospective*, de tudo o que ocorreu na iteração que passou, com a finalidade de melhorar o entrosamento da equipe e a qualidade do produto da próxima *Sprint* [17]. Todo o ciclo do Scrum pode ser visualizado na Figura 2.2.

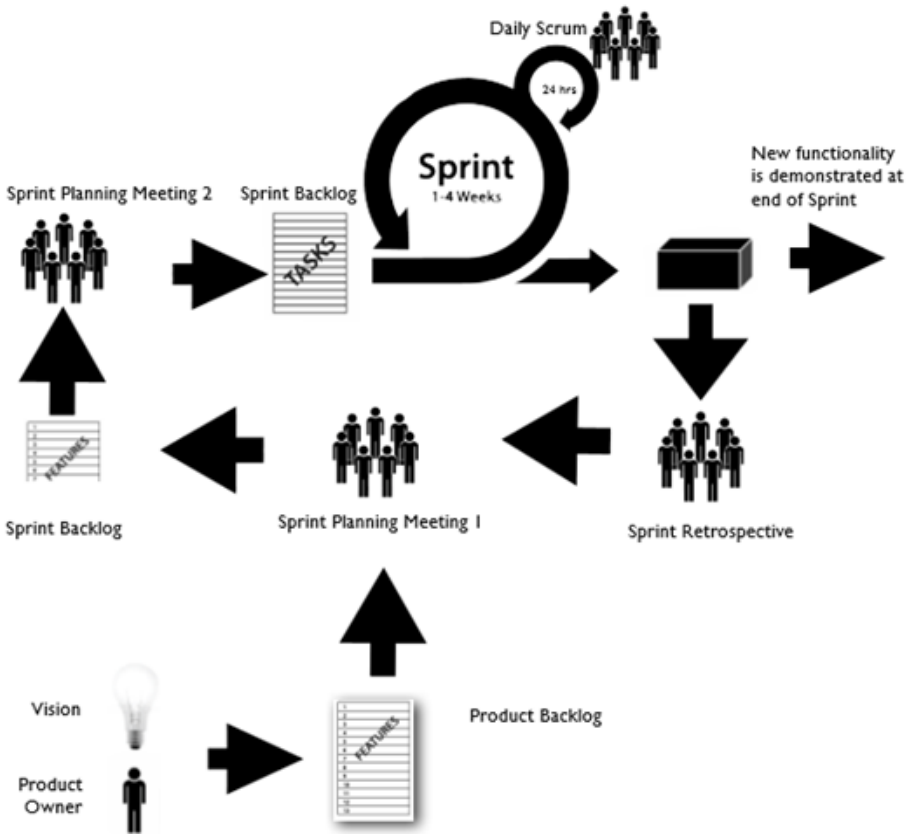

Figura 2.2: Representação do ciclo Scrum [47]

2.2 Lean

Com o fim da Segunda Guerra Mundial, a indústria japonesa sofreu bastante com a escassez de recursos e a baixa produtividade. E para competir com os fabricantes de carros americanos, que na época adotavam o modelo de produção em massa, a Toyota precisava mudar a sua estratégia e adaptar-se aos problemas que enfrentava na década de 90. A resposta a esta concorrência foi a criação do Sistema Toyota de Produção, que teve como mentor Taiichi Ohno e teve como pensamento o princípio fundamental do Lean: eliminar desperdícios [40].

O Sistema Toyota de Produção, foi o precursor do termo *Lean Manufacturing*, que foi criado por Taiichi Ohno e Shigeo Shingo [10]. Tinha como pilares os seguintes princípio:

- ***Just-in-time:*** Uma peça só deveria chegar na linha de montagem somente no momento da sua utilização, ou seja, os bens e serviços só devem ser produzidos no momento exato que são necessários.
- **Redução de desperdícios:** Eliminação de todas as atividades que não têm valor agregado. E dentre os tipos de desperdícios o Ohno [40] identificou os sete principais:
  1. Superprodução;
  2. Espera;
  3. Transporte desnecessário (dos produtos e matérias primas);
  4. Processamento extra ou incorreto;
  5. Estoque (inventário);
  6. Movimento desnecessário;
  7. Defeitos.
- **Sistema Puxado:** Os produtos são retirados pelo cliente final e não empurrados para o fim da cadeia de produção, ou seja, quem "puxa"o produto é o cliente.

Em 1990, o livro *"The Machine That Changed The World"*, popularizou o termo Lean e ilustrava claramente a diferença de desempenho significativo entre as indústrias automotivas japonesas e ocidentais. Eles descreveram os principais componentes responsáveis por este desempenho superior, pontuando-os em cinco princípios da filosofia Lean [24], que são:

- O foco do que gera valor deve ser no cliente.
- Identifique todas as etapas ao longo de toda a cadeia de valor, removendo as fases que geram desperdícios.
- Fazer ações que geram fluxo de valor.
- Apenas faça o que é puxado pelo cliente, no tempo que necessário (*just-in-time*).
- A busca pela a perfeição deve ser algo constante, removendo continuamente camadas sucessivas de resíduos (Melhoria Contínua).

2.2.1 LSD

A filosofia Lean inicialmente aplicada em um sistema de produção se disseminou em várias áreas, inclusive a de desenvolvimento de software. Mas a aplicação das práticas e princípios criados especialmente para um sistema de produção de automóveis não é trivialmente aplicado na indústria de desenvolvimento de software. Software é um produto com maior grau de customização e seu desenvolvimento não deve ser comparado a produção de um automóvel.

Mary e Tom Poppendieck [40] traçaram pontos em paralelo entre os valores e práticas Lean com o desenvolvimento de software, gerando sete princípios, são eles:

- **Elimine Desperdícios:** tudo aquilo que não agrega valor para cliente final e que não são percebidos pelo cliente é considerado desperdício.
- **Amplifique o aprendizado:** O desenvolvimento de software é um processo continuo de aprendizado, pois eventualmente existe uma mudança seja no processo ou novos envolvidos ou até mesmo a adição de uma nova funcionalidade. E para acompanhar e assimilar todo esse futuro aprendizado é necessário um ambiente de desenvolvimento propício ao compartilhamento e expansão do conhecimento.
- **Adie Decisões:** Como dito anteriormente o ambiente muda constantemente e é preciso evitar tomar decisões precipitadas ou até mesmo apressadas que possam causar posteriormente desperdícios e até mesmo fracassos. Retardar decisões dá a oportunidade de aprender mais e consequentemente selecionar as melhores ações e assim diminuir as incertezas dentro do projeto.
- **Entregue o quanto antes:** Quando pequenas entregas são feitas em iterações podemos adquirir mais frequentemente os feedbacks e assim oferecer ao cliente produtos que agreguem mais valor para o mesmo. Pois, com entregas rápidas é possível aprender com erros.
- **Dê poder as Pessoas:** As pessoas precisam de mais do que uma lista de tarefas. Se seu trabalho é fornecer a motivação intrínseca, eles precisam entender e se comprometer com o propósito do trabalho. A motivação intrínseca é especialmente poderoso se as

pessoas em uma equipe cometer em conjunto para a realização de uma finalidade que preocupam.

- **Construa com integridade:** O conceito de integridade significa que os componentes separados de um sistema trabalham corretamente juntos com equilíbrio entre a flexibilidade, facilidade de manutenção, eficiência e capacidade de resposta. Construir com integredidade é atribuir boas práticas para garantir que isso funcione, por exemplo, refatoração, pois esse conceito é sobre como manter a simplicidade, a clareza, a quantidade mínima de recursos no código.

- **Otimize o Todo:** O entendimento das necessidades do cliente é um fator de extrema importância durante o desenvolvimento de software e que deve ser gerado não apenas na fase de desenvolvimento, ou seja, otimize desde o começo até o final do ciclo. Uma das formas indicadas pelo Lean de visualizar possíveis problemas é usando métricas que serão mais detalhadas na próxima subseção.

2.3 Kanban

De acordo com Boeg [5], a ferramenta que melhor representa o Lean é o Kanban. O método Kanban introduz um sistema complexo adaptável cujo o ideal é estimular um resultado Lean dentro de uma organização [2].

A palavra "Kanban"é uma palavra japonesa e significa "Cartão Visual". O Kanban é uma ferramenta de controle visual aplicada para mostrar e consequentemente otimizar o fluxo de trabalho no desenvolvimento de software [5]. Anderson [3], apresenta que o Kanban não é uma metodologia de ciclo de vida para o desenvolvimento de software ou uma abordagem de gerenciamento de projetos, afinal ele requer que haja alum processo em vigor de forma que o mesmo possa ser aplicado para alterar incrementalmente o processo base.

O Kanban, possui cinco propriedades fundamentais para criar um conjunto resultante de comportamentos Lean [3]. São eles:

- Visualizar o Fluxo de Trabalho;
- Limitar o Trabalho em Progresso (WIP);
- Medir e Gerenciar o Fluxo;
- Tornar as Políticas do processo explícitas
- Utilizar modelos para reconhecer as oportunidades de melhoria.

No Kanban, é necessário definir o processo atual, visualizar o fluxo de trabalho e mapear a cadeia de valor. O segundo passo é estabelecer as colunas que devem ter um WIP (por exemplo, o número máximo de cartões em cada fase), que deverá limitar as atividades em andamento para cada estágio do processo. E um novo trabalho é puxado para dentro de um processo assim que um cartão de sinalização retorna o momento em que uma atividade ou pedido for finalizado [3].

Um exemplo de quadro em Kanban pode ser visto na Figura 2.3. Ela mostra ideias dos conceitos fundamentais citados nesta seção, como, por exemplo, os limites de WIP (escritos na parte de cima de cada coluna). O motivo pelo qual nos referimos a esse sistema como um "Sistema Puxado Kanban"(Kanban Pull System) é que, ao visualizar o fluxo e estabelecer os limites de WIP, garantimos que nunca se pode introduzir mais trabalho no sistema do que a sua capacidade. Está disponível apenas uma quantidade limitada de permissões de trabalho ("kanbans"); então é necessário que se complete o trabalho existente antes que um novo trabalho possa ser iniciado. Isto resulta em funcionalidades sendo puxadas pelo sistema, com base na capacidade deste, em vez de empurradas com base em previsões ou demandas [5].

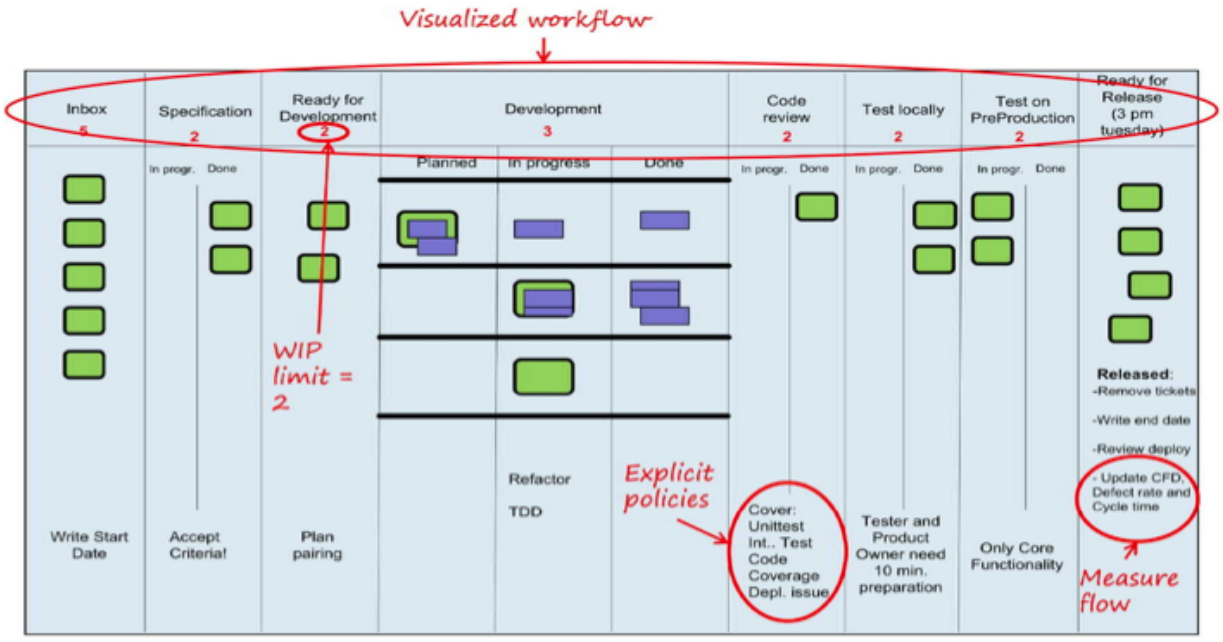

Figura 2.3: Exemplo quadro Kanban[5]

2.4 Métricas

Uma métrica é um método para determinar se um sistema, componente ou processo possui um certo atributo. Ela é geralmente calculada ou composta por duas ou mais medidas [21]. E as métricas de software são meios úteis para ajudar os engenheiros de software desenvolver sistemas de software grandes e complexos [35] e é mais fácil controlar a qualidade do software e desenvolvimento esforço se podemos quantificar o software [33].

De acordo com Fenton e Neil [16], qualquer métrica de software é uma tentativa de medir ou prever algum atributo de algum produto, processo ou recurso - no caso de métricas de produtos é, obviamente, um produto de software. E elas não são uma parte do paradigma ágil e incorporá-los em um processo ágil não é uma tarefa trivial. Desde que, a ideia do desenvolvimento ágil de software está se libertando do processo de práticas onerosas e concentrando-se diretamente na produção de software de trabalho, que é a principal medida de progresso [14]

A maior prioridade no desenvolvimento ágil de software é satisfazer o cliente através da entrega de valor [14]. E, sem medição não é possível controlar diretamente a entrega desses benefícios para o cliente e outras partes interessadas (por exemplo, os usuários). [32].

Os Métodos Ágeis estimulam um processo empírico para o desenvolvimento de software. Esse tipo de abordagem exige um ciclo constante de inspeção, adaptação e melhoria. Encontrar maneiras eficazes de avaliar o processo e a equipe de desenvolvimento não é uma tarefa simples [43]. A importância das métricas se ressaltam a partir do momento que elas possibilitam o acompanhamento mais preciso das atividades do projeto.

Na área de desenvolvimento de software, um projeto é uma atividade que envolve trabalho intenso para que os critérios de qualidade estabelecidos atendam aos requisitos dos clientes ou usuários. Sendo assim, neste meio as métricas se apresentam como informações fundamentais para mensurar o que está sendo desenvolvido de forma precisa. Métricas de software podem auxiliar de forma efetiva na administração de um projeto ou simplesmente controlar o nível da qualidade de determinada tarefa ou processo, garantindo desde as primeiras tarefas do desenvolvimento de software o comprometimento do time envolvido com a qualidade, sendo elas também serem uns dos agentes propulsores para a melhoria continua [41]

Diante disto, Pressman [41] afirma que para um projeto de desenvolvimento de software seja bem sucedido, é necessário que alguns parâmetros sejam previamente analisados como, por exemplo, o seu escopo, os riscos aos quais o projeto está envolvido, os recursos que serão necessários para que os objetivos estabelecidos sejam atendidos, as tarefas que deverão ser executadas e os indicadores a serem acompanhados, ou seja, métricas devem ser estabelecidas e monitoradas.

Capítulo 3

Revisão da Literatura

Neste capítulo será exposto o método de pesquisa aplicado por este trabalho. Nele será apresentado todas as etapas realizadas, e também o detalhamento de cada uma delas.

3.1 Método de pesquisa

De acordo com Barbara Kitchenham [28], uma revisão sistemática da literatura é uma forma de identificar, avaliar e interpretar todas as pesquisas disponíveis relevantes para uma questão de pesquisa específica, ou área temática, ou fenômeno de interesse. Analisando esta definição e o projeto em questão, concluiu-se que a metologia aplicada se baseia nesta abordagem.

Para conduzir este modelo de pesquisa é necessário seguir alguns estágios, são eles:

- Definição da(s) questão(ões) da pesquisa.
- Especificação das bases de extração de dados que serão utilizadas.
- Análise e seleção das palavras-chave.
- Triagem dos estudos encontrados, nesta etapa são utilizados os critérios de seleção e exclusão dos conteúdos obtidos.

- E por fim uma análise qualitativa, buscando assim investigar a qualidade dos trabalhos encontrados, visando selecionar aqueles mais relevantes para a área em estudo.

As subseções a seguir detalham cada um dos estágios abordados anteriormente.

3.1.1 Definição da pergunta

O primeiro passo foi definir a pergunta da pesquisa, que é o direcionamento desta revisão. Ao final desta revisão será obtida uma resposta para a questão apresentada.

No desenvolvimento ágil de software, quais são as métricas mais aplicadas?

Tabela 3.1: Pergunta a ser respondida no final da pesquisa.

3.1.2 Bases de dados relevantes

Esta etapa compreende a triagem das bases de dados. Elas foram selecionadas por serem mais confiáveis e inseridas no tema proposto. (Tabela 3.2).

Nome
ACM
Portal CAPES
IEEE
Google Scholar
Springer

Tabela 3.2: Fontes de dados utilizadas na revisão.

3.1.3 Palavras-chave

Neste estágio, foram analisadas e definidas as palavras-chave (Tabela 3.3) que foram utilizadas durante a pesquisa. É importante ressaltar que as mesmas foram analisadas a partir da quebra da questão de pesquisa.

Palavras-chave
Agile Metrics
Agile Measurement
Lean Metrics

Tabela 3.3: Palavras-chave aplicadas na pesquisa.

3.1.4 Critérios de Exclusão e Inclusão

Após a escolha das palavras-chave é necessário definir quais serão os critérios de exclusão e seleção dos estudos. Para consequentemente realizar as pesquisas seguindo esses pontos. Os critérios de seleção e/ou exclusão foram:

- **Língua:** Devido à algumas bases de dados poderem estar em várias línguas um dos preceitos para a busca foi apenas estudos nos idiomas inglês e português.
- **Palavras-chave em lugares estratégicos:** Outro critério escolhido foi filtrar a pesquisar de maneira que as palavras-chave estejam no título ou no resumo.
- **Base atualizada:** Outro ponto de seleção foi realizar uma busca por estudos atualizados, assim foi escolhido como critério apenas aqueles realizados a partir de 2008 até 2014.

3.1.5 Extração dos dados

Após definir os critérios de exclusão e inclusão dos dados, foi realizada a pesquisa de acordo com o que foi estabelecido nos estágios da revisão sistemática. Segue abaixo a tabela (Tabela

3.4) com todas as fases da seleção de estudos, as bases de dados utilizadas e também a quantidade encontrada de materiais.

Fases da seleção de estudos	**Base de dados**	**Quantidade**	
		Agile Metrics	**Lean Metrics**
Apenas palavras-chave	ACM	1917	1299
	Google Scholar	24800	62400
	IEEE	680	21
	Springer	6989	7952
	Portal CAPES	213	4285
Língua	ACM	1917	1299
	Google Scholar	24800	62400
	IEEE	680	21
	Springer	6782	7824
	Portal CAPES	192	3955
Palavras-chave no título e/ou resumo	ACM	30	2
	Google Scholar	28	16
	IEEE	31	1
	Springer	2	1
	Portal CAPES	21	4
Artigos entre 2008 e 2014	ACM	12	2
	Google Scholar	18	3
	IEEE	14	1
	Springer	1	1
	Portal CAPES	11	3

Tabela 3.4: Quantidade de estudos encontrados em cada fase da seleção.

No final da quarta fase dos critérios foram totalizados sessenta e seis materiais. Sendo que dentro desse conjunto foram encontrados trabalhos que muitas vezes faziam referências a outras, o que gerava repetição de arquivos. Eliminando estas reincidências foram totalizados

54. Dentre eles seis foram excluídos por não se tratarem de desenvolvimento de software, ou seja, 48 materiais.

3.1.6 Análise Qualitativa

Como mencionado um dos passos da revisão sistemática é investigar a qualidade e validade dos estudos encontrados. A avaliação dos documentos, obtidos após os critérios de exclusão e de seleção, foi realizada através de uma leitura do resumo, da introdução e também das palavras-chave dos mesmos. Realizando esta leitura foram totalizados 15 materiais.

Além disso, durante o processo de revisão sistemática, foram encontrados dois artigos ([13] e [20]) ambos antes de 2008, e mesmo sendo excluídos no critério de exclusão foram adicionados à pesquisa, pelo fato de estarem alinhados à mesma.

3.1.7 Conjunto de métricas ágeis selecionadas

Após a escolha dos trabalhos através da revisão sistemática serão apresentadas as métricas ágeis segundo uma perspectiva multidimensional. Esta visão será detalhada abaixo, através de uma tabela (Tabela3.5) que apresenta os nomes das métricas e os artigos que mencionam as mesmas.

Nome da Métrica	**Artigos**
Velocidade	[11], [12], [18], [9], [19], [20], [38] , [31]
Burndown	[11], [12], [18], [49], [19], [13], [38], [31]
Burn-up	[11], [12], [18], [38]
Cumulative Flow	[11], [12], [18]
Estimativa de Esforço	[11], [12]
Tamanho do Software	[11], [12], [13]
Cálculo de re-trabalho (*Re-work measurement*)	[11]
EBV	[11], [12], [49], [1], [50], [19]

AgileEVM	[12]
Spikes	[12]
CoV	[12]
Technical Debt	[18], [31]
ACPF	[18]
SDI	[49]
PSU	[49],[19]
Story Points	[12], [1], [19], [20], [32], [48]
Enterprise Balanced Score Card	[19]
NPV	[20]
IRR	[20]
ROI	[20], [38]
Esforço para Integração e Testes	[12]
Pontos de função	[1], [9], [32]
LOC	[1], [9], [32], [38]
Velocidade vs Linhas de Código	[9]
Lead time	[11], [18], [10]
Capacidade	[10]
OEE	[10], [4]
Cycle Time	[10], [4], [38]
Throughput (Vazão)	[11], [18], [49], [10]
Takt Time	[10]
Dock-to-Dock Time	[10], [4]
BTS Ratio	[10], [4]
FTT capability	[10], [4]
Queues time	[18]
WIP	[11], [18], [10], [4]
Number of Test Suites	[32]
KLOC	[32]

Functional Tests per User Story	[20], [38]
Acceptance Test per User Story	[38]
Unit tests per User Story	[20], [13], [38]
Bug Distribution	[18], [1], [32]
Test Points	[1], [32], [13]
Defect Density	[1], [9], [32]
Cumulative Number of Defects	[1], [32], [38], [48]
Running Automated Tests	[18], [49],[32]
Running Tested Features	[49], [9], [50], [19], [32], [13], [38]
Burn-Up de defeitos	[12], [18]
Burndown de defeitos	[12], [18]
Kanban guard	[32]
Pulse	[13]
Faults	[13]
Number of Check-in	[13]

Tabela 3.5: Conjunto de métricas encontradas com os seus respectivos artigos.

Após todas as fases da revisão sistemática é possível verificar que a quantidade de métricas ágeis, incluindo métricas Lean, é bem vasta e pelo volume de trabalhos atuais é perceptível que este número tende a crescer ainda mais devido a importância da medição durante um projeto ou ação. E com as fases aplicadas na pesquisa foi possível realizar um conjunto de métricas e seus respectivos artigos, mostrado na Tabela 3.5.

Com o uso dessa metodologia foi alcançado o objetivo deste capítulo, que corresponde responder a questão proposta, ou seja, mostrar quais são as métricas ágeis mais citadas, conforme pode ser observado no gráfico (Figura3.1).

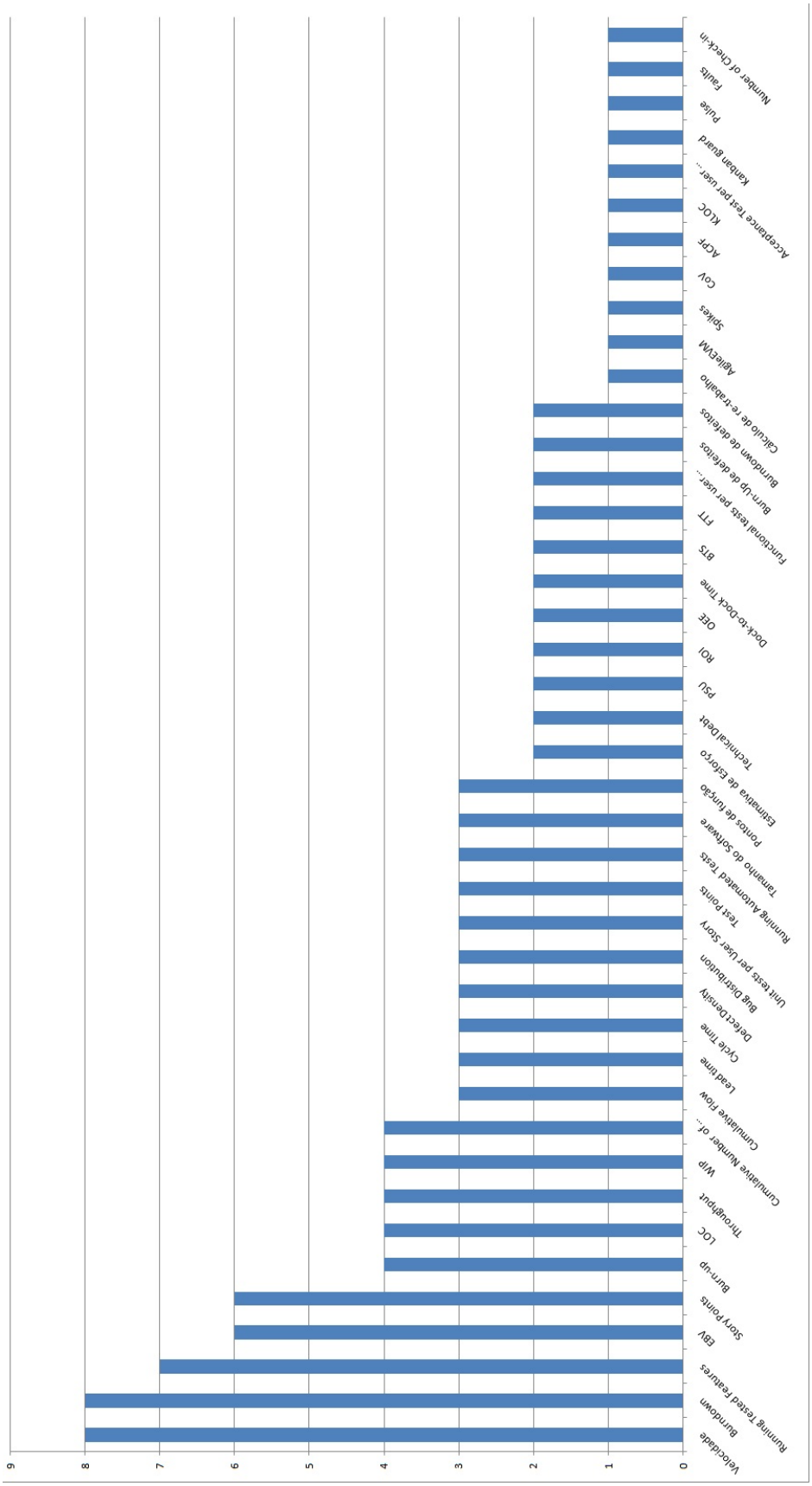

Figura 3.1: Gráfico: Métricas vs quantidade de artigos

No final desta seção foram selecionadas ao todo quinze métricas dentre as mais citadas pelos autores e que serão utilizadas durante a proposta deste projeto de conclusão de curso. Elas serão definidas e selecionadas no próximo capítulo.

Capítulo 4

Conjunto de Métricas Ágeis

Neste capítulo é apresentado o conjunto de métricas que será utilizado para obter o resultado esperado pelo projeto. Ou seja, elas foram consideradas as mais relevantes para apoiar na adoção dos princípios ágeis.

4.1 Definição das Métricas

Nesta seção, serão apresentadas as quinze métricas mais citadas pelos autores na revisão sistemática. As quais serão posteriormente analisadas para que façam parte, ou não, do conjunto de métricas viáveis. As métricas selecionadas na revisão sistemática estão detalhadas a seguir.

1. **Velocidade:** Este termo é mais utilizado em comunidades ágeis ao invés de "produtividade". [7]. Dito de forma simples, a velocidade é o volume de trabalho realizado em um determinado período de tempo, por uma determinada equipe. É, comumente, definida como o número de histórias de usuários concluídas por iteração e é frequentemente usada para estimar tempo que ela permaneceu até o final do projeto. [11]. Ela pode ser usada também com um comparativo com a velocidade esperada e velocidade atual. Como pode ser observado na Figura 4.1.

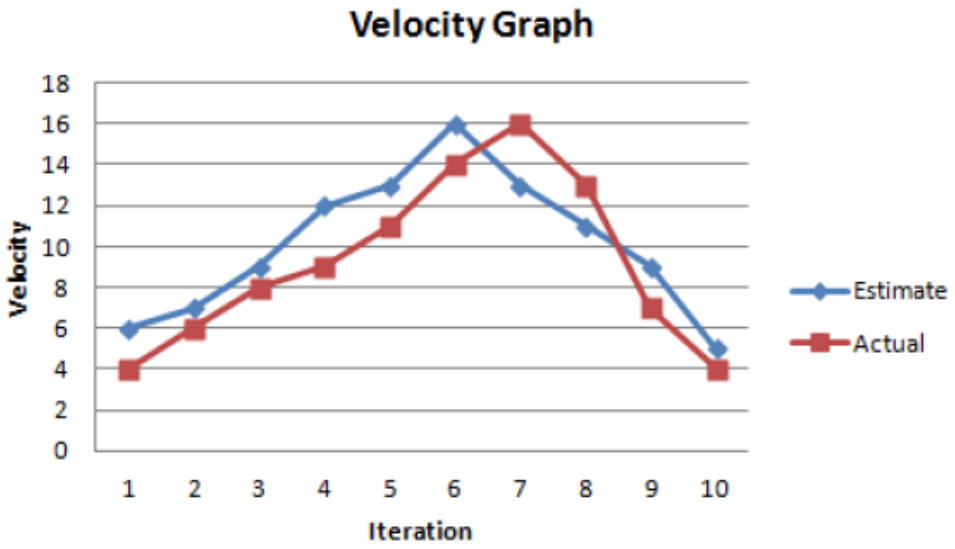

Figura 4.1: Gráfico: Velocidade atual e esperada. [11]

Um exemplo de gráfico de velocidade mais simples pode ser visto na Figura 4.2.

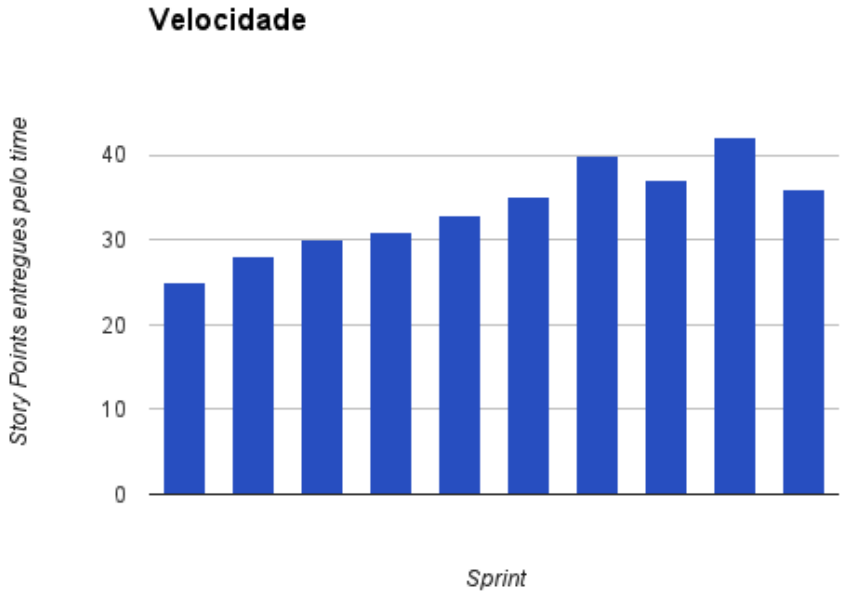

Figura 4.2: Gráfico de coluna: Velocidade Simples

2. ***Burndown:*** É uma importante ferramenta de medição para o planejamento e monitoramento do progresso nos métodos ágeis com base em software funcionando.[44].

Esse gráfico representa a quantidade de trabalho que falta ser feito no eixo vertical (y) versus o tempo no eixo horizontal (x). Ele é comumente usado de duas formas: *iteration burndown* e *release burndown*. [11]. Sendo a estimativa do trabalho restante das tarefas a serem concluídas na iteração e para versão atual, respectivamente. Ou seja, um gráfico simples que mostra a quantidade de trabalho na linha vertical e o tempo sobre o eixo horizontal. O acompanhamento é feito justamente no trabalho ainda não está feito. O objetivo é atingir o "chão". A inclinação da curva pode nos ajudar a aproximar quando isso vai acontecer ou, em outras palavras, quanto estamos próximos a atingir o trabalho a ser feito. A Figura 4.3 mostra uma representação deste gráfico.

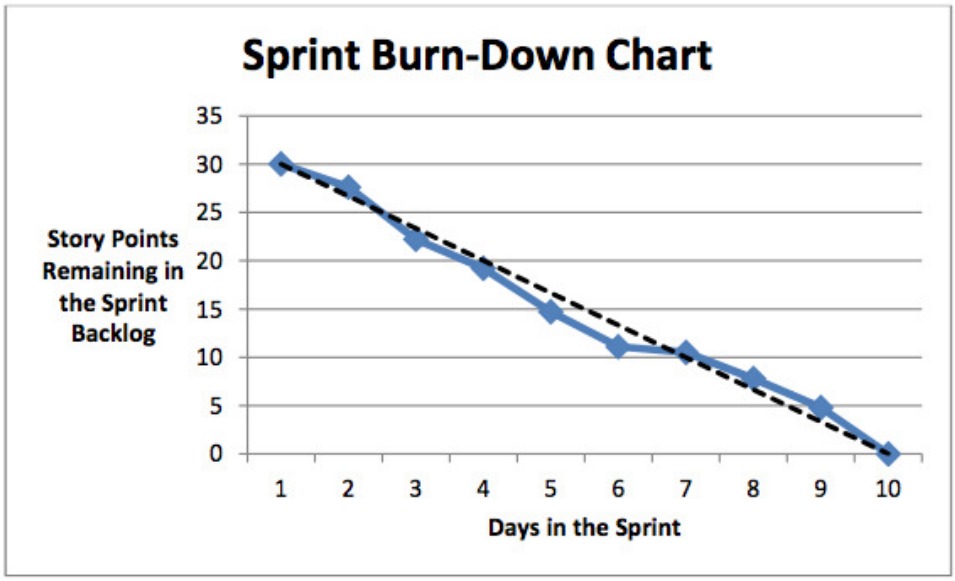

Figura 4.3: Gráfico: Burn-down [12]

3. ***Running Tested Features:*** Segundo Jeffries, [25], o grande ponto da maioria dos projetos de desenvolvimento de software é um sistema que funciona, e que tem a maioria dos recursos possíveis por cada dólar de investimento. Ele descreve esta a noção como *Running Tested Features*. Ainda segundo este autor a seguinte definição dessa métrica seria:

   - O software desejado é dividido em recursos nomeados (requisitos, estórias) que fazem parte do que significa entregar o sistema desejado.

- Para cada recurso nomeado, há um ou mais testes de aceitação automatizados que, quando trabalham, mostram que o recurso em questão foi implementado.
- Os RTF mostram métricas, em todos os momentos do projeto, quantos recursos estão passando em todos os seus testes de aceitação.

Este gráfico deverá aumentar, essencialmente, de forma linear a partir de um dia até ao final do projeto. Para isso, a equipe terá que aprender para se tornar ágil. Desde o primeiro dia, até que o projeto estiver concluído, este crescimento deve ser suave e consistente no número de "Runninng Tested Features". "Running"significa que os recursos são enviados em um único produto integrado. Testado significa que os recursos estão passando continuamente os testes estabelecidos pelos requisitos. [25]

Um exemplo deste gráfico pode ser visto na Figura 4.4.

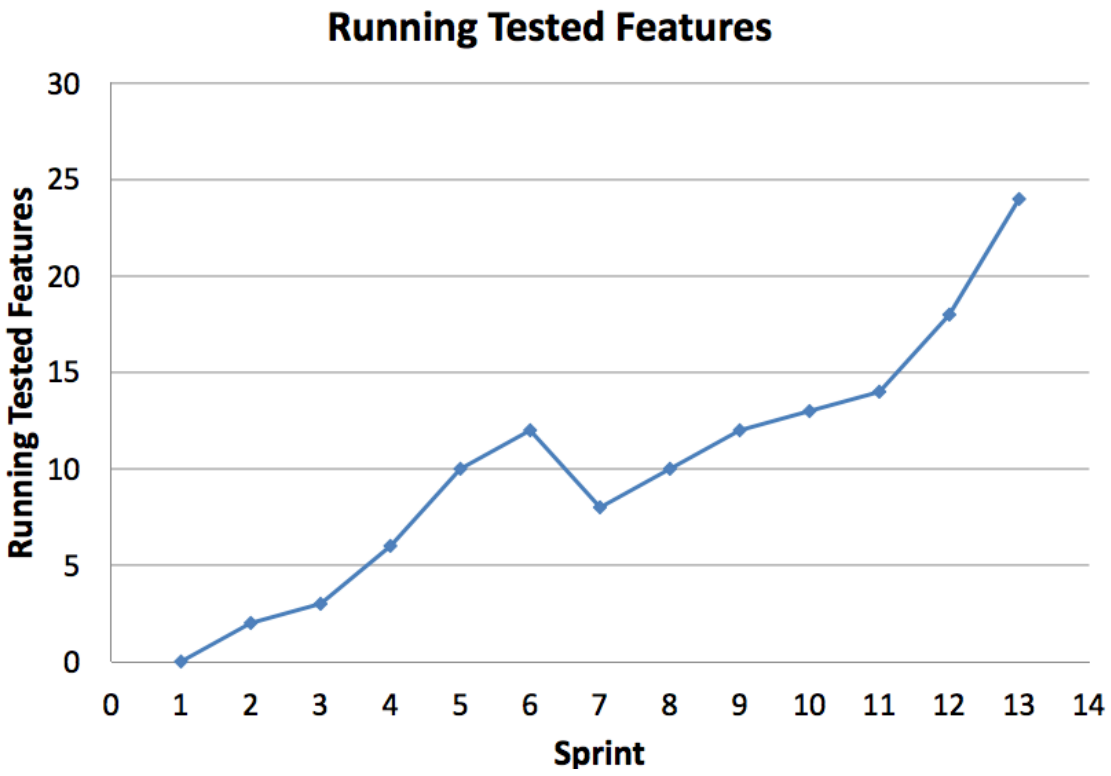

Figura 4.4: Gráfico: Running Tested Features

4. **EBV:** Segundo Rawsthorne [42], para descobrir se estamos recebendo o valor de negócio que é esperado ou necessário, temos que usar valores de negócios das histórias

concluídas, e não tamanhos. O EBV foi definido como "a percentagem do valor do negócio, conhecido, do produto, que atualmente está ganho". Em outras palavras, temos as definições na tabela4.1.

Nome	Definição
BVI(Story)	O Índice de Valor de Negócio de uma história - a percentagem do valor do negócio total que a história é o \"dono" $= \frac{BV(Story)}{\sum BV(Story)}$ onde a soma é sobre todas as estórias the um produto ou / *release*, conforme for apropriado.
EBVn	O valor absoluto de todo o valor dos índices de negócios para as estórias concluídas = Σ BVI (estória) para todas as estórias concluídas nas primeiras n *sprints* (seria um total em execução).

Tabela 4.1: Definições EBV e BVI

Um modelo desta métrica pode ser visto na Figura 4.5.

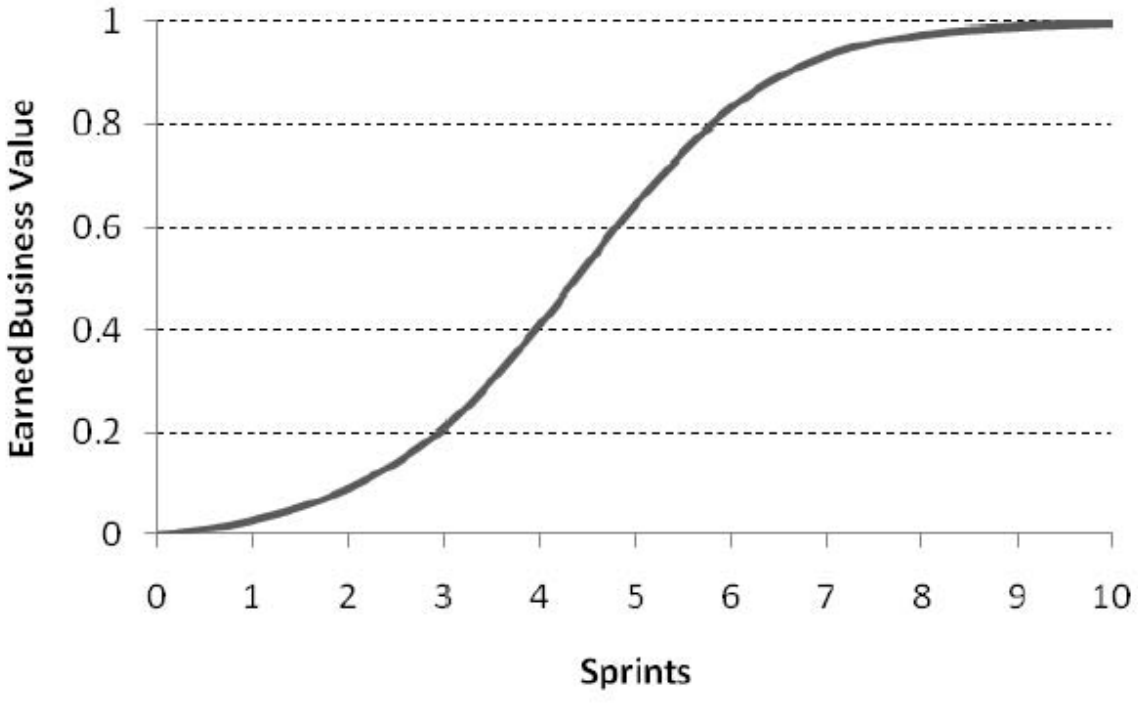

Figura 4.5: Gráfico: EBV

5. ***Story Points:*** Serve para medir o esforço necessário para implementar uma estória. Esta medição é feita através de um pontuação arbitrária, que define o quão é difícil ou fácil de desenvolver uma determinada estória. Uma série comumente utilizada para pontuar um estória é a série de Fibonacci, que representa a estimativa do tamanho daquela parte do software como também a complexidade do mesmo.

6. ***Burn-up:*** Esse gráfico, segue uma mecânica bem parecida com o gráfico de Burn-down, em que o escopo (esforço sendo até mesmo *Story Points*) é representado na linha vertical e o tempo no eixo horizontal, em outras palavras, este gráfico mostra a taxa com que a equipe completa os requisitos ao longo do tempo e se esta taxa está de de acordo com o planejado [12]. A grande diferença entre o gráfico de Burn-up e o Burndown é que, em vez de acompanhar a quantidade de trabalho que resta a ser feito, o que é levado em consideração é a quantidade de trabalho que já concluída, de modo que a curva está subindo, não para baixo como mostra a Figura 4.6.

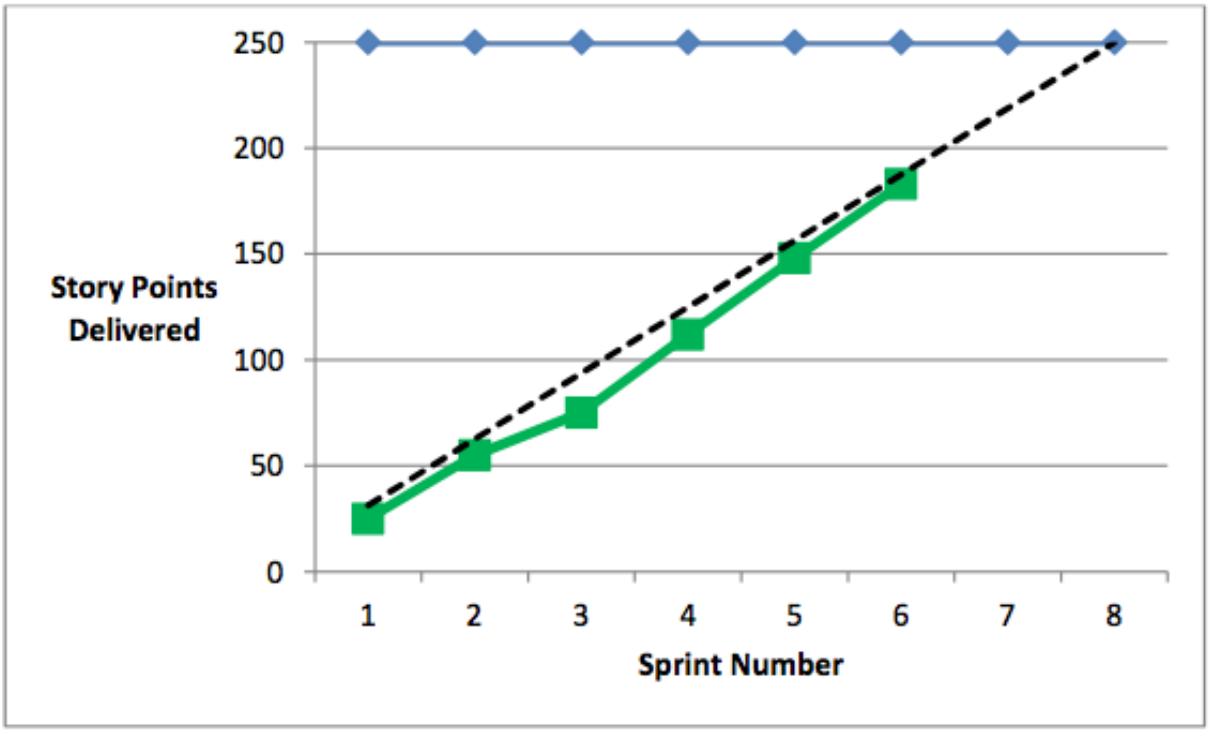

Figura 4.6: Gráfico: Burn-up [12]

7. **LOC:** O número total de linhas de código de produção de um sistema é um exemplo de medida de software consideravelmente famoso, porém segundo [43], não existe um padrão universal para representar linhas de código, pois as linguagens podem variar, assim como as regras para cálculo de linhas de código.

8. ***Throughput:*** Pode ser descrita como a velocidade com que um sistema gera seus produtos ou serviços por unidade de tempo. O tempo começa a ser contado partir do momento em que a organização começa a trabalhar em um pedido até que o produto ou serviço seja entregue ao cliente.

9. **WIP:** Nesta medida temos a quantidade de trabalho em execução. Quanto menor é este trabalho-em-progresso melhor.

10. ***Cumulative Number of Defects:*** Examinando o fluxo de trabalho para descobrir e corrigir defeitos, as métricas que revelam o tempo de ciclo e o acúmulo de defeitos em vários estados de trabalho formam um indicador de alerta muito forte [12]. A Figura 4.7, retirada do artigo [12] ilustra uma análise particular para carga de trabalho relacionada com defeitos. Os sete estados listados à direita são representadas em faixas coloridas no diagrama.

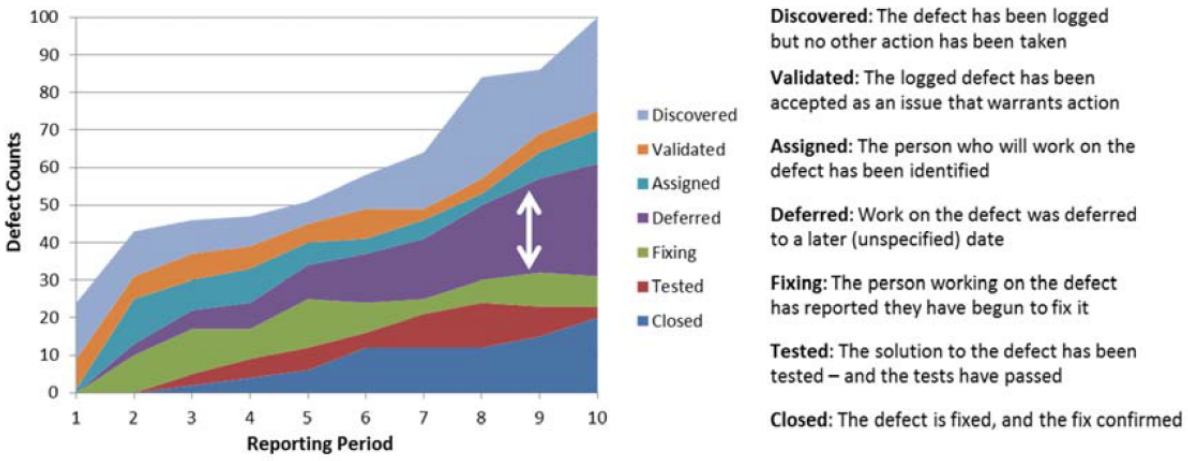

Figura 4.7: Análise de defeitos usando um Cumulative Flow [12]

11. ***Cumulative Flow:*** Este diagrama foi introduzido por Anderson em 2003, como o melhor substituto para o Burndown.[2]. Ele apresenta uma quantidade de trabalho em um determinado estado. A Figura 4.8 mostra um exemplo desse diagrama. Nesta figura quantidade de trabalho em cada iteração, é ilustrada em um ou mais estados predefinidos (concepção, desenvolvimento, teste e implantado) [11].

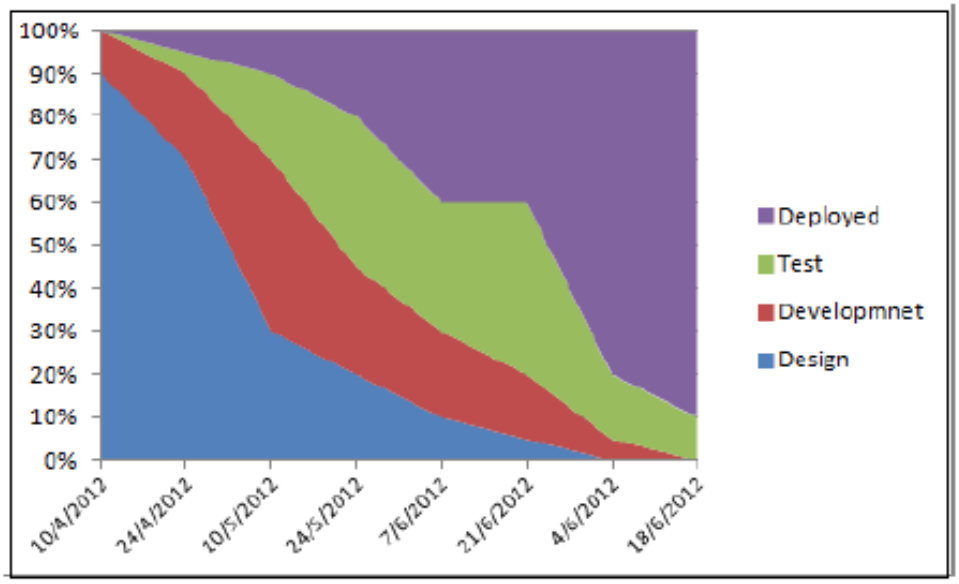

Figura 4.8: Diagrama: Cumulative Flow [11]

O Cumulative Flow não só é útil para o controlar o progresso, descrevendo o trabalho em andamento (WIP), mas também para aumentar a produtividade e na redução do prazo de entrega alcançando às necessidades dos clientes [27].

12. ***Lead Time:*** Tempo gasto para se entregar um produto desde o momento em que sua solicitação é iniciada. Segundo Anderson [3], uma métrica fundamental para sabermos o quanto é previsível é a nossa organização na entrega, considerando as definições de classe de serviço.

13. ***Cycle Time:*** Tempo para execução de todos os passos de uma atividade, em outras palavras, seria o período requerido para produzir uma parte ou completar um processo, ao tempo de medição real.

14. ***Defect Density:*** Indica o número de defeitos por uma unidade. É normalmente utilizado para identificar a quantidade de defeitos por cada mil linhas de código, ou seja, é o número de defeitos encontrados por ferramentas de análise estáticas pelo KLOC. [37]

15. ***Bug Distribution:*** Esta métrica busca descobrir a distribuição de bug no código e identificar módulos que estão propensos a este tipo de problema. [18] Um exemplo deste tipo de gráfico pode ser visto na Figura4.9, retiradas do artigo [18].

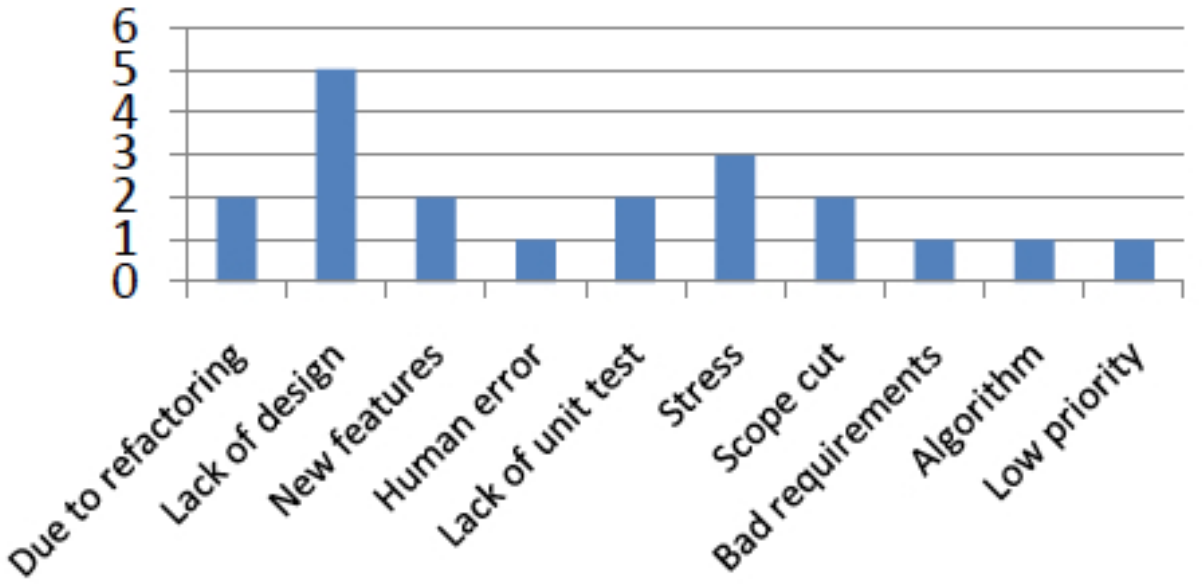

Figura 4.9: Bug Distribution: Amostragem da causa raiz

4.2 Análise das Métricas Selecionadas

Antes do mapeamento é necessário realizar o fechamento do conjunto de métricas que serão utilizadas no mesmo. Ou seja, nesta seção será explanado quais métricas mais relevantes e consequentemente elas vão ser aplicadas no suporte aos princípios ágeis.

Outro item que precisa ser delineado é que nem todas as métricas encontradas na revisão sistemática serão empregadas no mapeamento. E outras métricas consideradas clássicas, por

serem conceituadas por autores de renome precisam ser incluídas, pois durante a pesquisa não foram incluídos livros, afinal as pesquisas foram feitas em bases mais voltadas a jornais, revistas, monografias entre outros. Mas devido a importância aferida para as referências provenientes de livros é essencial a adição dos mesmos neste projeto de conclusão de curso.

Uma métrica encontrada na revisão sistemática que será excluída é o LOC, afinal apesar de ser popular ela não mostra tanta confiança como as outras métricas encontradas. Uma das explicações plausíveis é que a quantidade de código gerado muitas vezes depende da linguagem que está sendo utilizada, o que pode gerar uma disparidade se formos realizar essa contagem de linhas de código.

O *Throughput*, métrica Lean, que mede justamente a vazão. Em outras palavras, pode ser descrita como a média do número de itens produzidos em um determinado período de tempo. Esta definição é justamente a mesma dada para a métrica de velocidade do ágil. Como ambas possuem a mesma interpretação, consequentemente não é necessário adicionar ambas no conjunto escolhido de métricas. Assim, a denominação escolhida para este projeto foi a velocidade.

Outras mecanismo de medição como o *Story Points* e o WIP foram também removidos do conjunto de métricas, pois ambos podem ser inclusos em outras métricas selecionadas. O WIP pode ser visto no gráfico *Cumulative Flow*, ele descreve o trabalho em andamento [11]. Já o *Story Points* pode ser realizado por exemplo quando é utilizado o *Burndown*.

Uma métrica que é bastante utilizada no Lean é o *Takt-time*, e apesar de ter tido poucos artigos que a referenciavam, ela se mostra como uma métrica viável e segundo Khadem [10], esta métrica é uma base para mensurar a implementação do Lean. O *Takt time* é definido como o tempo disponível para a produção dividido pela demanda de mercado, em outras palavras, é o tempo de trabalho disponível, dividido pelo número de unidades concluídas é nesse período de tempo. Esta métrica muitas vezes é referida como o "batida"da planta. normalmente, é um momento teórico que a fábrica deve atender a atingir o rendimento diário planejado [8]. Na Figura 4.10, retirado do site, representa uma fórmula para esta métrica.

$$Takt\ Time = \frac{T_A}{D_R} = \frac{Available\ Work\ Time\ /\ Unit\ of\ Time}{Customer\ Demand\ Rate}$$

Figura 4.10: Representação da fórmula para o Takt Time

Anderson, considerado um autor clássico, em seu livro "Kanban - Mudança Evolucionária de Sucesso para Seu Negócio de Tecnologia"[3], cita algumas métricas que não foram encontradas na revisão sistemática como:

- **Problemas e Itens de Trabalho Bloqueados:** Representa um diagrama de fluxo acumulado de impedimentos reportados sobreposto a um gráfico da quantidade de trabalho-em-progresso que se tornaram bloqueados, como pode ser visualizado na Figura 4.11.[3].

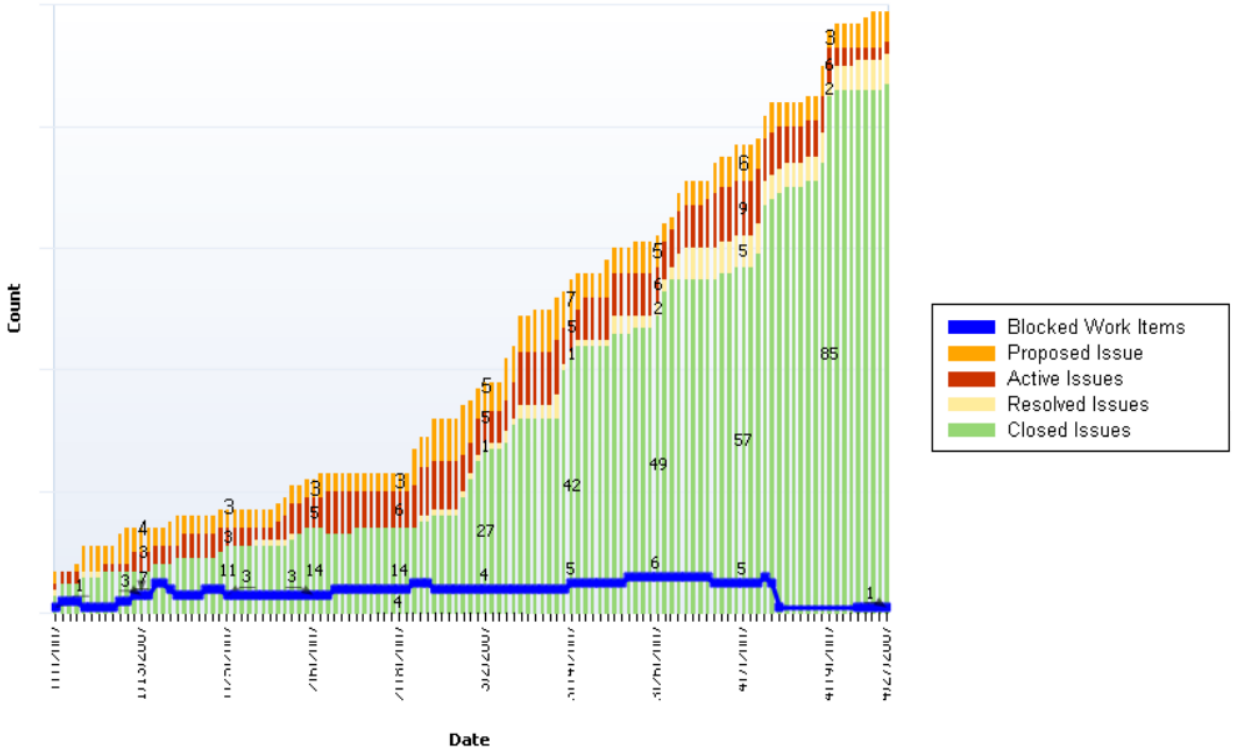

Figura 4.11: Exemplo gráfico de problemas e itens de trabalho bloqueados

Com este gráfico é possível visualizar o quanto a organização consegue identificar, re-

portar e gerenciar problemas bloqueados e seus impactos. Se o desempenho de entrega na data é ruim, o gráfico mostrará alguns impedimentos que foram descobertos e não resolvidos rápido o suficiente [3].

- **Qualidade Inicial:** Defeitos atuam como custo de oportunidade e afetam o *Lead time* e o rendimento do sistema Kanban. Logo, é necessário reportar o número de defeitos escapados como um percentual em relação ao WIP total e rendimento. Na Figura 4.12, baseada no gráfico mostrado no livro de Anderson [3], pode ser visto que a taxa de defeito aproximasse a zero .

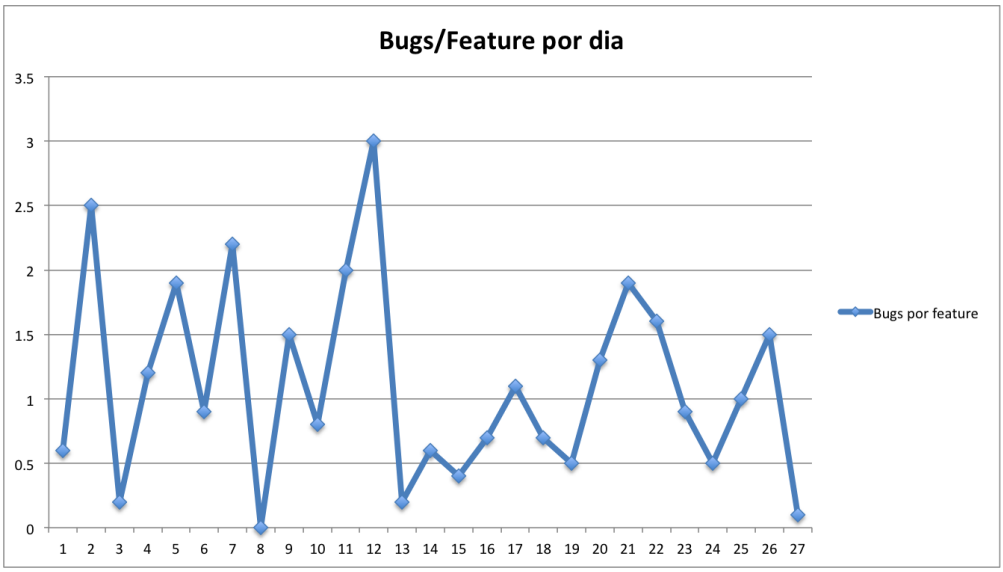

Figura 4.12: Gráfico: Defeitos por feature

A métrica de rendimento citado por Anderson ([3]), é muito similar a métrica Ágil "velocidade", afinal ela indica quantas *user stories*, ou *story points* foram finalizadas num período de tempo determinado. Logo, está métrica estará inclusa quando for mencionado a velocidade durante o mapeamento.

Apesar de todo o estudo e análise elaborado para obter as métricas é preciso considerar que existem outras além das mencionadas neste projeto de conclusão de curso. E que este trabalho selecionou um conjunto de acordo com uma pesquisa que aplicou a metodologia de Revisão Sistemática e alguns livros como o de Anderson para enriquecer ainda mais as bases

de dados. Consequentemente, podem existir métricas que deem suporte a algum princípio, porém não se encontram neste projeto de conclusão de curso.

Em resumo foram selecionadas algumas métricas que para este projeto se mostraram relevantes e que trariam um ganho para a trabalho de conclusão de curso. Logo, após toda a análise realizada para selecionar o conjunto de métricas, as escolhidas foram:

- Velocidade
- Burndown
- Burnup
- Running Tested Features
- EBV
- Lead time
- Takt-time
- Cycle time
- Cumulative Flow
- Cumulative Number of Defects
- Problemas e itens de trabalho bloqueados
- Defect Density
- Bug Distribution
- Qualidade Inicial (bug por *feature*)

Capítulo 5

Mapeamento

Neste capítulo é realizado o mapeamento entre cada princípio ágil, atribuindo ao mesmo as métricas ágeis, incluindo as métricas Lean, mais utilizadas (que foram definidas no capítulo anterior).

5.1 Análise e considerações

Nesta seção será respondida a questão proposta no projeto de conclusão do curso. Toda a análise e motivos cabíveis para que uma determinada métrica ágil dê suporte a um certo princípio. As métricas que serão utilizadas neste mapeamento foram elegidas no capítulo anterior de acordo com a análise qualitativa das mesmas.

Para facilitar a visualização do mapeamento a tabela 5.1 mostra em alto nível quais são os princípios (retirados do manifesto ágil [14]) e as métricas que dão suporte aos mesmos. E cada subseção a seguir apresentará um princípio e a justificativa de ter uma determinada métrica como seu pilar.

-	**Princípio**	**Métrica(s)**

1	Nossa maior prioridade é satisfazer o cliente, através da entrega adiantada e contínua de software de valor.	*Burn-down*, *Burn-up*, Velocidade, *Running Tested Features*, EBV.
2	Aceitar mudanças de requisitos, mesmo no fim do desenvolvimento. Processos ágeis se adequam a mudanças, para que o cliente possa tirar vantagens competitivas.	*Burn-down*, *Burn-up*.
3	Entregar software funcionando com freqüencia, na escala de semanas até meses, com preferência aos períodos mais curtos.	Velocidade, *Burn-up*, *Burn-down*, *Lead time*, *Takt time*
4	Pessoas relacionadas à negócios e desenvolvedores devem trabalhar em conjunto e diariamente, durante todo o curso do projeto.	*Cumulative Flow*
5	Construir projetos ao redor de indivíduos motivados. Dando a eles o ambiente e suporte necessário, e confiar que farão seu trabalho.	*Cumulative Flow*, Velocidade, *Burn-down*, *Burn-up*, *Takt Time*
6	O Método mais eficiente e eficaz de transmitir informações para, e por dentro de um time de desenvolvimento, é através de uma conversa face a face.	*Cumulative Flow*
7	Software funcional é a medida primária de progresso.	*Burn-down*, *Burn-up*, *Cumulative Flow*, Velocidade, *Running Tested Features*, *Lead time*, *Cycle time*
8	Processos ágeis promovem um ambiente sustentável. Os patrocinadores, desenvolvedores e usuários, devem ser capazes de manter indefinidamente, passos constantes.	Velocidade, *Takt time*, *Lead time*, Problemas e itens de trabalho bloqueados

9	Contínua atenção à excelência técnica e bom design, aumenta a agilidade.	*Cumulative Number of Defects*, *Defect Density*, *Bug Distribution*, *Running Tested Features*, Qualidade inicial
10	Simplicidade: a arte de maximizar a quantidade de trabalho que não precisou ser feito.	*Lead time*, *Cycle time*, *Burn-down*, *Burn-up*
11	As melhores arquiteturas, requisitos e designs emergem de times auto-organizáveis.	*Cumulative Flow*, *Cumulative Number of Defects*, Problemas e itens de trabalho bloqueados
12	Em intervalos regulares, o time reflete em como ficar mais efetivo, então, se ajustam e otimizam seu comportamento de acordo.	*Burn-down*, *Burn-up*

Tabela 5.1: Conjunto de métricas que dão suporte aos 12 princípios

5.1.1 1º Princípio

Este princípio enfatiza a satisfação do cliente e além disso a entrega antecipada e contínua de software de valor. Ou seja, há um destaque no *Business Value*, algo bem mais alto nível do que por exemplo as *Story Points*. Aqui o que é o valor para o cliente fica em foco. Entregar este valor o mais cedo possível ajuda o cliente a entender o que vai trazer mais valor ainda em um futuro próximo.

Um dos gráficos sugeridos foi o *Burn-down* e o *Burn-up* que ambos tratam esforço, representado na linha vertical, e o tempo no eixo horizontal. Mas, nesse princípio o valor do negócio está em ênfase, ou seja, é necessário para ambos os gráficos analisem o esforço como sendo o *Business Value*. E o tempo sendo a *Sprint* (ou iteração). Focando a análise do gráfico no valor do negócio então consequentemente atingiremos a satisfação do cliente, consequentemente estas métricas se encaixam perfeitamente neste princípio. Para uma melhor

visualização deste suporte vamos analisar o gráfico de *Burndown* da Figura 5.1.

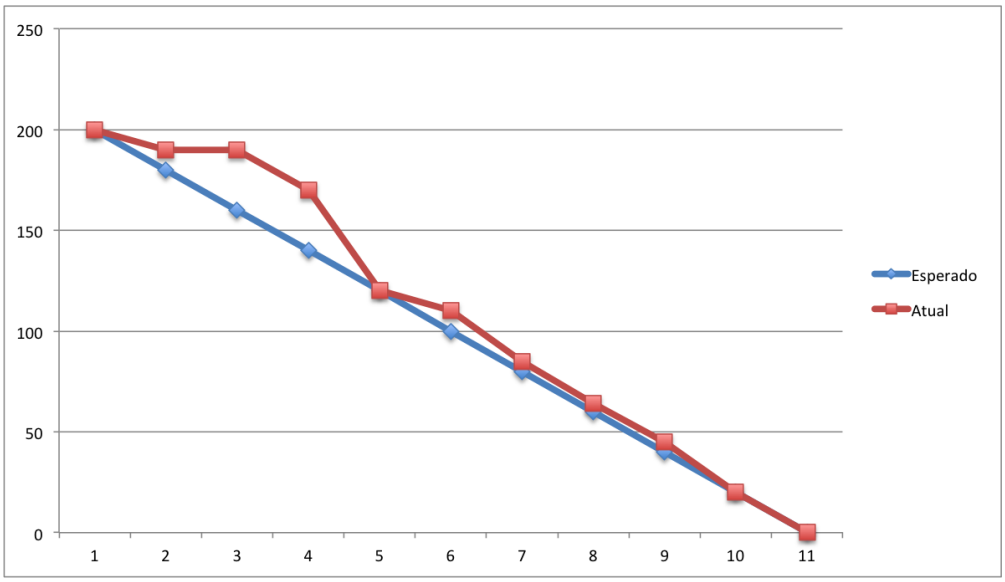

Figura 5.1: Release de Burn-down

No gráfico da Figura 5.1 o time começou em um projeto com onze *Sprints* que começaram com 200 pontos, neste caso os pontos são contados do *Business Value*. Na primeira interação foi entregue os pontos requiridos, mas depois vemos que a equipe começou a entregar menos, o que pode gerar uma insatisfação por parte do cliente. Com o passar das interações o time começa a seguir mais o fluxo de entrega e o reflexo disso é que a linha atual se equipara a linha esperada, ou seja, mais valor está sendo entregue ao cliente.

Outra medição que se ajusta a este ponto do manifesto ágil é a velocidade, que neste caso deve ser feita analisando o valor de negócio e não, por exemplo pelo *Story Points*. Com esta métrica também conseguimos mensurar se a entrega será feita de forma adiantada ou não e ainda analisar se está sendo de forma contínua.

Como mencionado por Jeffries ([25]) através da métrica *Running Tested Features* é possível que um sistema funciona e trará qualidade a medida que é realizado os testes de aceitação, ou seja, se os requisitos implementados correspondem às necessidades atuais do usuário, assim é totalmente factível utilizar esta forma de medição para analisar a satisfação do cliente.

O EBV, como explanado anteriormente, corresponde a soma do peso das estórias prontas

[42], levando em consideração o valor de negócio da estória neste somatório. Como o EBV pode ser definido pela percentagem do valor do negócio, logo é viável adicioná-lo a lista de métricas que dão suporte ao primeiro princípio.

5.1.2 2º Princípio

As turbulências tanto nos negócios como na tecnologia causam mudanças, que podem ser vistas como ameaças a serem evitadas ou oportunidades a serem abraçadas. Ao invés de resistir às mudanças, o enfoque ágil serve para acomodá-las da maneira mais fácil e eficiente possível. Metodologias ágeis entendem que facilitar mudanças é mais efetivo do que tentar evitá-las [17]. Em resumo, é necessário receber as mudanças de requisitos independente da etapa que o projeto se encontre [14].

Uma maneira de ver estas mudanças é quando aumentamos ou diminuímos o *Business value* ou *Story points*, por exemplo, se foram adicionados ou removidos requisitos do escopo. Para visualizarmos esta modificação o *Burn-down* e o *Burn-up*, são úteis, pois podemos traçar uma linha contendo a quantidade total de pontos, o esperado para as entregas e o que foi concluído. Quando forem adicionados mais pontos, consequentemente a quantidade de pontos deverá aumentar e o esperado para a entrega também, o que realça a mudança ocorrida.

Analisando essa situação da Figura 5.1, que representa um gráfico de *Burn-ip*, vemos que houve uma adição dos pontos total de 100 pontos para 120, logo elas precisaram ser gerenciadas para que todo o projeto seja entregue o final.

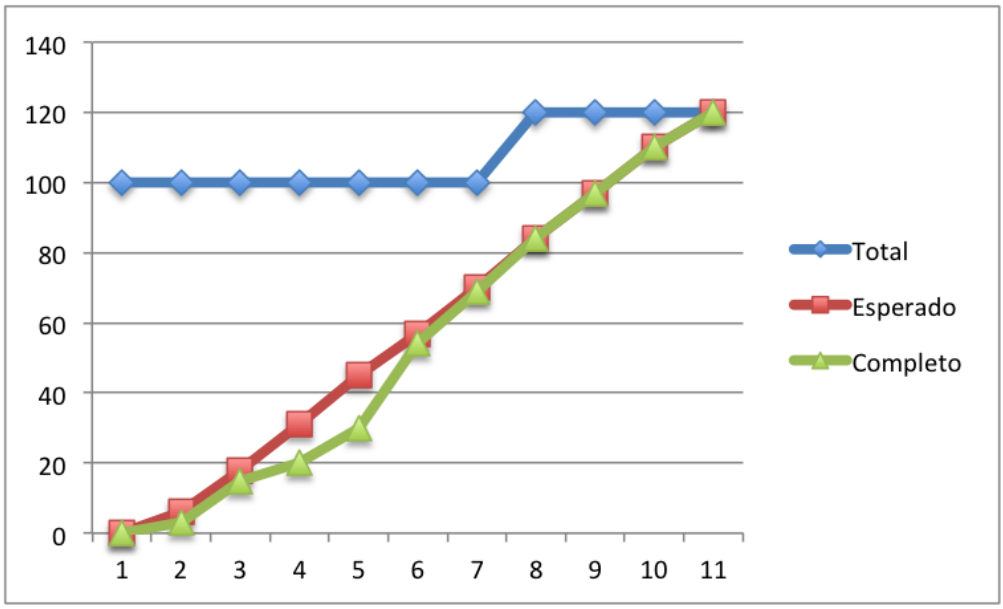

Figura 5.2: Exemplo do gráfico Burn-up com mudança de escopo

5.1.3 3º Princípio

As entregas com frequência destacadas aqui aparecem por uma razão simples: não é possível ter a certeza de que estamos avançando pelo caminho certo enquanto o software não tiver usuários reais. Ou seja, um *Feedback* constante facilita as mudanças e também o maior entendimento do negócio e consequentemente o que o cliente realmente deseja/precisa.

Uma das formas de medir esse *Feedback* é saber a velocidade que o time está entregando na iteração. Neste caso a velocidade seria medir, por exemplo, *Story points* por *Sprint* para sabermos o quanto estamos entregando de software funcionando em a uma certa frequência. Em Hayes [12] é mencionado que esta é uma métrica destinada principalmente para orientar a própria equipe de desenvolvimento para entender o realismo em compromissos de entrega. Por exemplo, se a velocidade esperada pela equipe é de 40 pontos a cada 4 semanas, e tem-se 200 pontos que devem ser liberados, então a equipe deve ter 20 semanas para concluir a *Release*. Analisando assim, fica mais fácil saber os intervalos de tempo que devem existir para a entrega de Software funcionando como também os pontos que podem ser entregues.

Outros gráficos que ajudam este princípio são o *Burdown* e o *Burnup* que se usarmos ambos no nível de estória, ou seja, por pontos, conseguimos observar o quanto de trabalho falta a ser entregue, no primeiro caso e no segundo quantidade de trabalho que já concluída. Como ambos analisam o software em funcionamento, seja o que falta ou o que foi concluído, em um período de tempo, podendo ser por iteração estas métricas se ajustam ao que o princípio retrata.

O *Lead Time* quando mencionado no LSD é justamente o tempo desde quando um pedido é feito (podendo ser uma *User Story* até o momento que ele é feito. Como estamos falando sobre o tempo do software desde quando é solicitado até o momento do seu funcionamento também conseguiremos atribuir esta métrica ao 3º Princípio, pois iremos medir a frequência de entrega destas partes do software. É importante ressaltar que no *Lead time* temos uma sútil forma de analisar o tempo do pedido, como pode ser visto na Figura 5.3.

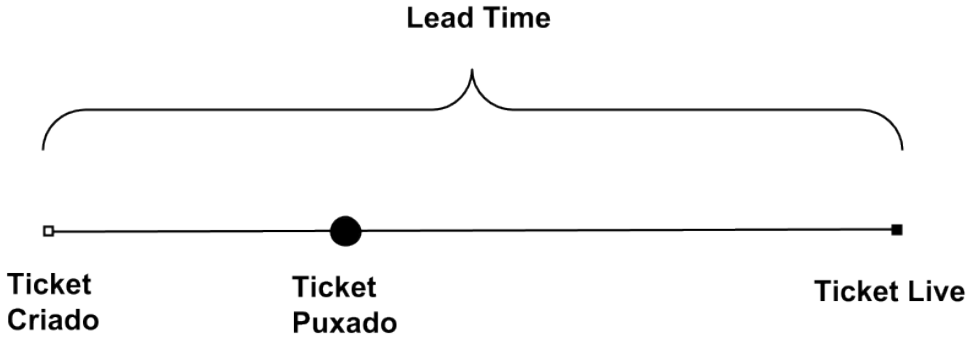

Figura 5.3: Demostração do tempo definido Lead Time

Outra maneira de medição que dá suporte a este princípio é o *Takt Time*, mas como a área é a de software, logo o tempo que é disponível para a produção seria o tempo esperado para uma nova *release*, ou seja, desde o primeiro dia do desenvolvimento da *release* até sua entrega, dividido pelos *Story points* que precisam ser entregues. Assim esta medida proporciona a frequência que entregamos software funcionando, o que é esperado para dar suporte ao princípio em questão.

5.1.4 4º Princípio

Neste princípio em questão temos que as pessoas envolvidas, seja indivíduos na área de negócios até os desenvolvedores precisam trabalhar em conjunto e diariamente, durante todo o ciclo do projeto. Em resumo o ponto principal é o grau de envolvimento dos participantes do time. O interessante aqui é monitorar a efetividade do negócio e desenvolvedores que trabalham em conjunto.

Com o *Cumulative Flow* é possível controlar os itens por estado. Através dele conseguimos observar o trabalho-em-progresso em cada estágio do desenvolvimento de software. Se o processo está fluindo corretamente, as faixas do gráfico devem ser regulares e os tamanhos devem ser estáveis. Através dele é justamente analisado o quanto cada área (por exemplo, design, análise de negócio, desenvolvimento, testes entre outros) está sendo efetiva, na verdade no final será o quanto aquele requisito pedido para aquele time foi efetivo. Logo, por meio deste gráfico é possível considerar o quanto as pessoas relacionadas estão trabalhando bem em conjunto e alcançando o propósito esperado.

Um exemplo deste andamento pode ser visto na Figura 5.4 encontrada em [30]. Através dele sabemos o quanto de trabalho está no *Backlog*, o WIP, o que está em desenvolvimento, em teste e em produção [5]. Deixando bem claro o trabalho em conjunto do time.

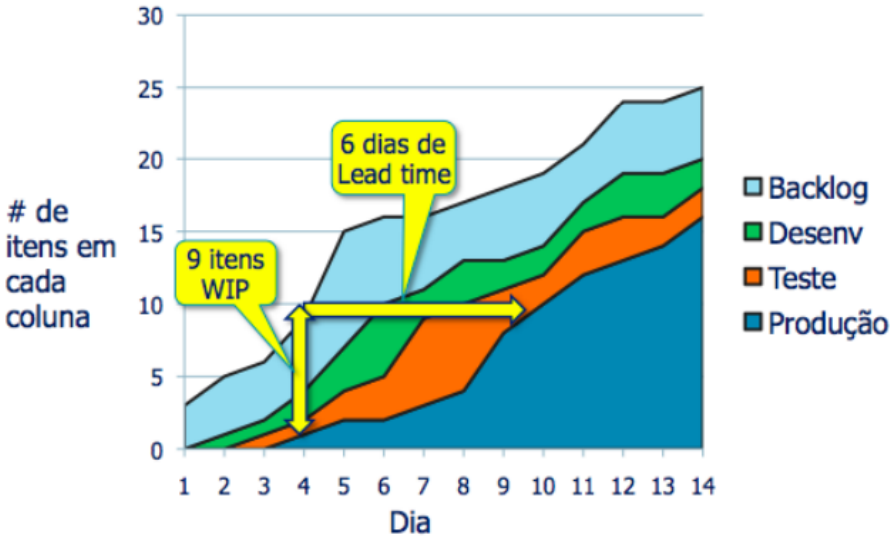

Figura 5.4: Exemplo do gráfico Cumulative Flow [30]

5.1.5 5º Princípio

O ponto crucial deste princípio é construir projetos em volta de pessoas motivadas, e a confiança como base do relacionamento. Analisando esta frase, vemos que é um tema bem subjetivo. Aqui mede-se o grau de satisfação das pessoas de uma determinada organização, o que muitos usam para averiguar o nível de motivação das pessoas é realizando questionários e fazendo quadros de satisfação do funcionário. Porém o estudo feito neste projeto de conclusão de curso trouxe métricas mais pragmáticas, e algumas propostas para tentar medir esta motivação serão definidos a seguir.

Outra métrica aplicada seria a velocidade, pois segundo [15] numerosos estudos mostram que funcionários satisfeitos prestam melhores serviços do que os funcionários que não estão satisfeitos, isto é, eles se tornam mais produtivos. Mas, é importante ressaltar que não apenas a falta de motivação gera lentidão no desenvolvimento de um determinado trabalho, existem outros motivos que podem reproduzir esse problema, mesmo assim caso haja uma diminuição ou até mesmo uma velocidade inesperada pelo time, isto pode ser um sinal que existe algum problema relacionado a falta de estímulo por membros do time. A retrospectiva

do *Sprint* pode ser um grande momento para entender qualquer discordância e insatisfação dentro da equipe.

O *Burndown* e o *Burn-up* também podem refletir o quanto o time está motivado ou não. Afinal, ambos retratam a representação gráfica de trabalho em relação ao tempo, seja para fazer ou completado. Mais uma vez, é significativo frisar que só o fato desses gráficos demostrarem que há algo errado com o andamento do projeto não expressa que a culpa é exatamente das pessoas envolvidas, mas é sim um ponto que precisa ser estudado. Assim como Burndown e o Burn-up utilizando o Takt-time e o Thoughtput, também conseguimos saber se um determinado time está sendo efetivo para aquele intervalo de tempo.

5.1.6 6º Princípio

Quando se discute sobre metodologias ágeis, inevitavelmente, surge a questão da documentação. O registro físico possui sua importância, mas o ponto chave é que haja o entendimento. A distinção entre as metodologias ágeis e aquelas centradas em documentos não é uma documentação extensiva versus sem documentação, mas sim um conceito diferente da mistura de documentação e conversação necessária para obter a compreensão [17].

Com o gráfico *Cumulative Flow* conseguimos medir, por exemplo, o WIP através da distância entre as duas linhas na área de trabalho da Figura 5.5, e caso ele aumente isso pode ser um sinal de gargalos [5]. Com este tipo de sinal uma das razões deste problema pode ser a falta de um comunicação adequada, ou seja, os envolvidos podem não estar compreendendo o desenvolvimento, o processo ou até mesmo o negócio. E através deste diagrama conseguimos até mesmo saber as etapas que mais sofrem espera.

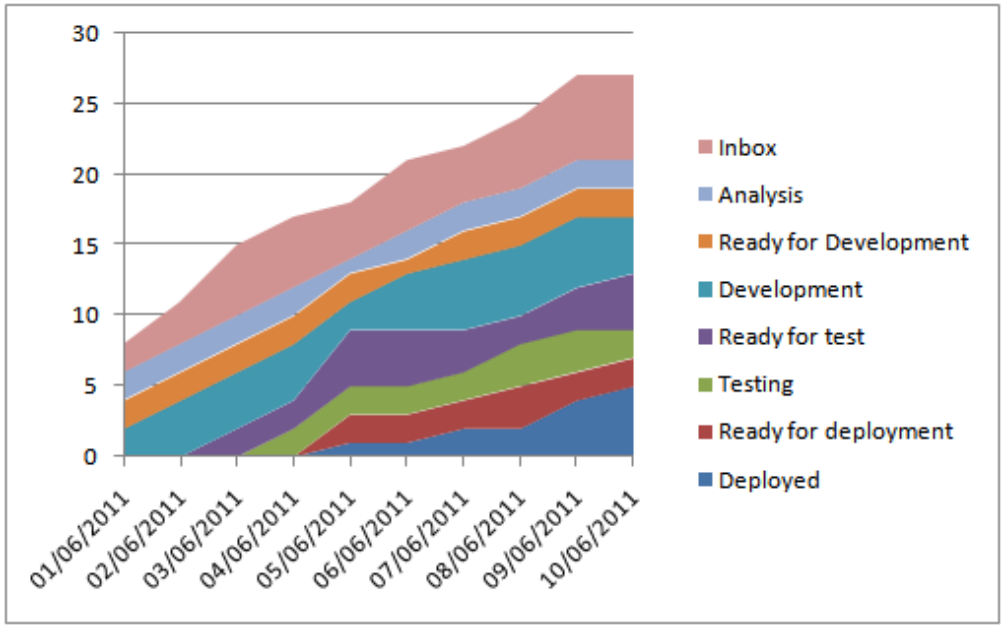

Figura 5.5: Exemplo *Cumulative flow*

É importante ressaltar que esta métrica, neste caso, dá um suporte indireto ao sexto princípio. Ela sozinha não trará todas as respostas, porém ela ajuda ao time a estudar as adversidades que possam ocorrer e consequentemente os mesmos poderão tomar uma atitude mais adequada.

5.1.7 7º Princípio

Este princípio mostra que software funcionando precisa ser a medida primária do andamento do projeto. Como a questão é mostrar que existe este progresso, logo, uma forma de medir é utilizando métricas que exibam o crescimento do projeto. Por exemplo, podemos citar o *Burn-down* e o *Burn-up*, ambos usam o tempo por *sprint* (iteração) e a quantidade de trabalho medida por *Story Points*. Assim conseguiremos mensurar se de fato há o funcionamento daquela determinada parte entregue. Por exemplo, no *Burn-up* da Figura 5.6 podemos visualizar o quanto e trabalho está feito e o que realmente era esperado para ser entregue, com esta comparação entre o *Story Points* previstas durante o planejamento da *Sprint* e as que foram realmente entregues podemos conseguir mensurar o progresso do sistema em questão.

E o *Cumulative Flow* também mostra-se como um bom meio de visualizar se aquela tarefa está pronta e quanto tempo ela leva para ser concluída.

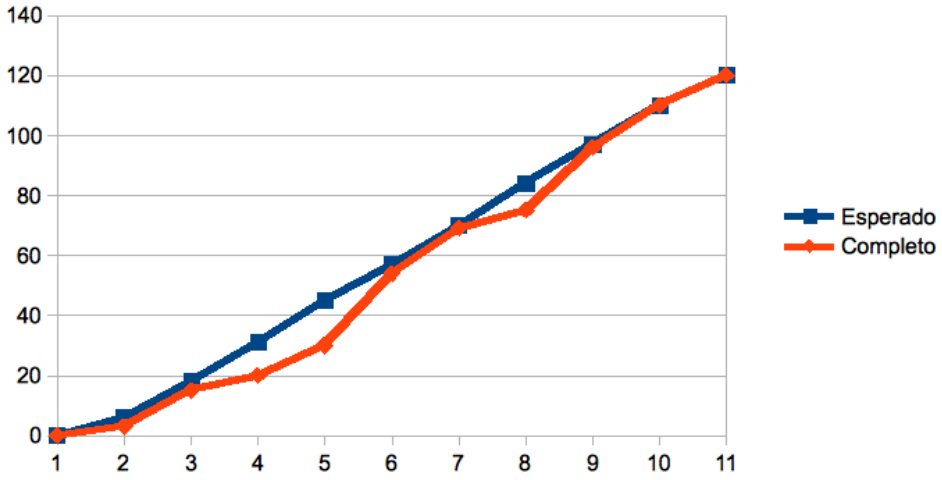

Figura 5.6: Exemplo do gráfico Burn-up

Da mesma maneira também pode se adicionar a velocidade, medindo-a através de *Story Points*. Com isso é possível mensurar o quanto o time consegue entregar de software funcional. Com o *Lead time* e o *Cycle Time* também seguem a mesma lógica, ambos expõem o tempo que um determinado cartão fica de fato pronto. O *Lead time* realiza este cálculo desde o momento que o cartão é criado até a sua entrega e o *Cycle Time* é desde quando é iniciado o desenvolvimento do cartão.

O *Running Tested Features* mostram métricas, em todos os momentos do projeto, quantos recursos estão passando em todos os seus testes de aceitação, ou seja, se de fato aquela determinada parte em estudo está de fato funcionando ela precisa ter qualidade para consequentemente mostrar se capaz de suportar o ambiente que pode ser inserido. Segundo Kulas [32], a RTF é uma série de recursos para o usuário final (pequenos e coesos fragmentos de funcionalidade que é visível para o usuário final) que estão trabalhando, passando continuamente por todos os testes de aceitação automatizados fornecidos pelos "doadores"de requisitos (os clientes), e que foram liberados, e ele é usado para rastrear o progresso do sistema. Com esta citação é possível deduzir que esta métrica consegue dar suporte ao sétimo

princípio.

5.1.8 8º Princípio

Segundo Fowler [17], a agilidade depende de pessoas que estejam atentas e criativas, e que elas consigam manter este estado para o comprimento total de um projeto de desenvolvimento de software. O desenvolvimento sustentável significa encontrar um ritmo de trabalho que a equipe possa sustentar ao longo do tempo e se manter saudável [17], tendo assim um desempenho estável e satisfatório.

Um dos fundamentos trazidos aqui é que a organização seja capaz de manter a cadência nas suas entregas. Quando colocamos justamente este ponto em questão estamos querendo medir justamente o *Takt time*. Esta métrica mostra que em um determinado tempo previsto uma certa quantidade de trabalho é entregue, ou seja, ela mede o ritmo do desenvolvimento, estando assim alinhada ao objetivo deste princípio.

O *Lead time* também traz a ideia de manter passos constantes existente no princípio, mas nesse caso o conceito utilizado é do tempo gasto para a entrega de um produto desde o instante que é solicitado. O que estaria em análise no *Lead Time* é a duração do tempo que o time como um todo consegue construir um requisito até finalizá-lo existindo no final a entrega para o cliente. Em resumo , esta métrica representa o tempo gasto para se entregar um produto desde o momento em que sua solicitação é iniciada. Segundo Anderson [3], esta é uma métrica fundamental para sabermos o quanto é previsível é a nossa organização na entrega, considerando as definições de classe de serviço.

A velocidade também dá suporte a este princípio, pois é possível analisar o volume de trabalho que foi concluído em um determinado período de tempo, assim, sabemos como a equipe se comporta e se adicionarmos a velocidade esperada a visualização se os passos estão sendo constantes ficam mais claros, afinal será que uma certa equipe está mantendo a "produtividade"que é prevista.

Outra métrica que também contribui para o apoio deste princípio é Problemas e itens de trabalho bloqueados, pois através dele conseguimos visualizar um motivo para que o ritmo

não se mostre constante. Ou seja, se ele for utilizado e apresentar-se com muitos casos em aberto e pouco concluídos é muito provável que haja uma quebra na constância das entregas, o que consequentemente mostraria que este princípio está sendo abalado.

5.1.9 9º Princípio

Neste princípio, a busca e um cuidado especial à excelência técnica e bom design são os pontos cruciais para a agilidade[14]. Ou seja, é necessário um acompanhamento incremental do desenvolvimento do produto aliado à um bom e intuitivo design.

Do conjunto de métricas elaborado neste projeto de conclusão de curso, é possível ver que as métricas que envolvem a qualidade do software. Se é mencionado uma contínua atenção, logo é necessário se atentar as métricas que também mostram defeitos que foram encontrados e o andamento da resolução dos mesmos.

Segundo Kulas, [32] o *Cumulative number of defects* e o *Defect Density* são métricas primárias para avaliar a qualidade. Diminuir o número de defeitos de um produto é uma meta importante em projetos ágeis assim como a satisfação do cliente é a maior prioridade para as equipes de software [1]. A partir destas referências e que a *Defect Density* indica a qualidade do produto (quanto menor a densidade de defeitos, maior a qualidade do produto) [32] é possível concluir que ambas as métricas se alinham com o concepção trazida no nono princípio.

O *Bug Distribution* como mostra a distribuição de *bug(s)* no código e indica quais módulos estão propícios a este tipo de adversidade [18], logo, ele contribuí para visualização da qualidade intrínseca no sistema. Assim, ela também apresentasse como uma métrica que apoia o princípio em estudo.

A Qualidade inicial, como o próprio nome sugere, também reflete o objetivo relatado no nono princípio. Com a característica de reportar o número de defeitos descobertos pelos testadores dentro do sistema e de indicar quanta capacidade está sendo gasta devido à baixa qualidade inicial [3], em outras palavras, o número de defeitos escapados pelo percentual em relação ao WIP e a velocidade [3], logo é visível o quanto esta métrica mostra-se alinhada

ao princípio em analise.

5.1.10 10º Princípio

O aspecto principal deste princípio é maximizar a quantidade de trabalho que não foi feito[14]. Ou seja, trazer a simplicidade no momento que é realizada as atividades. Segundo Islam [22], manter a simplicidade é a chave concretização das coisas e de forma rápida. E que os times ágeis devem fazer todas as tentativas para desenvolver a solução mais simples possível.

Uma das maneiras de monitorar a capacidade da equipe para maximizar a quantidade de trabalho não feito é, mantendo a atenção e atualizando o *Release Burndown*. Através deste gráfico o trabalho que ainda falta ser realizado mostra-se bem claro e dependendo do tempo que esse trabalho ainda está no gráfico conseguimos analisar se ele, por exemplo, pode ser quebrado em outros. Tendo um *Backlog* devidamente ajustado ajuda o time teve que fatorar os riscos e quaisquer desvios possíveis[15].

5.1.11 11º Princípio

Neste princípio é apontado o quanto um time auto-gerenciável consegue se desenvolver ao ponto de que as melhores arquitetura, requisitos de design surgem do mesmo [14]. O time do projeto ele precisa saber trabalhar junto, com autonomia para tomar decisões e quando necessário estar preparado para se auto-organizar de forma rápida e constante.

Com a ajuda da métrica *Cumulative Flow* é possível mensurar o quanto o time consegue trabalhar em conjunto, dividindo bem por etapas a visualização se torna mais clara e eficiente. Através dele, os gargalos, irregularidades e excesso de trabalho ficam mais evidentes [30].

Duas métricas que figuram bem os problemas que podem ser encontrados durante o desenvolvimento é justamente o *Cumulative number of defects* e os Problemas e itens de trabalho bloqueados, e se estes problemas existirem o time precisa se auto-organizar e tomar as decisões necessárias para melhoria ou correção dos problemas ou defeitos encontrados. Me-

dindo o acúmulo de atividades paradas, pode ser vantajoso para a equipe medir e visualizar explicitamente sua habilidade de lidar com esses problemas [5].

5.1.12 12º Princípio

Um grupo de estudos que se reuniram e realizaram o manifesto ágil propuseram que neste ponto a reflexão sobre como ficar mais efetivo e suas determinadas ações são crucias, e que devem acontecer com uma periodicidade [14]. É interessante que as retrospectivas ocorram e que hajam *Feedback* com todo time, isso obviamente incluí o cliente. A prática de *feedback* constantes com o cliente permite que o produto seja entregue com a qualidade desejada por ele [34].

Uma das maneiras de mensurar se o andamento do time é através dos gráficos *Burn-down* e *Burn-up*, com ambos sabemos a quantidade de trabalho e o tempo, cada uma a sua maneira. Quando realizamos a retrospectiva estes dados são importantes para sabermos do negócio e concentrar em quanto trabalho foi deixado para fazer e para tomar as decisões difíceis para manter o projeto no tempo e no orçamento [36]. Logo, existindo retrospectivas em intervalos regulares que analisem estes gráficos o time pode identificar os problemas e se ajustarem para serem mais efetivos. Durante a retrospectiva da *Sprint*, o gráfico *Burn-down* pode fornecer dados valiosos sobre como foi o *Sprint* [46].

Capítulo 6

Conclusões

Este capítulo descreve a conclusão do autor sobre todo o trabalho desenvolvido, o que incluí as principais contribuições, os trabalhos futuros, e para encerrar o mesmo, as considerações finais a serem feitas sobre o projeto.

6.1 Principais Contribuições

O objetivo deste trabalho foi explorar e escolher um conjunto de métricas para suporte aos princípios do desenvolvimento ágil de software. Inicialmente foi traçado um conjunto de métricas que enriqueceriam o projeto e que servissem para as organizações e pessoas envolvidas na área como formas de mensurar e identificar possíveis problemas. O principal ponto do projeto foi realizar um um mapeamento composto por métrica e princípio ágil, podendo uma métrica dar suporte a mais de um princípio. Este conjunto de métricas foram selecionadas através da revisão sistemática e também escolhidas pela importância na literatura Ágil, Lean e Kanban.

A reunião de determinadas formas de mensuramento foi de extrema importância para o trabalho, afinal existem várias métricas e selecioná-las, afim de diminuir o grande número encontrado, trouxe no final um ganho tanto para o trabalho como também para o meio

externo, como empresas e indivíduos incluídos neste campo de estudo. Após, agrupá-las foi necessário definir quais seriam usadas no mapeamento, esta fase também incluiu o estudo e maior detalhamento das métricas que foram selecionadas.

Concluindo esta etapa do projeto, que se mostrou necessária para que não fossem utilizadas métricas que não trouxessem um ganho real ao trabalho, a próxima fase foi estudar quais destas métricas estavam mais alinhadas aos princípios do desenvolvimento ágil de software. Foi gerada uma tabela, para que a visualização fosse mais eficiente, na qual temos os princípios e métricas que provêm suporte a sua institucionalização. Porém, só a tabela não faria sentindo sem as razões pela qual cada métrica dava suporte a cada princípio, logo as justificativas e considerações necessárias foram adicionadas.

Portanto, a questão proposta para este trabalho foi respondida, primeiramente apontando as métricas mais relevantes, e por fim, mostrando que para cada princípio do Manifesto Ágil há uma ou mais métricas que estão diretamente relacionadas ao mesmo, e que apoiam a sua institucionalização.

6.2 Trabalhos Futuros

Para enriquecer e evoluir este estudo considera-se relevante a realização de entrevistas com estudiosos e empresas que trabalham com ágil sobre o mapeamento, explicando as métricas e como elas podem apoiar os princípios, através de questionários modelados para fortalecer os resultados do trabalho.

Outro meio interessante seria a realização de experimentos para validar esta comparação teórica, sempre no sentido de contribuir com os resultados alcançados.

6.3 Considerações finais

As métricas, assim como *Agile* não devem ser o fim, é através deles que a organização deve alcançar os seus verdadeiros objetivos. Elas servem como indicadores para a melhoria

contínua do processo, e o projeto propôs justamente mostrar que esses indicadores dão suporte a determinados princípios ágeis. Assim, com uso das métricas conseguiremos aprender mais sobre o que precisa ser mensurado e identificar, assimilar e aplicar ações para que haja uma melhoria ou evolução do processo.

Referências Bibliográficas

[1] O. Aktunc. Entropy metrics for agile development processes. *2012 IEEE 23rd International Symposium on Software Reliability Engineering Workshops*, 2012.

[2] D. J. Anderson. *Agile Management for Software Engineering: Applying the Theory of Constraints for Business Results*. Prentice Hall, 2003.

[3] D. J. Anderson. *Kanban: Mudança Evolucionária de Sucesso para seu Negócio de Tecnologia*. Blue Hole Press, 2011.

[4] N. Andreeva. Lean manufacturing performance - metrics and evaluation. 2009.

[5] J. Boeg. *Kanban em 10 passos - Otimizando o fluxo de trabalho em sistemas de entrega de software*. Infoq Brasil, C4Media, 2012.

[6] M. Cohn. *User Stories Applied*. Addison-Wesley, 2004.

[7] M. Cohn. *Agile Estimating and Planning*. 2005.

[8] J. Duanmu and K. Taaffe. Measuring manufacturing throughput using takt time analysis and simulation. *Winter Simulation Conference*, 2007.

[9] Misra et al. Survey on agile metrics and their inter-relationship with other traditional development metrics. *SIGSOFT Softw. Eng. Notes*, 36(6), 2011.

[10] M.Khadem et al. Efficacy of lean metrics in evaluating the performance of manufacturing systems. 2008.

[11] T. Javdani et al. On the current measurement practices in agile software development. 2013.

[12] W. Hayes et al. Agile metrics: Progress monitoring of agile contractors, 2014.

[13] Y. Dubinsky et al. Agile metrics at the israeli air force. *Proceedings of the Agile Development Conference (ADC'05)*, 2005.

[14] K. et al. Beck. Manifesto for agile software development, 2001.

[15] M. V. L. Expedith. Measuring alignment with the principles of agile, 2014.

[16] N. E. Fenton and M. Neil. Software metrics: roadmap. *ICSE '00: Proceedings of the Conference on The Future of Software Engineering, ACM*, 2000.

[17] M. Fowler and J. Highsmith. The agile manifesto. *Software Development*, 2001.

[18] J. Gustafsson. Model of agile software measurement: A case study, 2011.

[19] N. Habra H. Ayed and B. Vanderose. Am-quick : a measurement-based framework for agile methods customisation. 2013.

[20] D. Hartmann and R. Dymond. Appropriate agile measurement: Using metrics and diagnostics to deliver business value. 2006.

[21] IEEE. Standard glossary of software engineering terminology, 1990.

[22] K. A. Islam. *Agile Methodology for Developing and Measuring Learning*. AuthorHouse LLC, 2013.

[23] M. Naim J. Naylor and D. Berry. Leagility: Integrating the lean and agile manufacturing paradigms in the total supply chain. *Int. J. Production Economics*, 1999.

[24] D. T. Jones J. P. Womack and D. Roos. *The Machine That Changed The World*. Free Press, 1990.

[25] R. Jeffries. A metric leading to agility, 2004.

[26] Jim Johnson. Roi, it?s your job! *Published Keynote Third International Conference on Extreme Programming*, 2002.

[27] P. Kai and W. Claes. Measuring the flow in lean software development. *Softw., Pract. Exper.*, 41:975–996, 2011.

[28] B. Kitchenham. Procedures for performing systematic reviews. Technical report, Department of Computer Science, Keele University, UK, 2004.

[29] H. Kniberg. Scrum e xp direto das trincheiras. *InfoQ, C4Media Inc*, 2007.

[30] H. Kniberg and M. Skarin. *Kanban e Scrum - obtendo o melhor de ambos*. 2009.

[31] O. Ktata and G.Levesque. Designing and implementing a measurement program for scrum teams: What do agile developers really need and want? *C3S2E 10 Proceedings of the Third C Conference on Computer Science and Software Engineering*, 2010.

[32] H. Kulas. Product metrics in agile software development. Master's thesis, University of Tampere School of Information Sciences, 2012.

[33] W. Li. Software product metrics. using them to quantify design and code quality. *IEEE Potentials*, 1999.

[34] D. F. S. Lira. Um conjunto de desafios e boas práticas para organizações ágeis de software com base na experiência da manufatura. Master's thesis, Universidade Federal de Pernambuco - UFPE, 2012.

[35] G. Succi M. Scotto, A. Sillitti and T. Vernazza. A non-invasive approach to product metrics collection. *Journal of Systems Architecture*, 2006.

[36] G. Meszaros and J. Aston. Adding usability testing to an agile project. *Agile Conference*, 2006.

[37] N. Nagappan and T. Ball. Static analysis tools as early indicators of pre-release defect density. *ICSE '05 Proceedings of the 27th international conference on Software engineering*, 2005.

[38] N. Oza and M. Korkala. Lessons learned in implementing agile software development metrics. *UK Academy for Information Systems Conference Proceedings 2012*, 2012.

[39] O. Salo P. Abrahamsson and J. Ronkainen. Agile software development methods: Review and analysis. *VTT Publications*, 2002.

[40] T. Poppendieck and M. Poppendieck. *Lean Software Development: An Agile Toolkit.* Addison-Wesley Professional, 2003.

[41] R. Pressman. *Engenharia de Software.* McGraw-Hill Interamericana do Brasil, 2006.

[42] D. Rawsthorne. Monitoring scrum projects with agileevm and earned business value (ebv) metrics. *CollabNet*, 2010.

[43] D. Sato. Métricas de acompanhamento para metodologias Ágeis. *Engenharia de Software Magazine*, 2007.

[44] K. Schwaber. *Agile Project Management with Scrum.* Microsoft Press, 2004.

[45] S. Shigeo. *A study of the Toyota production system from a industrial engineering viewpoint.* Cambridge: Rudra Press, 1989.

[46] C. Sims. Gráficos burn-down anotados ajudam durante as retrospectivas, 2009.

[47] J. Sutherland. Scrum handbook. *Scrum Training Institute*, 2010.

[48] K. A. Thisted. The impact of introducing agile metrics on team efficiency. *Advanced Software Engineering SASU F2013*, 2013.

[49] M. Daneva Z. Racheva and L. Buglione. Complementing measurements and real options concepts to support inter iteration decision making in agile projects. *34th Euromicro Conference Software Engineering and Advanced Applications*, 2008.

[50] R. Zornitza and D. Maya. Using measurements to support real-option thinking in agile software development. In *Proceedings of the 2008 International Workshop on Scrutinizing Agile Practices or Shoot-out at the Agile Corral*, 2008.