Tiago Martins de Alexandre

Análise do uso de quadro digital Kanban no desenvolvimento de software

São Paulo
2023

Tiago Martins de Alexandre

Análise do uso de quadro digital Kanban no desenvolvimento de software

Versão Corrigida

Dissertação apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Mestre em Engenharia

São Paulo
2023

Tiago Martins de Alexandre

Análise do uso de quadro digital Kanban no desenvolvimento de software

Dissertação apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Mestre em Ciências

Área de concentração: Engenharia de Computação

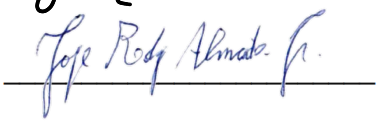
Orientador: Prof. Dr. Jorge Rady de Almeida Júnior

São Paulo
2023

Este exemplar foi revisado e corrigido em relação à versão original, sob responsabilidade única do autor e com a anuência de seu orientador.

São Paulo, 15 de setembro de 2023

Assinatura do autor: 

Assinatura do orientador: 

Catalogação-na-publicação

Alexandre, Tiago Martins de Alexandre
Análise do uso de quadro digital Kanban no desenvolvimento de software / T. M. A. Alexandre, J. R. A. J. Almeida Jr -- versão corr. -- São Paulo, 2023.
97 p.

Dissertação (Mestrado) - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Computação e Sistemas Digitais.

1.Engenharia de software 2.Métodos ágeis I.Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Computação e Sistemas Digitais II.t. III.Almeida Jr, Jorge Rady de Almeida Júnior

Dedico este trabalho aos meus avós, pais Valéria e Aristótalys, às minhas irmãs Priscila e Gabriela, à minha esposa Miriam, aos meus filhos, Lucas, Mateus e ao meu sobrinho Niko.

Agradecimentos

Ao Prof. Dr. Jorge Rady pela orientação, paciência e compreensão.

Aos colegas da pós-graduação, pela convivência e troca de experiências durante as aulas e os trabalhos.

Agradecimentos especiais a Daniel Makiyama e Rodrigo Rosauro, pelas riquíssimas contribuições ao trabalho.

Agradeço à minha esposa Miriam, pelo apoio para que este trabalho fosse concluído, pelo incentivo e paciência.

Aos meus pais, que nunca mediram esforços para facilitar o meu caminho nos estudos.

A todos da minha família, pela compreensão e incentivo para que este trabalho fosse finalizado.

À USP, pela oportunidade.

A todos que, de alguma forma, contribuíram com a evolução desta pesquisa e colaboraram para a sua conclusão.

Na viagem da vida, felicidade não é uma estação. É uma maneira de viajar.

Richard Simonetti

Resumo

O método Kanban tem sido utilizado no desenvolvimento de softwares em diversas empresas. O quadro kanban, uma das principais ferramentas do método, é construído de modo físico em diversas implementações. Este trabalho pretende analisar o uso do quadro Kanban digitalizado ou virtual que tem o potencial, tanto para melhorar o vínculo entre o que é reportado nas reuniões diárias e o que é de fato executado, quanto para permitir a extração de métricas de maneira automatizada. Foram analisados dados de, aproximadamente, dois anos de utilização de quadro digital Kanban em uma empresa de desenvolvimento de software, com o objetivo de se verificar a aderência das informações apresentadas no quadro em relação ao real estágio de desenvolvimento, por meio da comparação desses dados com os obtidos nos controladores de versionamento de código fonte. Desta forma, foi possível observar algumas divergências entre as marcações, o que demonstra ainda haver espaço para melhorias na utilização do método e no uso do quadro digital.

Palavras-chave: *Kanban, metodologia ágil, quadro kanban, quadro digital kanban, quadro virtual kanban*

Abstract

The Kanban method has been widely used in software development in various companies. The Kanban board, one of the main tools of the method, is physically constructed in several implementations. This work aims to analyze the use of the Kanban board in a virtual (or digitized) manner, which has the potential to improve the link between what is reported in daily meetings and what is executed, as well as to allow the extraction of metrics in an automated way. Data from approximately two years of using a virtual Kanban board in a software development company were analyzed to verify the adherence of the information presented on the board to the actual stage of development, by comparing this data with that obtained from source code versioning controllers. In this way, it was possible to observe some divergences between the markings, which demonstrates that there is still room for improvement in the use of the method and the virtual board.

Keywords: *Kanban, agile methodology, kanban board, digital kanban board, virtual kanban board*

Lista de Ilustrações

Lista de Abreviaturas e siglas

QP	Questão de Pesquisa
DQ	Digitalização de quadro
WIP	*Work in Progress*
XP	*Extreme Programming*
RUP	*Rational Unified Process*
QA	*Quality Assurance*

Sumário

1 Introdução

O presente trabalho se propõe a analisar o método Kanban aplicado à gestão do desenvolvimento de software quando se utiliza o quadro digital para o acompanhamento das atividades. A seguir, é descrito o problema de pesquisa, o objetivo e a justificativa para este trabalho.

1.1 Descrição do problema de pesquisa

A crescente complexidade dos sistemas computacionais a serem desenvolvidos gerou inúmeras dificuldades para os desenvolvedores de software ao longo dos anos. Para lidar com sistemas complexos, é bastante comum que se proponha sua divisão em pequenos componentes, em que cada um pode representar uma tarefa ou atividade. Dessa forma, um sistema é completo quando se completam todas as tarefas que compreendem o seu desenvolvimento.

(Vacanti, 2015) indica que é bastante comum que, após iniciada, uma tarefa permaneça cerca de 85% do tempo em alguma fila. Reduzir o tempo em fila das tarefas é a principal proposta do método Kanban. Esse método, derivado do Lean para o gerenciamento de produção de bens intangíveis, como software, apresenta o quadro de atividades (*kanban Board*) como sendo a principal ferramenta para se conseguir visualizar o fluxo de trabalho de uma determinada organização. (Anderson, 2010) sugere que este, em tempo, é o primeiro dos cinco elementos de uma implantação bem-sucedida do método.

O *kanban board* expõe o fluxo de trabalho o que permite a identificação de gargalos no processo. Assim, é possível estabelecer limites neste fluxo, permitindo a aceleração das entregas, bem como a redução dos riscos técnicos e de negócios (Middleton & Joyce, 2012). Devido a diversos fatores como a diversidade de localização física dos membros da equipe e a dificuldade de elaboração de métricas utilizando material não digitalizado, algumas empresas apresentaram ferramentas para auxiliar na digitalização do quadro (DQ) de atividades, tais como *LeanKit* e *Jira*.

Uma das características do Kanban é a sua inicial adesão ao processo de trabalho que requer poucas mudanças (Nevenka & Saso, 2015). A DQ pode enrijecer

um pouco esta adesão devido a restrições inerentes às ferramentas escolhidas, algumas das quais estão listadas no APÊNDICE A.

O Método Kanban, aplicado ao desenvolvimento de software, tem evoluído bastante entre as empresas e muitos de seus conceitos estão sendo testados empiricamente. Os resultados destas experiências estão orientando a evolução do método. O presente trabalho propõe analisar dados de dois anos de acompanhamento de métricas de Kanban, em uma empresa de desenvolvimento de software que utilizou quadro digital durante todo o período. Os dados extraídos da ferramenta de acompanhamento digital foram cruzados com os dados da ferramenta de versionamento de código.

A técnica do erro quadrático médio foi utilizada para avaliar se os dados informados no quadro digital representavam o trabalho gerado efetivamente, permitindo que fosse inferido o nível de confiança das informações extraídas da ferramenta. Contudo, não foi encontrado na literatura nenhum estudo de caso que evidencie se o uso de quadro digital de kanban reflete, para os gestores, o que de fato acontece no dia a dia do desenvolvimento.

1.2 Objetivos

Este trabalho tem como objetivo analisar o método Kanban e, mais detalhadamente, o uso de quadro digital para acompanhamento do desenvolvimento de software e mostrar se pode haver divergências entre o que é falado nas reuniões diárias e o que é de fato executado na criação do software. A digitalização e utilização do quadro é um primeiro passo para melhorar este vínculo, permitindo também viabilizar a geração automática de métricas para tomada de decisões.

No método Kanban tradicional, o desenvolvedor é responsável por registrar ou atualizar, no quadro físico, o início, revisão ou conclusão de uma atividade por meio da mudança de sua posição. Como há reuniões diárias de acompanhamento e atualização do quadro, é normal que haja, no máximo, até um dia de defasagem entre a movimentação das tarefas e os indicadores de versionamento de código.

Com a utilização de um quadro digital, as movimentações são registradas e salvas de forma permanente em banco de dados, criando-se assim um indicador adicional. Este trabalho visa analisar a defasagem entre estes dois indicadores. Se

houver discrepância entre eles, pode haver, portanto, indício de que o quadro e as métricas possam não estar refletindo a realidade da equipe de desenvolvimento, o que pode gerar relatórios gerenciais equivocados e criar dificuldades no acompanhamento das atividades.

Ainda, outro objetivo desta dissertação é o de abrir caminho para uma sólida integração entre o sistema de versionamento de código e o quadro digital de acompanhamento de atividades de modo que o quadro se torne uma consequência das ações executadas em código. Busca-se, assim, mitigar a eventual discrepância entre os indicadores e permitir que o quadro reflita, de fato, o que está sendo feito e quando.

1.3 Justificativa

A baixíssima eficiência do fluxo no desenvolvimento de software é um problema gerencial que atinge muitas empresas, conforme sinaliza (Vacanti, 2015), ensejando técnicas de desenvolvimento ágeis que possibilitem mitigar este problema. O método Kanban se apresenta, neste contexto, como uma destas técnicas, valendo-se de um quadro de controle de atividades para tornar evidente onde estão os gargalos no processo de desenvolvimento. Se as informações desse quadro não forem verossímeis, a aplicação do método fica comprometida, pois as métricas ficam distorcidas, conduzindo desenvolvedores e gestores à tomada de decisões equivocadas.

Como parte do método, reuniões diárias são realizadas para, entre outras ações, se atualizar o quadro e evitar discrepâncias. Por se tratar, geralmente, de um quadro físico em lousa ou quadro branco com adesivos identificados, pode-se prever a existência de alguns riscos, que evidenciam fragilidades no controle das atividades:

a) Possível falha na fixação do adesivo correspondente a uma atividade, causando queda e possível desvio em seu rastreamento;
b) Perda do adesivo por manutenção/limpeza por parte de terceiros;
c) Falha na movimentação do adesivo, ocasionada por eventual distração dos participantes na reunião, ou mesmo por falta ou ausência de um ou mais responsáveis pela atividade no momento da reunião;

d) Cancelamento da reunião de atualização das atividades no quadro por motivos diversos.

Tais riscos podem ser mitigados com a proposta da digitalização do quadro, melhorando a experiência da gestão do Kanban na busca pela melhoria da eficiência do fluxo de desenvolvimento de software.

1.4 Estrutura do trabalho

Este trabalho está estruturado em oito capítulos, incluindo este, de caráter preliminar e introdutório.

O Capítulo 2 aborda o método científico utilizado, as hipóteses de pesquisa consideradas e que métodos foram utilizados para se chegar às conclusões.

O Capítulo 3 apresenta alguns dos métodos ágeis mais utilizados na indústria.

O Capítulo 4 detalha os principais estudos relacionados aos métodos Kanban, também, apresenta o estado da arte de sua utilização.

O Capítulo 5 versa sobre os detalhes de implementação do método Kanban, suas formas de aplicação e suas principais características.

O Capítulo 6 apresenta um estudo de caso utilizando o quadro digital de kanban.

O Capítulo 7 apresenta os resultados deste trabalho e possíveis trabalhos futuros.

2 Metodologia

Escolher uma metodologia para pesquisa em desenvolvimento de software não é uma tarefa simples. Os métodos empíricos, a postura filosófica e o problema de pesquisa influem muito na decisão da metodologia e não há uma receita pronta para a escolha (Easterbrook, Singer, Storey, & Damian, 2008).

Devido à importância das atividades humanas no desenvolvimento de software, muitos métodos de pesquisa apropriados à Engenharia de Software são oriundos de disciplinas que estudam o comportamento humano, seja no nível pessoal, ou no nível de equipes e organizacional. Assim, é prudente considerar a postura filosófica do pesquisador e o problema de pesquisa para se escolher um método apropriado, bem como para entender suas limitações e endereçá-los durante o desenvolvimento (Easterbrook, Singer, Storey, & Damian, 2008).

2.1 Métodos empíricos de pesquisa

Os mais relevantes métodos empíricos de pesquisa em Engenharia de Software, ainda segundo (Easterbrook, Singer, Storey, & Damian, 2008) , são:

a) Experimentos controlados (*Controlled experiments*): uma investigação sobre uma hipótese testável, em que uma ou mais variáveis independentes são manipuladas e tem seus efeitos medidos em uma ou mais variáveis dependentes.
b) Estudo de caso (*Case Studies*): um método empírico que investiga um fenômeno em seu contexto real de execução. Estudos exploratórios são usados para uma investigação inicial da qual derivam hipóteses e possíveis construções teóricas. Estudos confirmatórios também são realizados para testá-las.
c) Pesquisa de opinião (*Survey research*): usada para identificar características de uma população de indivíduos, estando associada ao uso de questionários para levantamento de dados.
d) Pesquisa etnográfica (*Ethnographies*): é a forma de pesquisa focada na sociologia de significado, por meio de observação de campo. O objetivo

é estudar uma comunidade de indivíduos para entender como seus membros entendem a sua própria interação social. No contexto de desenvolvimento de software, esta técnica pode ser usada para entender como a cultura de uma empresa é construída para auxiliar no desempenho técnico da equipe como um todo.

e) Pesquisa de ação (*Action Research)*: contexto em que os pesquisadores tentam resolver um problema do mundo real enquanto, simultaneamente, estudam a experiência de resolver o problema.

2.2 Posturas filosóficas

Sobre as posturas filosóficas, tem-se: (Easterbrook, Singer, Storey, & Damian, 2008)

a) Positivismo: todo conhecimento precisa estar baseado em uma inferência lógica de um conjunto de fatos observáveis.
b) Construtivismo: o conhecimento científico está associado ao seu contexto humano de interpretação. Teorias propostas devem estar sempre associadas ao contexto aos quais foram elaboradas.
c) Críticos teóricos: julgam o conhecimento científico baseado na possibilidade deste conhecimento libertar as pessoas de um pensamento restritivo. Entendem a pesquisa como um ato político porque julgam que o conhecimento empodera diferentes grupos na sociedade. No contexto de engenharia de software, incluem-se pesquisas que buscam desafiar as percepções sobre as práticas existentes. Geralmente, utilizam-se de casos de uso para dar atenção a coisas que necessitam ser mudadas.
d) Pragmatismo: consideram que todo conhecimento é incompleto e aproximado, seu valor depende dos métodos utilizados na sua obtenção. O conhecimento é tão bom quanto a sua utilidade em resolver problemas reais. Pragmáticos valorizam mais o conhecimento prático ao conhecimento teórico e são favoráveis ao uso misto de metodologias de pesquisa.

2.3 Questão de pesquisa

Quadros físicos de controle de atividade são instrumentos pelos quais o método Kanban é caracterizado no contexto de sua aplicação (Anderson, 2010). Os quadros virtuais de Kanban agilizam a extração de métricas que são fundamentais para a melhoria contínua, devido à possibilidade de se compartilhar informações estruturadas digitalmente com diversas ferramentas. Isto posto, traz-se a indagação: quadros virtuais de Kanban podem ser utilizados no lugar de quadros físicos, possibilitando o acompanhamento das atividades de desenvolvimento, sem prejudicar a eficácia das reuniões diárias e a gestão? Em suma, quanto o uso do quadro digital de Kanban pode afetar o acompanhamento das atividades de desenvolvimento de software?

Se a resposta para esta pergunta for positiva, pode-se entender que a migração para o uso do quadro digital do Kanban é, de fato, a evolução da aplicação do método, abrindo espaço para novas discussões sobre os benefícios da digitalização. Tais discussões podem trazer à baila tópicos como a extração automatizada de métricas e a configuração do vínculo do quadro digital com o software de controle de versão.

Outra questão de pesquisa (QP) é como o quadro digital pode ser usado, consolidando, assim, duas questões a serem discorridas neste trabalho:

QP1: Como o quadro digital Kanban pode ser usado para o acompanhamento de atividades?

QP2: O uso do quadro digital afeta o acompanhamento do andamento das atividades?

2.4 Proposta de metodologia

Considerando uma postura pragmática, para responder às questões de pesquisa supracitadas propõe-se a seguinte metodologia:

a) Revisão bibliográfica para, primeiramente, levantar o estado da arte dos métodos ágeis, das aplicações do método Kanban no desenvolvimento de software e, ainda, verificar a eventual utilização de quadro digital de

acompanhamento de atividades em outras indústrias e identificar possíveis estudos similares;

b) Acompanhamento de um projeto de desenvolvimento de software que utilizou a digitalização do quadro kanban;
c) Discussão dos resultados;
d) Propor trabalhos futuros.

2.5 Método para análise dos insumos coletados no Estudo de caso

O Estudo de caso consistiu no acompanhamento, por dois anos, de um projeto ainda em execução, há mais de quatro anos, por uma equipe de aproximadamente 20 pessoas, cujo objetivo foi a migração de software utilizando o quadro digital de acompanhamento (*Jira Kanban Board*) para o planejamento das atividades. Também foi utilizado um software de controle de versão de código fonte (*Git - Bitbucket*).

Assim, foi utilizada a técnica do erro quadrático médio para verificar se o fluxo das atividades registradas no quadro digital de acompanhamento de atividades (*Kanban Board*) reflete de maneira efetiva o fluxo de desenvolvimento registrado no software de controle de versão.

Se os valores forem baixos, pode-se inferir que não houve negligência na atualização do quadro e concluir, portanto, que não houve prejuízos ao acompanhamento do projeto por parte dos gestores. As pequenas discrepâncias que forem, eventualmente, encontradas, mostrarão se há espaço para melhorar o processo, vinculando, talvez, o andamento do quadro ao software de controle de versão através de ferramentas a serem pesquisadas.

3 Métodos Ágeis

A indústria de software, desenvolvida ao longo de várias décadas com diversos métodos e abordagens distintos para atender a competitividade do mercado e a satisfação dos clientes, sempre sofreu com a dinâmica das mudanças de mercado e das necessidades dos clientes (Salma & Gómez, 2021).

Tais mudanças acarretam problemas e conflitos nos planos e contratos de desenvolvimento que precisam ser constantemente atualizados e criam insatisfações tanto por parte dos clientes quanto por parte dos desenvolvedores.

Diversas metodologias vêm sendo propostas desde a publicação do manifesto ágil prometendo maior foco em indivíduos e interações, software funcionando e colaboração dos clientes em detrimento de contratos, planos e ferramentas. Porém, há muitas variações de processos, papéis e nomenclaturas entre estas propostas. Para uma metodologia ser considerada ágil, ela precisa: 1. Ter entregas incrementais; 2. Promover colaboração entre clientes e desenvolvedores em estreita e constante comunicação; 3. Ser simples de implementar, aprender, modificar e documentar; 4. Ser capaz de absorver mudanças de última hora (Abrahamsson, Salo, Ronkainen, & Warsta, 2002).

A agilidade refere-se a responder rapidamente a mudanças, seu reconhecimento de pessoas como os principais impulsionadores do sucesso do projeto, juntamente com um foco intenso na eficácia e manobrabilidade (Highsmith & Cockburn, 2001).

Neste contexto, apresentam-se dois métodos ágeis, *Scrum* e *Extreme Programming*, como base de comparação com o método Kanban que é mais detalhadamente apresentado.

3.1 Scrum

O Scrum é um framework baseado no empirismo e *Lean Thinking*, que ajuda pessoas, times e organizações a gerar valor por meio de soluções adaptativas para problemas complexos. Fundamenta-se em 3 pilares empíricos: a transparência: que diz que o trabalho deve ser visível tanto para quem executa quanto para quem recebe, a inspeção: que verifica se os artefatos e o progresso em direção às metas estão

sendo cumpridos através de cinco tipos de eventos e a adaptação que é a atuação do time para corrigir um eventual desvio detectado durante uma inspeção (Schwaber & Sutherland, 2020).

3.1.1 Papéis

Há apenas 3 papéis no Scrum: o *Product Owner*, o Time de desenvolvimento e o *ScrumMaster*. Toda responsabilidade do projeto é dividida nestes três papéis. O *Product Owner* é responsável por definir e quebrar o escopo de trabalho em atividades que devem ser elencadas em uma lista priorizada chamada de *backlog* e que contém os requisitos do projeto escrito na forma de estórias de usuário. O time de desenvolvimento é responsável por desenvolver as funcionalidades. O *ScrumMaster* tem a função de ensinar as práticas do Scrum e garantir que os eventos do Scrum aconteçam (Schwaber, 2004).

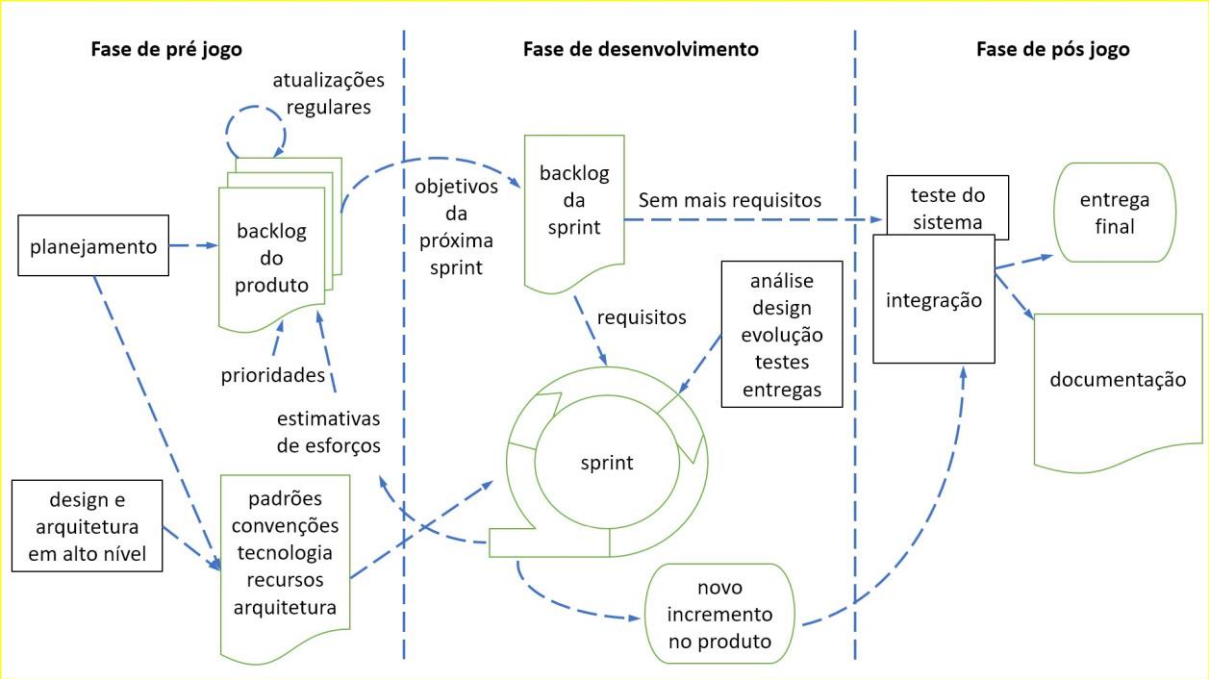
3.1.2 Eventos ou cerimônias

O Scrum possui 5 eventos principais. A *Sprint*, o *Sprint Planning*, o *Daily Scrum*, o *Sprint Review* e o *Sprint Retrospective* (Schwaber & Sutherland, 2020).

3.1.3 Processo

O processo do Scrum inclui 3 fases: “*pre-game*” (pré jogo), “*development*” (fase de desenvolvimento) e “*post-game*” (pós jogo) descritos na Figura 1.

Figura 1 Processo do Scrum



Fonte: adaptado de (Abrahamsson, Salo, Ronkainen, & Warsta, 2002)

A *sprint* é o principal evento do Scrum. Durante o processo de desenvolvimento, as sprints ocorrem uma após a outra em períodos de uma a quatro semanas. É, portanto, dentro das Sprints que os demais eventos do Scrum acontecem (Schwaber & Sutherland, 2020).

No início de cada sprint, o *Product Owner* se reúne com o time de desenvolvimento para determinar o conjunto de tarefas que serão executadas a partir da lista de estórias de usuário do backlog. Eles priorizam as estórias que serão executadas conforme entendem o que é possível realizar dentro do período da sprint e as quebram em tarefas menores documentando tudo no *Sprint Backlog* (Wangenheim, Savi, & Borgatto, 2013).

O *Sprint Backlog* é um plano feito por e para os desenvolvedores que contém a meta da sprint, o conjunto de itens do *Product Backlog* selecionados para a sprint atual e o plano de ação para a entrega (Schwaber & Sutherland, 2020).

Durante a execução da sprint, os membros do time executam reuniões diárias (*Daily Scrum*) para monitorar o progresso e controlar o projeto. Durante a reunião, cada membro responde a 3 questões: O que fez ontem, o que pretende fazer hoje e o que, eventualmente, possa estar impedindo o seu trabalho a fim de cumprir o objetivo da sprint e do projeto (Wangenheim, Savi, & Borgatto, 2013).

Ao final de cada sprint, o time demonstra os resultados para o *Product Owner* e demais interessados em uma reunião chamada *Sprint Review*. Com base nestas informações o progresso é discutido e o *Product Backlog* pode ser ajustado para atender a eventuais novas oportunidades (Schwaber & Sutherland, 2020).

A sprint é finalizada com a reunião de retrospectiva (*Sprint Retrospective*) em que os pontos fortes e fracos no que diz respeito ao processo de Scrum que foram observados durante a sprint são identificados e sugestões para melhoria contínua são propostas (Wangenheim, Savi, & Borgatto, 2013).

3.2 *Extreme programming method* (XP)

Extreme programming, ou, simplesmente, XP, tem suas origens no final dos anos 90 quando Kent Beck publica seu livro que apresentou esta nova metodologia para desenvolver softwares orientados a objetos em ambientes com frequente mudanças de requisitos (Beck & Andres, 2004). De maneira resumida, trata-se de uma integração de práticas de engenharia de software já conhecidas à época que tenta reduzir o custo das mudanças de requisitos substituindo o longo ciclo de desenvolvimento por vários pequenos ciclos para alcançar a satisfação do cliente (Abrahamsson, Salo, Ronkainen, & Warsta, 2002).

3.2.1 Papéis

Há sete papéis previsto no XP. O **programador**, que escreve o código e os testes e os mantém o mais próximo possível do que foi definido, o **cliente**, que escreve as estórias e testes funcionais, o **testador** que ajuda o cliente e executa os testes funcionais, o **rastreador** que verifica se os esforços estimados foram adequados para futuras estimativas, o **técnico** que é o responsável pela implementação e manutenção do processo XP como um todo, o **consultor** que é alguém externo para auxiliar a resolver problemas técnicos e o **gerente** que toma as decisões (Abrahamsson, Salo, Ronkainen, & Warsta, 2002).

Em que pese esta disposição de papéis, a metodologia encoraja que as pessoas procurem participar das funções que puderem contribuir da melhor forma.

Por exemplo, um gerente de projeto pode dar conselhos e proposições sobre a arquitetura de um componente, assim por diante (Beck & Andres, 2004).

3.2.2 Valores, princípios e práticas

O método XP consiste em agrupar um conjunto de valores e princípios já conhecidos no contexto de engenharia de software que levam à prescrição de um conjunto de práticas, listadas a seguir, cuja adoção deve ser feita de maneira preferencialmente gradual (Beck & Andres, 2004).

Planejamento do jogo: A equipe de desenvolvimento se reúne com o cliente e estimam os esforços de cada tarefa e, considerando o tempo das entregas, o cliente prioriza as demandas.

Pequenas e curtas entregas: Ciclos de entregas reduzidos, geralmente algumas semanas, para que o cliente consiga ver o progresso.

Metáforas: A equipe usa metáforas ou analogias para criar uma estória compartilhada com o cliente durante todo o ciclo de desenvolvimento sobre como o sistema deve funcionar.

Design simples: A solução deve ser mantida o mais simples possível e fazendo estritamente o necessário. Qualquer código extra que seja identificado deve ser imediatamente removido.

Testes de unidade: Os testes devem ser escritos antes do código de modo que eles deixem de falhar quando a implementação da funcionalidade esteja concluída.

Refatoração: Os desenvolvedores melhoram a estrutura do código (removendo duplicidades e melhorando a comunicação) regularmente para torná-lo mais fácil de manter e expandir.

Programação em pares: Duas pessoas trabalham juntas em um mesmo computador para escrever código, revisar e corrigir erros.

Propriedade do código e responsabilidade compartilhada: Qualquer pessoa pode mudar qualquer parte do código a qualquer momento.

Integração contínua: Todo novo pedaço de código é adicionado à base de maneira automatizada assim que é submetido à esteira de testes existentes e passa com sucesso por ela.

Semana de 40 horas: O Máximo de 40 horas semanais deve ser respeitado. Duas semanas seguidas com horas-extras registradas devem ser tratadas como um problema a ser resolvido.

Cliente presente: O cliente ou um representante precisa estar presente e disponível o tempo todo para atender a equipe de desenvolvimento (Abrahamsson, Salo, Ronkainen, & Warsta, 2002).

3.2.3 Processo

O ciclo de vida do projeto de desenvolvimento de software dentro do contexto da metodologia XP, conforme exibido na Figura 2, consiste nas seguintes fases: exploração de negócios, planejamento, iterações para entregas, produção, manutenção e morte (Alsaqqa, Sawalha, & Abdel-Nabi, 2020).

3.2.3.1 Fase de exploração

Com duração entre algumas semanas até poucos meses, a fase de exploração consiste em construir um protótipo simples para testar possíveis arquiteturas e tecnologias. Além disso, com envolvimento do cliente, introduzir o time ao que será construído e também aos recursos e tecnologias que serão utilizados no projeto. Nesta fase, o cliente estabelece os requisitos para a primeira entrega do sistema (Alsaqqa, Sawalha, & Abdel-Nabi, 2020), (Abrahamsson, Salo, Ronkainen, & Warsta, 2002).

3.2.3.2 Fase de planejamento

Prevista para durar cerca de dois dias, a fase de planejamento consiste estimar os esforços das estórias que serão executadas e priorizá-las para a criação da primeira pequena entrega que, em geral, deve acontecer em até 2 meses (Abrahamsson, Salo, Ronkainen, & Warsta, 2002).

3.2.3.3 Fase de iterações para entregas

Como o próprio nome já sugere, esta fase consiste em diversas etapas de pequenas entregas em que cada uma delas é feita com as práticas supracitadas de programação em pares, testes de unidade e refatoração. A primeira iteração deve entregar uma visão geral de arquitetura através da escolha de estórias apropriadas. A última iteração será a que entregará o sistema pronto para produção. Ao final de cada iteração os testes funcionais são executados garantindo estabilidade e adequação aos requisitos do cliente (Alsaqqa, Sawalha, & Abdel-Nabi, 2020).

Um ponto importante a ser ressaltado é que, apesar do método prever pequenos ciclos de entrega, caso uma estória esteja demorando demais para ser desenvolvida, o escopo da iteração é reduzido para evitar redução de qualidade. Sacrificar a qualidade não é efetivo para se controlar entregas. Reduzir a qualidade irá criar software com mais problemas e irá aumentar os prazos no futuro (Beck & Andres, 2004).

3.2.3.4 Fase de produção

Antes da entrega em produção, mais testes devem ser desenvolvidos e executados para mitigar eventuais desalinhamentos e falhas. Os testes também precisam validar a performance do sistema. Após a primeira entrega em produção, as entregas subsequentes que irão adicionar novas funcionalidades não devem impedir que o sistema continue operando (Abrahamsson, Salo, Ronkainen, & Warsta, 2002).

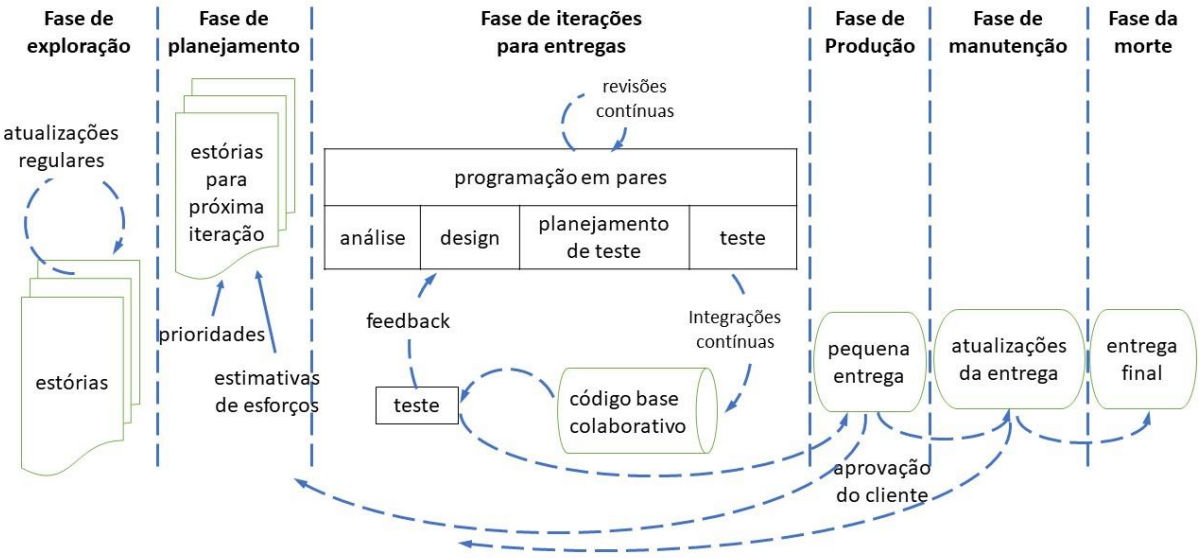
3.2.3.5 Fase de manutenção

Após a primeira entrega em produção, pode ser necessário adicionar mais pessoas à equipe para garantir a mesma taxa de entrega dado que, a partir deste momento, novas demandas de manutenção podem surgir além da previsão de uma nova demanda ligada ao atendimento de suporte ao cliente (Alsaqqa, Sawalha, & Abdel-Nabi, 2020).

3.2.3.6 Fase de morte

Ocorre quando o cliente não tem mais estórias para apresentar, a documentação pode ser finalizada e não há mais mudanças arquiteturais, de design ou mesmo em código. Este estágio também é previsto se o sistema não estiver mais entregando o que se espera, se estiver obsoleto ou se os incrementos em desenvolvimento estão muito custosos (Abrahamsson, Salo, Ronkainen, & Warsta, 2002).

Figura 2: Ciclo de vida do XP



Fonte: adaptado de (Abrahamsson, Salo, Ronkainen, & Warsta, 2002)

4 Kanban – estado de arte

O método Kanban, devido às suas características, pode ser aplicado a quaisquer processos que envolvam cadeia de valor. Assim, seu uso tem sido observado em diferentes indústrias como: Aeronáutica, Saúde, Têxtil, Recursos Humanos e Software (Ahmad, Dennehy, Conboy, & Oivo, 2018).

Apesar de sua larga aplicabilidade, os estudos selecionados para consulta são aqueles em que o Kanban é aplicado ao desenvolvimento de software.

4.1 Evolução das pesquisas sobre o método Kanban

Em 2013, uma revisão de literatura feita sobre o método Kanban (Tabela 1) no desenvolvimento de software mostrou que as publicações sobre o tema estavam crescendo. Os estudos se classificavam em: relatórios de experiências (47%), estudos qualitativos (32%), estudos de simulação (10%) e estudos quantitativos (11%). Ou seja, as publicações sobre Kanban, em sua maioria, são resultados de estudos empíricos. Estes estudos foram filtrados de um total de 1828 artigos encontrados com a combinação da palavra Kanban com junto às expressões *software engineering; software development; challenges; software industry* e *software design*, nas fontes ACM Digital Library; ABI/Inform (ProQuest); Science Direct (Elsevier); Springer Link (LNCS); Web of Science (ISI) e IEEE Xplore – IEEE/IEEE Eletronic Library, após uma sequência de três filtragens sendo a primeira pelo título, que filtrou 492 artigos, a segunda pelo título, resumo e palavras-chaves, que filtrou 79 e, a terceira pela relevância dos trabalhos, resultados encontrados, justificativas e aplicabilidade (Ahmad, Markkula, & Oivo, 2013).

Tabela 1 - Principais publicações anuais

Ano	2000-07	2008	2009	2010	2011	Total
Artigos	-	1	2	6	10	19
Porcentagem	-	5%	10%	32%	53%	100%

Fonte: (Ahmad, Markkula, & Oivo, 2013)

Em 2018, um novo estudo exploratório sobre as pesquisas em Kanban (Figura 3) identificou 23 estudos primários distribuídos na seguinte forma:

Figura 3 - Estudos sobre Kanban publicados por ano

Fonte: (Ahmad, Dennehy, Conboy, & Oivo, 2018).

Nesses novos resultados, estão listados apenas os principais artigos que foram classificados como foco de seu estudo. Tais artigos evidenciam os benefícios do Kanban em diversos aspectos. Dos 23 artigos, 8 foram publicados em Periódicos e 15 em conferências. Desses, (Ahmad, Dennehy, Conboy, & Oivo, 2018) extraem algumas conclusões acerca dos desafios e benefícios do método.

4.2 Desafios apontados para implementação do método Kanban

Alguns estudos evidenciaram dificuldades na implementação do Kanban, dentre as quais (Ahmad, Dennehy, Conboy, & Oivo, 2018) salientam:

a) Configuração e manutenção do Kanban;
b) Gestores não preparados para um novo método;
c) Pobre entendimento dos conceitos e práticas do Kanban;
d) Gestão da comunicação entre o time e o cliente;
e) Mudança da cultura organizacional;
f) Falta de suporte às práticas que envolvem o Kanban;
g) Falta de treinamento;
h) Pobre conhecimento da gestão.

Os itens ***a, b, c*** e ***h*** podem ser sanados com ministração de treinamentos e indicam falta de planejamento no processo de implantação, pois o treinamento é base fundamental de qualquer mudança.

A escassez de treinamento (item ***g***) é decorrente do estágio do método que ainda está em processo de amadurecimento. A *Lean Kanban University* possui uma agenda regular de treinamentos que contam com empresas certificadas em vários países. Conforme o método amadurece e novas pessoas são treinadas, este problema tende a se reduzir e, consequentemente, os problemas ***a, b, c*** e ***h***. O uso de jogos também é útil à aprendizagem do método, porém, ainda não há evidências de que esta abordagem é melhor em relação a métodos tradicionais de ensino (Heikkilä, Paasivaara, & Lassenius, 2016).

O item *d* é o maior desafio do Kanban e, conforme exposto na descrição do método, o principal fator de sucesso é a comunicação e o alinhamento com o cliente. Na proposta de (Anderson, 2010), há maneiras de se tratar com o cliente o modo de trabalho. Se o cliente não concordar, não há como se conseguir uma implantação totalmente bem-sucedida. Este é um tipo de problema que precisa ser mitigado por meio de muitas conversas, reuniões e experimentações junto aos clientes.

O item *e* também é um fator importante, entretanto, conforme descrito, o Kanban não impõe muitas mudanças no processo de trabalho do dia a dia, se comparado, por exemplo, ao Scrum. O intuito de começar a mapear o fluxo de trabalho pode dar uma errônea ideia de que se pretende identificar “culpados”. Este, porém, não é o objetivo do método, ponto que deve ser muito bem esclarecido para se conseguir o alinhamento da equipe. Equipes ineficientes também podem ter problemas com o método, uma vez que as ineficiências são expostas, assim como todos os gargalos.

4.3 Benefícios do método Kanban

Os benefícios do método Kanban são muitos. Conforme mostrado na Tabela 2 (Ahmad, Dennehy, Conboy, & Oivo, 2018) enumeram-nos da seguinte maneira, inclusive citando o número de artigos que abordam cada item, considerando que cada artigo pode mencionar mais de um benefício:

Tabela 2 - Benefícios do método Kanban

Benefício apontado	Número de artigos
a. Melhoria da visibilidade e da transparência	16 artigos
b. Melhor controle das atividades dos projetos	12 artigos
c. Identificação de impedimentos no fluxo	10 artigos
d. Melhoria no processo de trabalho	7 artigos
e. Redução do tempo de entrega	5 artigos
f. Melhoria na priorização de tarefas	4 artigos
g. Redução de defeitos e bugs	4 artigos
h. Aumento da qualidade	4 artigos
i. Método leve e intuitivo	4 artigos
j. Melhoria na colaboração e comunicação	7 artigos
k. Melhoria na motivação do time	6 artigos
l. Coesão e formação do time	5 artigos
m. Aumento da satisfação do cliente	6 artigos
n. Cultura de aprendizado contínuo	5 artigos
o. Alinhamento estratégico	3 artigos

Fonte: Elaborado pelo autor

A redução de defeitos e bugs está intimamente ligada à melhoria de qualidade. Então, por mais específicos que sejam os feedbacks, é possível unificar os itens *g* e *h* em apenas um item sobre Melhoria da Qualidade, com citações em seis artigos, dado que Middleton é citado nos dois itens (Ahmad, Liukkunen, & Markkula, 2014); (Al-Baik & Miler, 2015); (Ikonen, Pirinen, Fagerholm, Kettunen, & Abrahamsson, 2011); (Mahnič, 2015); (Middleton & Joyce, 2012); (Sjøberg, Johnsen, & Solberg, 2012). Todos estes trabalhos apontaram que o Kanban permitiu, de certa forma, promover melhoria na qualidade.

Os itens de ***a*** a ***d***, ***f, i*** a ***l, n*** e ***o*** são os benefícios do método que levam ao resultado do aumento da qualidade (itens ***g*** e ***h***) citados nos trabalhos acima. Os itens ***e, g*** e ***h*** levam ao ***m*** que é a percepção do trabalho pelo maior interessado. Estas induções ficam mais evidentes com o entendimento do método e a verificação dos demais estudos de casos.

Em indústrias de manufatura, em que o Kanban tem sido aplicado há várias décadas (Ikonen, Pirinen, Fagerholm, Kettunen, & Abrahamsson, 2011), tem se

observado uma decorrente facilitação da aplicação dos princípios do Lean, de forma simples e efetiva. A combinação da simplicidade do Kanban com as capacidades tecnológicas de um quadro digital facilita a automação de processos, melhora visibilidade e permite a medição de performance em tempo real (Wan & Chen, 2008).

4.4 Estudos experimentais sobre o método

Ainda na mesma pesquisa, (Ahmad, Dennehy, Conboy, & Oivo, 2018) enumeram 23 relatórios de estudos de casos envolvendo o uso do Kanban.

4.4.1 Desafios do Kanban segundo os estudos de casos

Os estudos experimentais apontaram os seguintes desafios na utilização do método Kanban:

a) Falta de boas-práticas (*guidelines*) e entendimento do método e sua implementação;
b) Avaliar o desempenho usando métricas;
c) Motivar o time para adotar novas práticas;
d) Troca de tarefas e fluxo de trabalho imprevisível;
e) Kanban requer integração com técnicas ágeis existentes, o que pode ser complicado, caro e demorado;
f) Mudança da cultura da organização;
g) Falta de habilidades especializadas e treinamento;
h) Implementação do Kanban requer profundo entendimento do Lean.

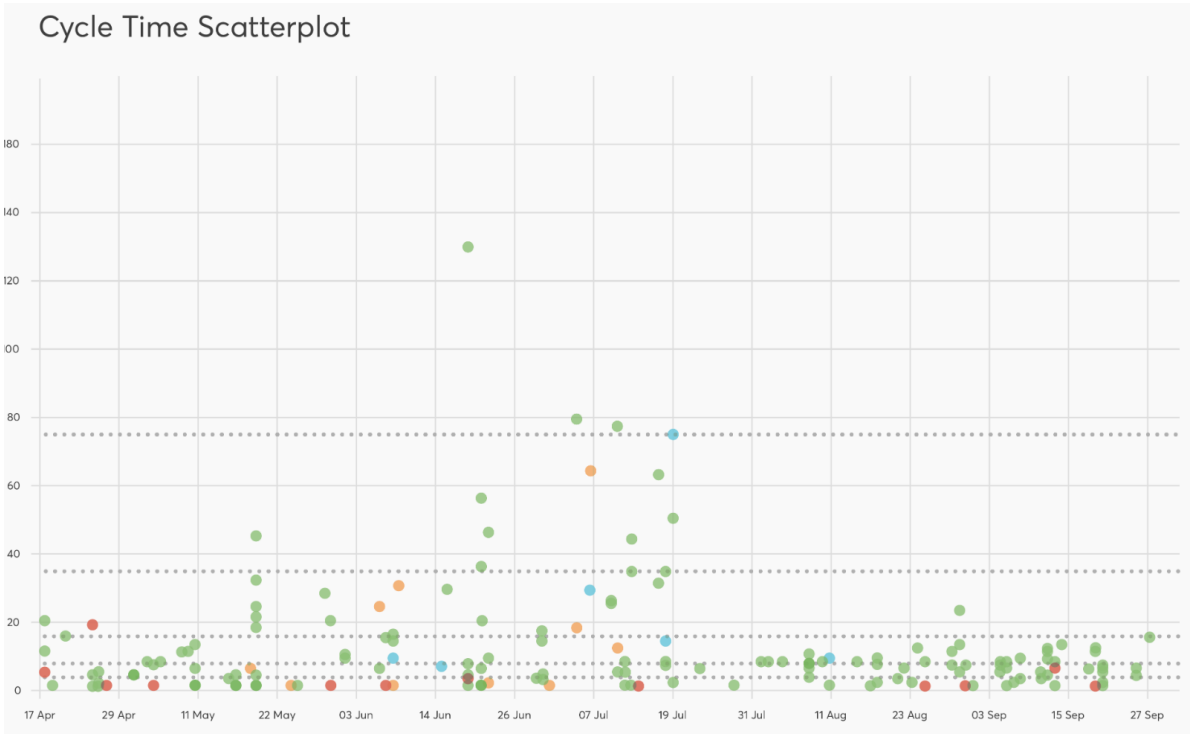
Os itens ***a***, ***b***, ***e***, ***g*** e ***h*** podem ser tratados com treinamento apropriado. O item ***h*** pode ser considerado como uma derivação do item ***g***, dado que treinamentos são, de fato, necessários. Contudo, os princípios do Lean, são melhor aprendidos através de sua prática e o Kanban provou ser um método eficaz para ensinar estes princípios (Shinkle, 2009). O dia a dia da utilização do Kanban não requer nenhum conhecimento complexo para ser adquirido pelos desenvolvedores. Contudo, o gestor precisará estudar alguns conceitos, caso os desconheça, como, por exemplo, extração e análise

de métricas que irão auxiliá-lo nas tomadas de decisões. É neste ponto que se observa um dos grandes benefícios do método (Hammarberg & Sundén, 2014).

A previsibilidade, apontada no item ***d*** como desafio, em alguns estudos, tende a ser decorrente da falta de treinamento do gestor, pois as métricas de algumas semanas de utilização podem, efetivamente, mostrar como é a taxa de entrega da equipe e, assim, tornar as entregas previsíveis. Figura 4 mostra um exemplo de como uma métrica bem estruturada permite dar previsibilidade a um projeto:

O eixo vertical representa o número de horas que uma tarefa demorou para ser concluída. No eixo horizontal, a data de início da atividade. Quanto mais alto o ponto no gráfico, mais tempo a tarefa demorou para ser entregue. De imediato, pode-se observar que nenhuma tarefa demorou mais que 140 horas e que a maioria demorou até 30 horas aproximadamente. De modo mais preciso, pode-se traçar retas horizontais com os percentis de tempo de conclusão. Estes percentis mostram o histórico de entregas. Ao traçarmos uma linha horizontal de modo que 95% dos pontos estejam abaixo desta linha, determina-se o percentil 95%. Ou seja, o tempo em que 95% das tarefas foram concluídas.

Figura 4 - Exemplo de métrica*: Cycle Time Scatterplot*



Fonte: (Siderovza, 2018)

A partir da análise do gráfico é possível prever, com 95% de confiança, que é possível entregar eventuais tarefas futuras similares dentro de até 80 horas. Este tipo de estimativa só é possível quando as mesmas condições de projeto continuem vigorando: uma alteração na quantidade de membros da equipe, por exemplo, afetaria os resultados futuros e invalidaria as previsões.

Os itens ***c*** e ***f*** fazem parte dos desafios de qualquer mudança e de implementação de qualquer tipo de metodologia, seja Scrum, *Extreme Programming* (XP) ou, até mesmo, *Rational Unified Process* (RUP). Destas citadas, o Kanban é a que menos impacta no dia a dia dos desenvolvedores pois, devido ao método de implementação (*STATIK*, mostrado adiante) é possível iniciar com apenas uma mudança, ou seja, a atualização diária de um quadro de atividades em uma reunião de 15 minutos. Não é necessário criar escopos, fazer reuniões de replanejamento, programação em pares, nem nenhuma outra rotina diferente do que já esteja adotado na organização.

4.4.2 Benefícios do Kanban segundo os estudos de casos

Os mesmos estudos, elencam os seguintes benefícios:

a) Facilita a visibilidade e o suporte ao processo de tomada de decisão;
b) Desenvolve estratégias de melhorias contínuas e aprimora o fluxo de trabalho;
c) Promove melhor entendimento do processo de desenvolvimento como um todo;
d) Melhora a previsibilidade da entrega do produto e permite uma melhor estimativa do trabalho;
e) Redução do tempo dos ciclos de trabalho e dos tempos de cada tarefa (*lead time*);
f) Melhora o balanceamento do trabalho;
g) Garantia de melhoria das habilidades e a coesão dos times;
h) Facilita a coordenação impondo uma auto-organização;
i) Dirige e facilita a gestão da mudança da organização.

É notável o alinhamento dos resultados entre os trabalhos primários selecionados e os estudos experimentais. Pode-se destacar um dos estudos em que (Santos, Beltrão, de Souza, & Travassos, 2018) apontam que o Kanban tem como principais benefícios, melhorar a visibilidade do trabalho, o controle das atividades e tarefas do projeto, o fluxo de trabalho e o tempo de entrega (*time-to-market*). Como principal desafio do método, o Kanban também visa melhorar a cultura da empresa.

Não há muitas discrepâncias nas implementações. A falta de treinamento adequado da gestão e do time fica evidente quando não há sucesso, fator agravado pela escassez de profissionais no mercado. Trabalhos como o presente podem auxiliar a mitigar um pouco este problema.

4.5 Principais conclusões sobre o método Kanban

Com origem no conceito de gestão de manufatura do Lean, o Kanban tem obtido grande popularidade como método de desenvolvimento de software (Nevenka & Saso, 2015). Cada projeto tem sua particularidade e todos os métodos ágeis possuem vantagens e desvantagens. O que distingue o Kanban, entretanto, das demais metodologias ágeis são as poucas mudanças necessárias a serem aplicadas no processo de desenvolvimento para sua implantação. O método também oferece métricas para monitoramento e estudos com base em termos estocásticos. Prever o processo de desenvolvimento aumenta a eficiência e a colaboração dentro do time (Nevenka & Saso, 2015).

O Kanban também ajuda a identificar e a deixar claro quais são as tarefas de maior prioridade, o que permite que os gestores decidam o que deve ser feito e qual a alocação de recursos adequada (Ahmad, Kuvaja, Oivo, & Markkula, 2016). Melhora a visibilidade do trabalho e o fluxo de desenvolvimento (Ahmad, Markkula, & Oivo, 2016), (Willeke, 2009) e reduz o estresse mental por meio da diminuição do número de tarefas em execução ao mesmo tempo (*work in progress – WIP,* ou limitação do trabalho em progresso). Com a redução do WIP, os times se concentram em poucas tarefas que podem ser resolvidas mais facilmente (Ahmad, Markkula, & Oivo, 2016), (Al-Baik & Miler, 2015), (Hiranabe, 2008), (Ahmad, Liukkunen, & Markkula, 2014), (Ahmad, Markkula, & Oivo, 2013), (Anderson, 2010).

Apesar das vantagens e benefícios reportados pelas pesquisas feitas, há discordâncias e descrenças na inclusão deste novo método no processo de gestão diário. O envolvimento do Kanban no processo de desenvolvimento é novo e ainda há necessidade de entendimento mais profundo e mudanças no tradicionalismo para uma adoção mais rápida (Nevenka & Saso, 2015).

Os pontos de atenção que se pode elencar são a falta de treinamento apropriado, o desafio da mudança do *mindset* da gestão e da cultura da organização, que também são desafios compartilhados por outras metodologias ágeis, além da dificuldade de se implementar o Kanban em certos cenários em que há falta de tempo ou treinamentos adequados e trabalhos remotos (Ahmad, Markkula, & Oivo, 2016), (Al-Baik & Miler, 2015), (Ahmad, Liukkunen, & Markkula, 2014), (Ahmad, Markkula, & Oivo, 2013), (Rodríguez, Markkula, Oivo, & Turula, 2012). Apesar de todos os desafios mencionados, o Kanban tem sido aplicado com sucesso em diversas empresas de software (Seikola, Loisa, & Jagos, 2011), (Rutherford, Shannon, Judson, & Kidd, 2010), (Taipale, 2010) (Senapathi & Drury-Grogan, 2021), (Ahmad, Kuvaja, Oivo, & Markkula, 2016).

4.5.1 Manutenção de software

O Kanban oferece diversos benefícios para o trabalho de manutenção de software, tais como trazer visibilidade e esses tipos de tarefas, proteger o time em relação ao excesso de carga de trabalho, ajudar na priorização, facilitar sincronia de trabalho com outros times, favorecer a troca de pessoas entre times, melhorar a comunicação e promover a colaboração e o trabalho em equipe. Usando o Kanban, as ações do time endereçadas às tarefas de maior prioridade são espontâneas e a limitação do WIP (carga de trabalho) aumenta a cadência de entrega e a eficiência. (Ahmad, Kuvaja, Oivo, & Markkula, 2016).

5 O método Kanban

Para explicar o método Kanban, deve-se levar em consideração a sua origem, bem como o contexto de sua criação, para que se possa defini-lo e descrever como aplicá-lo em cada contexto.

5.1 Origem

A palavra *kanban* é japonesa e significa "*Signboard",* ou seja, cartão ou quadro contendo um texto de identificação. Já Kanban, com inicial maiúscula, refere-se à definição do método conforme proposta por (Anderson, 2010) que aplica os conceitos do Lean, criado por Taiichi Ohno, responsável pelo Sistema Toyota de Produção (TPS). Este sistema é baseado em dois conceitos, a saber *automação e produção just-in-time (JIT),* cujo fundamento se encontra em obter (ou prover) os recursos necessários exatamente quando eles são demandados. Nesse sistema, (Ohno, 1988), citado por (Anderson, 2010), utiliza o kanban como ferramenta para seu método que revolucionou os meios de produção.

Após a publicação do livro "*The Machine that Changed the World*" (Womack, Jones, & ROOS, 1990), o termo *Lean* é cunhado para designar o pensamento deste novo sistema de produção, que (Wang, Conboy, & Cawley, 2012), bem como (Ahmad, Dennehy, Conboy, & Oivo, 2018) sinalizam ter como conceitos:

a) Valor: Valor conforme definido pelo cliente;
b) Cadeia de valor: O mapa que identifica cada etapa no processo e as categoriza conforme o valor que ela adiciona;
c) Fluxo: Refere-se ao fluxo contínuo da cadeia de valor no processo;
d) Puxar: Pedidos dos clientes puxam demandas por novos produtos, garantindo que nada seja feito antes de ser necessário ou solicitado;
e) Perfeição: Buscar a melhoria dos processos, removendo desperdícios continuamente.

(Poppendieck & Poppendieck, 2003) fez a primeira publicação utilizando os conceitos Lean no desenvolvimento de software. Novas metodologias de

desenvolvimento, como XP e Scrum, surgiram com o manifesto ágil. (Anderson, 2010) começa a divulgar o termo Kanban para designar este conjunto de técnicas derivadas do Lean, a partir da publicação de seu livro. Embora existam muitas derivações do método Kanban e detalhes de implementação que são aplicados de forma particular a cada projeto, grupo ou instituição, o método é descrito usando as considerações de Anderson, assim como a proposta apresentada como uma evolução do método.

5.2 Definição

Kanban, com inicial maiúscula, é um método evolucionário de acompanhamento de projetos, que se vale de um kanban, de inicial minúscula, e outras ferramentas para visualizar um sistema puxado para aplicar ideias do Lean ao processo de desenvolvimento tecnológico, bem como em operações de tecnologia da informação (Anderson, 2010).

Portanto, o kanban iniciado em minúscula é um quadro com cartões que representam as tarefas em um dado fluxo de valor e que se caracteriza como um sistema puxado, ou seja, um sistema por onde uma nova atividade só pode ser iniciada se outra tiver sido previamente concluída. A conclusão de uma tarefa permite puxar uma nova, definindo assim este tipo de sistema. Não se pode, portanto, empurrar uma tarefa, de modo que é possível limitar o trabalho em progresso em qualquer etapa. Esta definição foi aceita, também por outros autores, como (Ahmad, Markkula, & Oivo, 2013), (Dennehy & Conboy, 2016), (Fitzgerald, Musiał, & Stol, 2014), (Harzl, 2016), (Law & Lárusdóttir, 2015), (Mahnič, 2015), (Tripathi, Rodríguez, Ahmad, & Oivo, 2015), (Rodríguez, Partanen, Kuvaja, & Oivo, 2014) e por (Senapathi, Middleton, & Evans, 2011).

5.3 Implementação

O método Kanban possui algumas propostas de implementação. A principal, chamada *STATIK*, utiliza uma abordagem sistêmica e consiste em uma sequência de passos a serem aplicados de forma iterativa (Anderson & Carmichael, 2016). Outra abordagem que é o uso de jogos, também é muito comum, porém, ainda não se dispõe

de evidências de que esta abordagem seja eficaz (Heikkilä, Paasivaara, & Lassenius, 2016).

5.3.1 STATIK

STATIK é um acrônimo para *System Thinking Approach to Introducing Kanban* ou, *abordagem de pensamento sistêmico para introdução do Kanban*. É uma abordagem colaborativa para projetar um sistema kanban baseado no processo atual de entrega de um produto ou serviço apresentados em uma sequência de passos (Senapathi & Drury-Grogan, 2021). É recomendável começar a implantação com um pequeno time, avaliar os resultados gradualmente e, a partir daí, expandir os conceitos e mudanças para os demais times progressivamente. (Maassen & Sonnevelt, 2010). Os primeiros serviços a serem tratados com o STATIK devem ser aqueles que operam em um nível mais alto e que entregam diretamente aos clientes, em detrimento de serviços internos que entregam dentro da organização (Anderson & Carmichael, 2016).

Há pequenas divergências entre os passos na bibliografia, mas não são significantes ao ponto de serem contraditórias. Apresenta-se um resumo dos passos sob a perspectiva de organização apresentada por (Burrows, 2014).

5.3.1.1 Entender as fontes de insatisfações

Há dois pontos de vista para se entender as insatisfações: Internas e externas.

As insatisfações externas são coletadas por agentes de fora do time, ou seja, stakeholders como clientes ou quem recebe as entregas (Muniz, Irigoyen, Mafra, Trierveiler, & Villanova, 2021).

O processo de entender as fontes de insatisfação internas pode envolver a realização de entrevistas com funcionários, a análise de dados do processo de trabalho e a observação direta dos colaboradores. As entrevistas podem ser individuais ou em grupo com a aplicação de técnicas de *brainstorming* e votação. Nesta etapa, deve-se evitar assumir possíveis soluções para os tópicos elicitados.

As fontes de insatisfação podem incluir problemas como atrasos, falta de visibilidade do trabalho, falta de transparência, excesso de trabalho em andamento e problemas de qualidade (Burrows, 2014).

5.3.1.2 Análise das demandas e capacidades

Ter uma demanda maior do que se consegue atender é o tipo de problema bom. O oposto, seria ter que reduzir a força de trabalho. Esta disparidade entre demanda e possibilidades de implementação precisa estar explicitamente clara para não gerar a impressão de que os recursos são ilimitados (Leopold, 2018).

A análise das demandas e capacidades trata-se da coleta de fatos qualitativos e quantitativos específicos sobre o processo em vigor.

Qualitativamente, compreender os diferentes tipos de trabalho envolvidos ajuda a identificar as diferentes variações no fluxo de trabalho que precisam ser gerenciadas. Compreender as diferentes fontes de demanda ajuda a se preparar, moldar e gerenciar o trabalho em progresso. Compreender por que o trabalho é necessário ajuda a entender os tipos de riscos envolvidos, para que possam ser gerenciados adequadamente.

Quantitativamente, compreender a quantidade de trabalho envolvida ajudará a escolher uma granularidade gerenciável para visualizá-la e controlá-la.

Compreender a lacuna entre a capacidade real do processo (medida em termos de taxas de entrega, tempos de espera, previsibilidade etc.) e as expectativas dos clientes e da organização em geral destacará que tipo de melhorias são necessárias.

Alguns diagramas que podem ajudar a visualizar estes dados são: *diagrama de lead time*, *throughput* e *cumulative flow diagram* (Burrows, 2014).

5.3.1.3 Modelagem do fluxo de trabalho

A modelagem do fluxo deve focar em descrever o sistema atual evitando fazer julgamentos ou modificações precipitadas (Muniz, Irigoyen, Mafra, Trierveiler, & Villanova, 2021).

Três abordagens podem ser utilizadas na modelagem do fluxo de trabalho: Esboço; Decomposição de cima pra baixo e Organização de baixo para cima.

Independentemente da abordagem escolhida, o objetivo deste passo é criar um mapa contendo as principais atividades que representam a cadeia de valor. Entre algumas destas atividades, podem existir filas, que devem ser explicitamente identificadas. É interessante identificar também, o ponto de comprometimento, ou seja, a partir de qual etapa uma determinada tarefa é considerada como uma tarefa em andamento e não apenas como um item de backlog (Burrows, 2014).

5.3.1.4 Descrever as classes de serviço

As classes de serviço servem para guiar como cada tipo de demanda será tratada. (Muniz, Irigoyen, Mafra, Trierveiler, & Villanova, 2021) Se todas as demandas atuais são tratadas da mesma forma, como por exemplo, seguindo a lógica FIFO (*First in first out*, primeira a chegar é a primeira a ser iniciada, em tradução livre e não-literal) não há classes de serviço diferenciadas e todas as demandas são comuns ou (*Standard*). Porém, se algumas demandas devem receber prioridade quando chegam, então uma notação para estas demandas precisa estar explícita no fluxo. Alguns exemplos de classes de serviço, a saber: expressa (*Expedite*), compreende os *bugs* emergenciais e as demandas regulatórias, também conhecida como *Urgente* ou *Prioridade Zero*; padrão (*Standard*), que compreende as tarefas de novas funcionalidades e os *bugs;* datas-fixas (Fixed-date), tarefas cuja conclusão deve ocorrer em data específica. Geralmente são alterações que envolvem o cumprimento de leis ou demandas de órgãos regulatórios (Burrows, 2014).

5.3.1.5 Desenho do quadro Kanban

O desenho do quadro é a etapa que caracteriza a aplicação do método e consiste em compilar todos os registros levantados nas etapas anteriores, estruturando-os em um quadro que seja posicionado em local de fácil visibilidade e acessibilidade para os desenvolvedores.

- **Desenho da estrutura do quadro (*frame*)**

Nas linhas deve-se aplicar as classes de serviço. Como exemplo: Expedite e Standard conforme mostra a Figura **5**.

Figura 5 - Classes de Serviço

Expedite
Standard

Fonte: Elaborado pelo autor

Nas colunas, deve-se aplicar as etapas da cadeia de valor mapeadas anteriormente, como, por exemplo, Backlog, To Do, Doing, Ready to Test, Testing, Done e outras, Figura **6**.

Figura 6 - Etapas da cadeia de valor

Projeto A	Backlog	To Do	Doing	Ready to Test	Testing	Done
Expedite						
Standard						

Fonte: Adaptado de (Anderson, 2010)

a) *Backlog*: Conjunto de todas as ideias e vontades dos clientes e dos desenvolvedores empacotadas em tarefas, que devem ser tanto menores quanto possível, mas de modo que tragam algum valor ao cliente, ao time de desenvolvimento ou a outro interessado (*stakeholder*).
b) *TO DO*: Fila das tarefas extraídas do backlog para serem desenvolvidas. Estes itens já estão comprometidos para serem entregues.

c) *Doing*: Tarefas que estão sendo trabalhadas. É comum adicionar um cartão extra com a identificação do desenvolvedor que está cuidando de cada tarefa nesta etapa, assim como na etapa de teste e em todas as outras que não representam uma fila, mas sim, uma transformação no processo.
d) *Ready to Test:* Tarefas finalizadas pela equipe de desenvolvimento e estão prontas para serem testadas.
e) *Testing*: Tarefas sendo testadas.
f) *Done:* Tarefas concluídas pela equipe de desenvolvimento.

Aqui, foram utilizadas apenas colunas de exemplo. Entretanto, outras colunas poderiam ser adicionadas de acordo com o processo real mapeado. Por exemplo, pode-se inserir colunas para homologação, fila para *deploy* em ambiente, empacotamento de produto, entre outras, adequadas à realidade de cada projeto. Para fins didáticos, o processo neste quadro de exemplo é finalizado em *Done*.

- **Definição dos cartões**

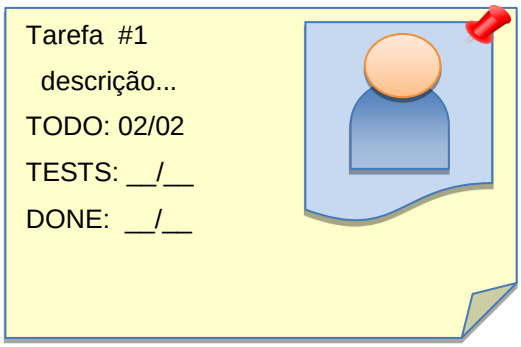
A estrutura dos cartões merece um detalhamento pois são eles que mostram os itens de valor percorrendo a cadeia. As informações devem facilitar a autonomia da equipe nas decisões para puxá-los. Dependendo da classe de serviço, a informação no cartão pode variar. Por exemplo, itens com data fixa, devem conter a data limite para finalização. (Anderson, 2010)

Os cartões devem conter, prioritariamente, as seguintes informações:

a) Identificação (número ou id);
b) Descrição breve da tarefa;
c) Data de início;
d) Data de fim (preenchida quando concluído).

Conforme se avança na utilização do método e se julgue necessário, é possível adicionar outras informações como datas intermediárias para se extrair medições mais precisas do avanço das atividades no fluxo, Figura 7.

Figura 7 - Representação de uma atividade do fluxo



Fonte: Adaptado de (Anderson, 2010)

- **Definição das políticas**

Uma das políticas importantes do Kanban é a reunião diária para o acompanhamento de atividades (*Daily Standup meetings*). As reuniões diárias feitas de pé, preferencialmente, ocorrem geralmente pela manhã antes do trabalho começar. Elas envolvem uma equipe de até doze pessoas, e cada membro responde a três perguntas sobre seu progresso e eventuais bloqueios: O que foi feito ontem? O que será feito hoje? E, se há algum impedimento. Com o uso do sistema Kanban, as reuniões assumem um formato diferente, focando no fluxo de trabalho. Equipes mais avançadas podem se concentrar apenas em tarefas bloqueadas ou com defeitos. A organização pode usar estratégias de visualização para melhor gerenciar questões e permitir que os membros solicitem ajuda quando necessário. Nesta reunião, o quadro é observado da direita para a esquerda. A motivação desta orientação é baseada no fato de que as tarefas que estão mais à direita, foram, eventualmente, iniciadas primeiro e, além disso, estão mais próximas de se tornar valor para o cliente. Portanto, precisam ser concluídas e priorizadas em relação a quaisquer outras que estejam mais à esquerda dentro da mesma classe de serviço (Anderson, 2010).

Também são políticas importantes: as tarefas das classes de serviço superiores devem ser executadas com prioridade em relação às tarefas das classes de serviço inferiores; uma nova atividade só pode ser iniciada se houver espaço para ela, ou seja, se o WIP não estiver ultrapassado; (Burrows, 2014) é necessário sempre terminar algo antes de começar algo novo. O sistema deve ser puxado (*pulled-System*) e, por fim, a etapa de *TODO* é a etapa de comprometimento: a equipe se compromete a entregar o mais rápido possível e o cliente promete não mais interferir

naquela atividade. Estes foram alguns exemplos de políticas comumente utilizadas no método Kanban (Anderson, 2010).

- **Definição dos limites de trabalho (WIP)**

O Kanban é um processo empírico que enfatiza pequenas mudanças respaldadas por dados empíricos. Para determinar os limites de trabalho em progresso (WIP) mais adequados para um contexto específico, é importante rastrear métricas como *lead time*, *throughput* e *cycle time*. A Lei de Little destaca a importância da relação entre WIP, Taxa de Entrega e *lead time*. Embora possa haver incertezas quanto à definição dos limites iniciais de WIP, essas discussões devem ser vistas como um mecanismo saudável que possibilita o surgimento de uma cultura de melhoria contínua (Senapathi & Drury-Grogan, 2021).

A Figura 8 mostra um exemplo de deixar explícita a política de limitação de WIP

Figura 8 – Exemplo de limitação do WIP

Projeto A	Backlog	TO DO	Doing	Ready to Test	Testing	Done
WIP Máximo		3	4	3	3	
Expedite						
Standard						

Fonte: Adaptado de (Anderson, 2010)

Estes números são arbitrários e devem ser revistos de tempos em tempos, a cada medição de *throughput,* de modo a otimizá-lo (Anderson, 2010).

Limitar o WIP, comprovadamente, reduz o tempo de execução das tarefas (Maassen & Sonnevelt, 2010).

- **Acordo de entrega (cadência)**

Algumas metodologias ágeis, como o Scrum, propõem um acordo de entrega baseado em faixas de tempo, chamadas *sprints*: a cada mês, a cada 15 dias ou a cada semana, por exemplo. Esta abordagem traz alguns inconvenientes como a necessidade de se planejar um conjunto de tarefas que se encaixem no período de desenvolvimento; a sobrecarga do time em cumprir todas as atividades para que não haja atraso em relação ao planejamento, ou ainda; a dificuldade em se modificar o escopo do que está sendo desenvolvido durante a sprint (Greening, 2010).

Como uma taxa de entregas regulares geram mais confiança, os times têm adaptado seus ciclos de entregas para intervalos menores para mitigar variabilidades. Contudo, esta prática levou a necessidade de se quebrar demais as estórias e este processo tem se mostrado bastante penoso e ineficiente (Anderson, 2010).

O Kanban não prevê um período de entrega baseado em tempo, mas sim, prevê que as principais atividades e funcionalidades priorizadas pelo cliente sejam entregues o quanto antes, no menor tempo possível, o que tem se mostrado totalmente viável (Birkeland, 2010).

Quando o cliente não está acostumado com o método Kanban, é normal que fique desconfortável se não lhe for apresentado um prazo para a entrega do produto. Inclusive, dentro da própria empresa, pode haver conflitos com a falta de uma definição forte de data de entrega, entre, por exemplo, as equipes de marketing e vendas (Neely & Stolt, 2013).

5.3.1.6 Início da implantação

Todos os passos anteriores devem ser alinhados com time e as demandas atuais devem ser colocadas no quadro de acordo com a etapa em que cada uma se encontra (Muniz, Irigoyen, Mafra, Trierveiler, & Villanova, 2021).

Na Figura 9, pode-se observar como as tarefas ficam dispostas, em um determinado momento, no fluxo de trabalho do dia a dia de um projeto em andamento.

Neste fluxo, o WIP máximo está sendo respeitado em todas as etapas e, portanto, há possibilidade de se iniciar uma nova tarefa ou atender uma demanda urgente que apareça.

Com o WIP limitado, o time consegue entender se pode ou não puxar uma nova tarefa, além de se sentir encorajado para aplicar a técnica do *Swarming* (quando vários membros do time param suas tarefas ao mesmo tempo e focam em ajudar alguém que está com dificuldades) para resolver problemas. As classes de serviço e o backlog priorizado permite que o time decida qual o próximo passo sem atuação dos gestores (Anderson, 2010).

Figura 9 - Etapas de desenvolvimento conforme o quadro Kanban

Projeto A	Backlog	TO DO	Doing	Ready to test	Testing	Done
WIP Máximo		3	4	3	3	
Expedite			# 8			
Standard	#11 #10 #12	# 9	# 7 # 6	# 5	# 4	# 3 # 2 # 1

Fonte: Adaptado de (Anderson, 2010)

Com o Kanban implementado, dentro de algumas semanas as métricas já poderão fornecer valiosas informações para melhorias incrementais conforme detalhado a seguir.

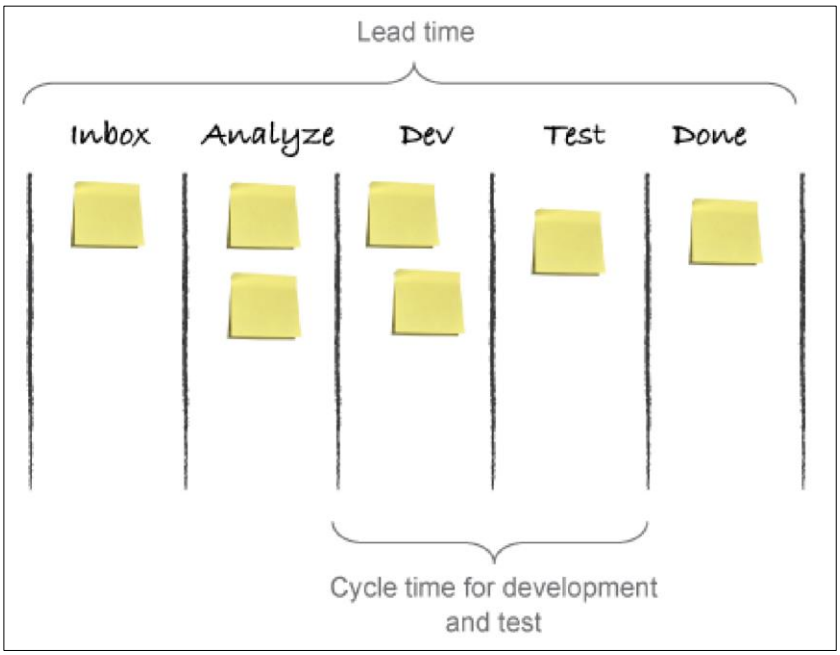
5.3.2 Métricas

Após o início do projeto, em aproximadamente duas semanas já é possível visualizar algumas métricas, que possibilitam o acompanhamento do projeto, identificar se as decisões tomadas foram assertivas e ajudar a planejar as próximas.

Neste contexto, (Hammarberg & Sundén, 2014) apontam que, entre as principais métricas, pode-se elencar:

a) *Cycle* e *Lead Times:* (Figura **10**) estas métricas mostram o quão rápido o trabalho está fluindo ao longo do processo. **Lead time** é o tempo que uma tarefa demorou para ser concluída, enquanto **Cycle times** são os tempos entre um processo e outro qualquer, que sejam objetos de estudo. Tentar reduzir os cycle times pode ajudar em otimizações locais dentro das equipes. Por sua vez, o lead time deve ser o menor possível, pois demonstra o quão rápido a tarefa está fluindo por todos os grupos de trabalho. Esta métrica é fundamental e simples de ser colhida, bastando se observar as datas de entrada e saída em cada etapa do fluxo, para posterior análise. Uma das ações possíveis para se reduzir o lead time é reduzir o WIP, como consequência da Lei de Little aplicada ao Kanban. (Hammarberg & Sundén, 2014), (Leopold, 2018) Ter um bom lead time ainda não garante muitas entregas: para que se possa observar quantas entregas estão acontecendo por intervalo, pode-se usar o *throughput*.

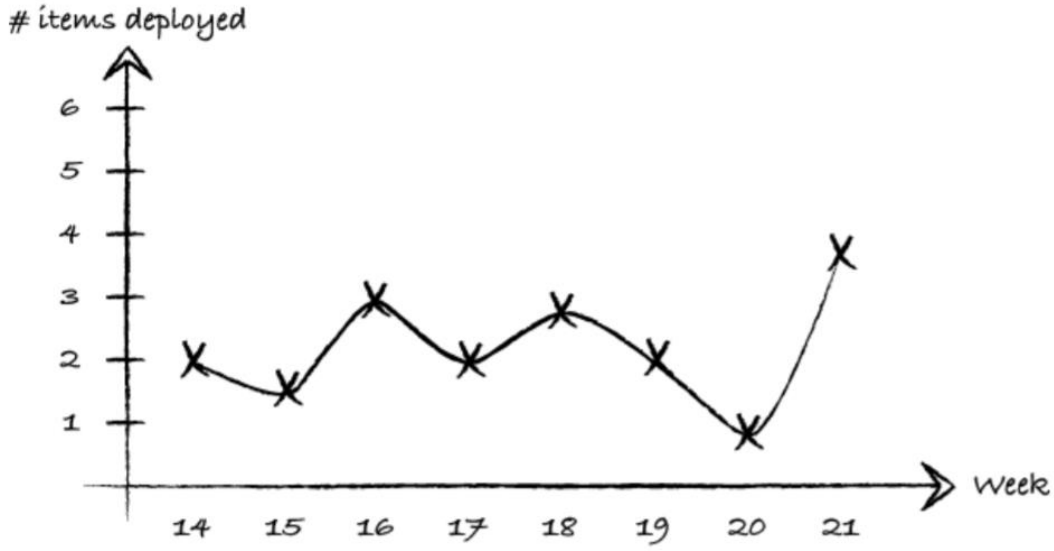
Figura 10 – *Lead time e cycle time*



Fonte: (Hammarberg & Sundén, 2014).

b) *Throughput:* (Figura **11**) é o movimento de entrada e saída de tarefas no sistema produtivo por unidade de tempo, ou seja, é o quão rápido as entregas estão acontecendo (Leopold, 2018) e, quanto maior o *throughput*, mais entregas. Para capturar esta métrica, pode-se contar quantas tarefas foram entregues durante um determinado período e visualizá-las em um gráfico, com o número de itens no eixo vertical e as unidades de tempo no eixo horizontal. Ao mesmo tempo em que há um esforço para a redução do *Lead Time,* é necessário verificar se o *throughput* está aumentando. Se o *throughput* abaixar juntamente com o *Lead time*, isso pode significar que o WIP está muito baixo e, em outras palavras, as tarefas estão sendo feitas rapidamente, mas ainda há poucas tarefas. Portanto, não há muito o que ser entregue. (Hammarberg & Sundén, 2014) Assim, uma boa gestão Kanban deve sempre buscar reduzir o *Lead Time* e aumentar o *Throughput,* que também pode ser entendido como a relação do WIP / *Lead time*. Visualiza-se de forma mais didática esta relação no *Cumulative Flow Diagram,* a ser abordado oportunamente mais adiante.

Figura 11 – Exemplo de *throughput*



Fonte: (Hammarberg & Sundén, 2014)

c) *Issues* e *blocked work items:* os *Issues* (defeitos ou problemas) são indicadores de que as entregas não estão sendo feitas com qualidade. Enumerá-los à medida que são encontrados é importante para se evidenciar

a importância da qualidade no processo, impedindo que os desenvolvedores apressem o trabalho focando em reduzir o *Lead Time* a todo custo. Itens bloqueantes, que obstruem o trabalho, também precisam ser expostos para que chamem a atenção e são fontes interessantes de análise, devendo ser entendidos e resolvidos o quanto antes pelo time e gestores. (Hammarberg & Sundén, 2014) Estes itens podem ser capturados e medidos com alterações em seus status no quadro (cores diferentes, por exemplo).

d) *Due-date performance:* quando há necessidade de se entregar um conjunto de tarefas em uma data pré-acordada, seja por uma obrigação regulatória, por um acordo de SLA ou por uma promessa a um cliente, a data limite (*due-date*) deve ser anexada às tarefas. No momento de sua conclusão, é possível observar se foi cumprida ou não. Anteriormente, ainda, é possível reorganizar a equipe para que foque nas tarefas com prazos delimitados, a fim de se evitar perder as datas. Um exemplo de visualização para esta métrica pode ser um gráfico de pizza com as marcações das tarefas entregues no prazo e fora do prazo. (Hammarberg & Sundén, 2014) Um baixo número de tarefas entregues fora do prazo dará mais confiança ao gestor ao se comprometer com as próximas, considerando, ainda, que ele conheça o *throughput* e *lead time* do time.

e) *Quality:* a palavra qualidade, isoladamente, é muito ampla e pode significar concepções diferentes para pessoas e partes interessadas diferentes. Para efeito de gestão de entrega, considera-se como boa qualidade um produto que, ao longo de sua concepção, foi construído com poucos defeitos. Com esta simplificação quantitativa, pode-se capturar algumas métricas sobre qualidade, dentre as quais elencam-se o número de defeitos por entrega (*release*); o número de defeitos sendo tratados por unidade de tempo e; a possibilidade de se usar as métricas anteriores em um quadro separado, considerando apenas os defeitos (*lead time*, *throughput,* etc.). Neste contexto, qualidade não está se referindo ao que será percebido pelo cliente, mas sim, ao que a equipe está se propondo a entregar. Quando um defeito é detectado, significa que a equipe falhou em entregar o que foi prometido conforme uma expectativa.

f) *Value demand* e *failure demand:* as demandas de valor e de falha competem por recursos entre as equipes que trabalham em sistemas já em produção, ou que já possuem pelo menos uma parte em testes. As demandas de valor são as novas funcionalidades ou qualquer demanda que traga algo novo ao produto, enquanto as de falha são aquelas geralmente originadas pela *Quality Assurance* (QA) ou pelos usuários que tinham uma expectativa em relação a uma funcionalidade, que não foi atingida em algum cenário. O objetivo desta métrica é entender o quanto do time está focado em cada uma destas classes de demanda, para que se possa priorizar uma ou outra.

g) *Abandoned* e *discarted ideas: e*sta métrica permite acompanhar quantos itens do *backlog* são descartados em vez de entrarem em produção, além de apontar quantos itens em produção foram descartados ao longo do caminho. Manter esta contagem é importante para se parametrizar que, se há poucos itens descartados no *backlog,* então poucas opções foram criadas e talvez, isso pode indicar baixa taxa de inovação; por outro lado, se há muitos itens descartados dentre os que entraram em desenvolvimento, entende-se que a equipe está iniciando muitos trabalhos que não deveriam. Estes dois parâmetros podem indicar se a equipe deveria se dedicar mais nas análises e no entendimento do negócio (Hammarberg & Sundén, 2014).

5.3.3 *Cumulative Flow Diagram* (CFD)

O CFD é útil para a visualização de uma série de informações pertinentes ao andamento do projeto, servindo de pano de fundo para discussões de melhorias no processo. É o diagrama mais utilizado, possibilitando identificar WIP, *Lead Time*, *cycle times* e gargalos no fluxo (Corona & Pani, 2013), de simples construção e permite compreender bem o impacto e a eficácia das mudanças de processo (Greaves, 2011).

O CFD é um gráfico com uma unidade de tempo no eixo horizontal e a quantidade acumulada de tarefas segregadas por etapa no eixo vertical. Para se construir o CFD, parte-se da anotação da quantidade de cada tarefa, em cada etapa do desenvolvimento, sempre ao término de cada dia. Utilizando-se uma ferramenta

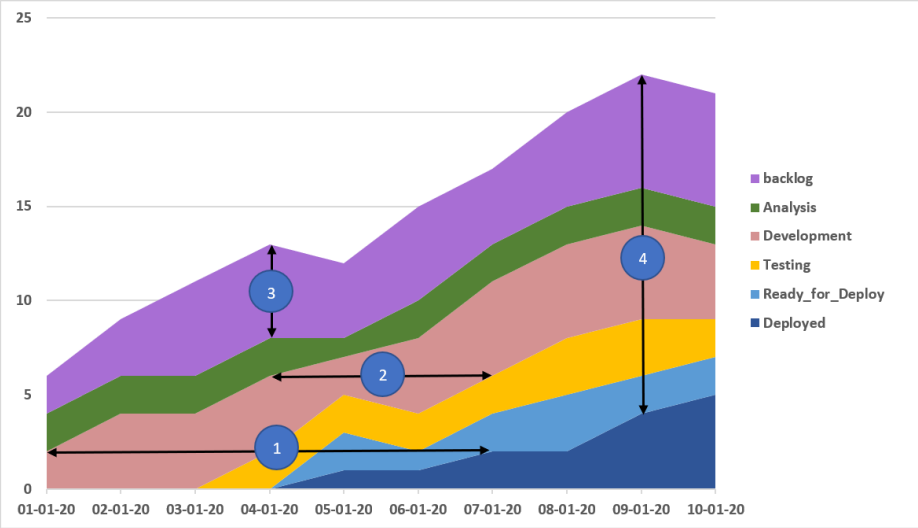
de construção de gráficos, a plotagem dos dados relativos à Figura **12** resultará num gráfico como o apresentado na Figura **13**.

Figura 12 – Exemplo de dados que alimentam o gráfico CFD

Date	backlog	Analysis	Development	Testing	Ready_for_Deploy	Deployed
01/01/20	2	2	2	0	0	0
02/01/20	3	2	4	0	0	0
03/01/20	5	2	4	0	0	0
04/01/20	5	2	4	2	0	0
05/01/20	4	1	2	2	2	1
06/01/20	5	2	4	2	1	1
07/01/20	4	2	5	2	2	2
08/01/20	5	2	5	3	3	2
09/01/20	6	2	5	3	2	4
10/01/20	6	2	4	2	2	5

Fonte: Adaptado de (Hammarberg & Sundén, 2014)

Figura 13 – Cumulative flow diagram



Fonte: Adaptado de (Hammarberg & Sundén, 2014)

O CFD permite a leitura imediata dos seguintes indicadores, visualizados no gráfico pelos respectivos números de identificação: *Lead time* (1), *Cycle time* (2), *Backlog* (3) e WIP (4). A relação entre WIP e *Lead Time* também pode ser observada. Quanto mais trabalho em andamento (4), mais alto será o gráfico (soma de todas as etapas) e maior será o *Lead Time* (1). Isto traz um ponto importante de discussão para o time avaliar que valor de WIP faz o fluxo andar mais rápido (Hammarberg & Sundén, 2014).

5.4 Utilização de quadro digital

O quadro físico Kanban é o principal elemento, ao redor do qual os membros do time de desenvolvimento se reúnem para fazer as discussões a respeito de suas dificuldades e, também, para atualizar o andamento das atividades nas reuniões diárias. Dentre os efeitos desta prática, pode-se citar a manutenção da atualização das atividades; a busca coletiva pelos pontos de gargalo e para se identificar impedimentos; a decisão de aplicação de *Swarming* para uma determinada atividade e; a melhoria da visualização do fluxo de valor (Laanti & Kangas, 2015).

É possível utilizar recursos tecnológicos como lousas digitais ou telas sensíveis ao toque para substituir o quadro físico, sem que haja perda significativa de seus benefícios dado que a manipulação durante as reuniões é similar.

Outra possível solução é a utilização de aplicações web para se exibir uma cópia do quadro para cada desenvolvedor, tendo, cada um, o acesso ao quadro digital por meio da sua estação de trabalho com a possibilidade interação.

Ambos são tipos de quadros digitais Kanban. Uma vez digitalizado, o quadro permite coleta automatizada de dados para geração das métricas e para uso nas reuniões de retrospectiva do método (Anderson, 2010).

5.4.1 Telas sensíveis ao toque ou lousas digitais

Utilizar um painel digital sensível ao toque não altera significativamente o dia a dia, no que diz respeito à aplicação original do método, trazendo junto de si a vantagem da economia gerada com a adoção do método. Contudo, tal proposta exige um investimento elevado, tornando-o, muitas vezes, inviável para muitas empresas.

Organizações que possuam recursos para aquisição de lousas digitais ou monitores grandes, sensíveis ao toque, que possibilitem que as reuniões de Kanban aconteçam da mesma forma que aconteciam com o quadro físico, se beneficiarão com os ganhos nas automações de extrações de métricas, conforme apresentado a seguir, sem terem que passar pelas adaptações que as organizações que não podem, ou não desejam fazer tal investimento, necessitariam.

5.4.2 Quadro digital compartilhado

Permitir a visualização do painel de atividades pela web é a proposta de muitos sistemas de controle de atividades on-line que estão sendo lançados no mercado como: *Lean Kit Kanban*, *Agile Zen*, *Target Process*, *Silver Catalyst*, *RadTrack*, *Kanbanery*, *Version One*, *Jira*, *Flow.io* e *Kanban tool*. Mais detalhes de algumas destas ferramentas estão disponíveis no Apêndice A.

A ausência de um quadro físico, substituído por um quadro digital que pode ser acessado de qualquer estação de trabalho dos desenvolvedores, ou de seus celulares, implica em mudanças inevitáveis em relação às reuniões diárias. Em sua proposta original, (Anderson, 2010) chama a reunião de: "*daily standup meetings*" ou reuniões diárias em pé, em uma livre tradução, sugerindo que seja realizada com os participantes em pé para que seja o mais breve possível e que cause incômodo se demorar demais.

Como não é razoável propor que os desenvolvedores fiquem de pé em suas estações de trabalho durante uma reunião *online*, uma proposta para ajudar a controlar o tempo da reunião é utilizar um cronômetro para que todos acompanhem o tempo decorrido e assim, buscar resumir melhor seus comentários.

Algumas ferramentas podem ser utilizadas para esta reunião, dentre as quais se pode elencar *Microsoft Skype*, *Microsoft Teams*, *Google Hangouts*, *Discord* entre outras.

A seguir, destacam-se as principais atividades da reunião diária e os cuidados a serem tomados para não prejudicar estas atividades com a digitalização do quadro.

5.4.2.1 Manutenção da atualização das atividades

A manutenção das atividades deve poder ser feita durante as reuniões diárias. Assim, é importante escolher uma ferramenta que facilite a movimentação das atividades de maneira fluente e simples como, por exemplo, a funcionalidade de clicar e arrastar.

5.4.2.2 Busca coletiva pelos pontos de gargalo

Como todos os integrantes do time podem acessar o quadro digital, todos podem ter a mesma visualização sobre o fluxo, para tentarem identificar os gargalos. Dessa forma, a identificação dos gargalos não deve ser prejudicada.

5.4.2.3 Busca coletiva para identificar impedimentos

As ferramentas de comunicação permitem que todos os integrantes conversem em tempo real e, mesmo quem participa remotamente pode compartilhar suas opiniões e relatar suas dificuldades para todos. Assim, é possível identificar eventuais impedimentos de maneira similar às reuniões em torno do quadro físico.

5.4.2.4 Decisão de aplicação de *Swarming* para uma determinada atividade

Quando um impedimento é anunciado por um dos participantes, é possível atribuir mais pessoas para o apoiar. As ferramentas podem prover estas funcionalidades para que esta ajuda seja exposta de maneira clara para todos, caso contrário, ainda é possível indicar esse evento com uma observação na própria tarefa.

5.4.2.5 *Propostas de melhorias*

Foram encontrados diversos relatos na bibliografia sobre empresas que já utilizam quadros virtuais para auxiliar em suas atividades, muitas delas, inclusive, com equipes remotas e que demandam esta alternativa (Tanner & Dauane, 2017).

Utiliza-se como estudo, o caso de uso de uma destas empresas, detalhado no capítulo 6.

5.4.3 Considerações sobre o Kanban em relação a outros métodos

Em uma recente revisão da literatura sobre Scrum e Kanban, (Ozkan, Bal, Erdogan, & Gok, 2022) conclui que o Kanban, em muitos aspectos e aplicações, é

mais vantajoso em relação ao Scrum. Os resultados do estudo também mostram uma porcentagem considerável de transições do Scrum para modelos híbridos e para o Kanban. Outros estudos, mostram que migrações do Scrum para o Kanban tem acontecido devido a problemas com o próprio método Scrum (Nikitina & Kajko-Mattsson, 2011), (Sjøberg, Johnsen, & Solberg, 2012), (Ahmad, Kuvaja, Oivo, & Markkula, 2016).

O Kanban é mais flexível e adaptável do que o Scrum, sem as chamadas “time-boxes” ou “*sprints*”, papéis, e regras, as equipes Kanban sentem-se mais capazes de lidar com seus processos internos e responder rapidamente com seu gerenciamento de fluxo contínuo, permitindo o replanejamento constante diante da incerteza, mudanças frequentes e respostas mais rápidas (Ozkan, Bal, Erdogan, & Gok, 2022).

Não exige grandes mudanças no comportamento diário da equipe de desenvolvimento pois propõe poucas cerimônias e se encaixa no processo em andamento (Ozkan, Bal, Erdogan, & Gok, 2022). Já o Scrum, por introduzir diversas cerimônias, sofre adaptações a cada implementação de acordo com os costumes da organização (Raunak & Binkley, 2017).

Além de proporcionar uma melhor qualidade, cronograma de projeto, gerenciamento de riscos e recursos, confiabilidade, resultados de tempo de execução, desempenho, visibilidade, rastreabilidade, trabalho em equipe, satisfação dos membros individuais da equipe, feedback regular mais rápido e contatos mais próximos com os usuários, o Kanban também não impõe prazo para a conclusão de conjunto de tarefas em janelas de tempo como no Scrum e, portanto, resulta em melhor qualidade e menos estresse conforme relatado por diversos estudos que o compararam qualitativamente e quantitativamente com o Scrum (Ozkan, Bal, Erdogan, & Gok, 2022).

6 Estudo de caso

O desenvolvimento de software é conduzido por indivíduos, grupos e organizações, ou seja, envolve atividades multidisciplinares, que são áreas em que estudos de casos são conduzidos normalmente, com forte influência de questões sociais, culturais e políticas. Isso significa que muitas questões de pesquisa em engenharia de software são adequadas para a aplicação de estudo de caso. (Runeson & Höst, 2008).

As questões de pesquisa QP1 (Como o quadro digital Kanban pode ser usado para o acompanhamento de atividades?) e QP2 (O uso do quadro digital afeta o acompanhamento do andamento das atividades?) são endereçadas neste estudo de caso. Este método de pesquisa é adequado para responder a estas perguntas, pois atende as seguintes condições de pesquisa, elencadas por (Yin, 2017):

a) Forma da questão de pesquisa estruturada em: “como” o quadro digital pode ser usado;
b) Não requer controle sobre os eventos comportamentais;
c) É focado em eventos contemporâneos.

Por sua vez, os principais processos para serem seguidos durante o estudo de caso são apontados por (Runeson & Höst, 2008):

a) Planejamento: definição dos objetivos e planejamento das etapas;
b) Preparação para coleta de dados: definição dos procedimentos e protocolos para coleta de dados;
c) Coleta de evidências: execução do estudo com a coleta de evidências;
d) Análise dos dados coletados;
e) Divulgação dos resultados.

6.1 Planejamento

O planejamento deste estudo de caso está dividido da seguinte forma:

a) Objetivo – o que é esperado encontrar como resultado;
b) Objeto de estudo – o que é estudado;
c) Questões de pesquisa – que se quer saber;
d) Métodos – como os dados serão coletados;
e) Estratégia de seleção – onde os dados serão procurados.

6.1.1 Objetivo

O objetivo deste estudo de caso é entender como as atividades foram acompanhadas em um projeto de desenvolvimento de software que envolve a evolução de um sistema existente (em linguagem C/C++), adaptando-o a novas tecnologias (C# .net) e a uma nova arquitetura (micro serviços), em uma empresa de grande porte com a utilização de um quadro digital. Também se objetiva compreender se houve muitas discrepâncias entre as movimentações do quadro e a execução real das atividades.

6.1.2 Objeto de estudo

O objeto de estudo é o quadro digital de Kanban, no contexto do projeto citado acima. A organização para a qual o projeto foi estudado possui algumas características particulares como, por exemplo, fazer entregas em produção em intervalos regulares de, no mínimo, seis meses. Para deixar o estudo devidamente caracterizado, outros aspectos da organização serão detalhados a seguir.

6.1.2.1 Entendimento das características da organização

A organização possui diversos produtos e equipes específicas para que atendam cada um deles. O projeto foco deste estudo de caso foi executado por uma equipe de um dos produtos, possuindo, aproximadamente, quinze pessoas, dois coordenadores, um gerente e um gerente sênior. Dos desenvolvedores, nenhum de nível júnior, dois plenos e o restante seniores. Todos possuem conhecimento básico do método Kanban, do sistema de versionamento *Git* e da ferramenta de controle de tarefas, *Jira*, que foi usado também como quadro digital Kanban.

6.1.2.2 Processo de desenvolvimento

O processo de desenvolvimento de uma determinada tarefa se inicia já na sua priorização, pois, quando uma tarefa é priorizada, entra no quadro Kanban e o tempo de desenvolvimento começa a ser contabilizado. O primeiro desenvolvedor livre puxa a tarefa e inicia o desenvolvimento, devendo sinalizar que iniciou a tarefa no *Jira*. Ao completá-la, o desenvolvedor deve confirmá-la (*comitar*) no *Git* e indicar a sua conclusão no *Jira*. Por conseguinte, outro desenvolvedor deve revisar a tarefa e indicar no Jira o término da revisão. Dessa forma, um dos coordenadores ou os gerentes podem aplicar a alteração no código (merge na branch de desenvolvimento).

6.1.2.3 Papéis e responsabilidades

Os desenvolvedores, além de, evidentemente, desenvolverem as tarefas definidas pelos coordenadores, devem também reportar dificuldades que impeçam o avanço, relatar o término de cada tarefa, revisar as tarefas executadas por outros desenvolvedores e se reportarem ao seu respectivo coordenador.São atribuições comuns aos coordenadores: acompanhar e auxiliar o trabalho dos desenvolvedores, auxiliar na definição de implementação da arquitetura, detalhar os requisitos e atividades, criando pequenas tarefas, resolver impedimentos no processo de desenvolvimento e reportar para o gerente.

Os gerentes, por sua vez, imbuem-se de acompanhar o desenvolvimento das atividades dos desenvolvedores, garantir a implementação da arquitetura, cronograma, reportar para o gerente sênior e gerenciar a entrega do produto junto a outras equipes responsáveis.

Por fim, cabe ao gerente sênior definir a arquitetura, o cronograma, a estratégia de desenvolvimento e dimensionar a equipe para acomodar eventuais mudanças de requisitos ou demandas estratégicas.

6.1.2.4 Geração do backlog

Após a quebra dos requisitos e as definições das atividades pelos coordenadores, o *backlog* é gerado e priorizado. Não há um momento no tempo

específico para se revisar o *backlog*, porém, sempre que há um evento externo que exija uma alteração, o *backlog* pode ser modificado. Esta é uma das características do Kanban: nenhuma mudança na direção e nos requisitos do projeto afeta negativamente o time de desenvolvimento. Tudo o que está no *backlog* pode ser modificado, removido ou substituído sem prejuízo, ou até mesmo ciência desse time, que apenas se preocupa com as atividades que eles se comprometeram a fazer (coluna *TO DO*) em diante.

6.1.2.5 Definição de pronto

O time de desenvolvimento possui um *checklist* que precisa ser atendido para considerar uma tarefa como pronta. Entre outros itens do *checklist*, estão:

a) Revisão do código por um ou mais pares;
b) Completude de 95% do código com testes unitários;
c) Atualização do andamento da tarefa no quadro de acompanhamento *(Jira)*;
d) Atualização do código no controlador de versão *(Git)* com indicação da tarefa pertinente.

Há ainda outros itens, considerados irrelevantes para este estudo.

6.1.2.6 Reuniões diárias

As reuniões diárias aconteceram via ferramenta de comunicação de grupo (*Skype for business*), neste caso, com cada desenvolvedor em sua máquina. Nas reuniões, os seguintes itens eram discutidos:

a) Dificuldades individuais eventuais;
b) Próximas atividades e priorização;
c) Melhorias nos fluxos e nos processos;
d) Compartilhamento de conhecimentos pontuais de alta relevância;
e) Atualização das tarefas em atraso no quadro de acompanhamento.

O projeto seguiu por dois anos e, durante este tempo, centenas de tarefas foram executadas, diversos *bugs* foram captados pelo time de qualidade e foram introduzidos e priorizados no *backlog*, sendo que os mais críticos receberam prioridade máxima e entravam sempre como alta prioridade.

6.1.3 Questões de pesquisa

As questões de pesquisa deste trabalho estão endereçadas no estudo de caso.

> QP1: Como o quadro digital pode ser usado para o acompanhamento de atividades Kanban?
> QP2: O uso do quadro digital afeta o acompanhamento do andamento das atividades?

6.1.4 Métodos

Previamente ao início do projeto, os gerentes e os coordenadores já haviam definido que o software *Jira* seria utilizado para controle das atividades, que o método Kanban seria adotado e, também, que o quadro de acompanhamento deveria ser provido pelo *Jira*. A tela para acompanhamento das atividades de projeto conforme o método Kanban define já é provida pela ferramenta. Alguns poucos atributos precisaram ser configurados como: nome das etapas de desenvolvimento e o fluxo permitido entre uma etapa e outra.

O software também permite usar os grupos de rede de usuários para distribuir as permissões conforme o papel de cada um. Dessa forma, os desenvolvedores tinham permissões para criar, alterar e apagar tarefas. Já os coordenadores, além destas permissões, também podiam criar etapas no fluxo de tarefas, bem como definir os fluxos de trabalho, ou seja, como as alterações de estados poderiam ser feitas.

As colunas foram definidas, os perfis atribuídos e o projeto foi iniciado na reunião de *Kick-off*, com a apresentação do escopo e das primeiras tarefas a serem desenvolvidas. Na primeira reunião, um dos coordenadores utilizou o quadro digital e arrastou a primeira tarefa da coluna inicial (*backlog*) para a coluna de itens priorizados

(*TO DO*), que seriam atendidos de imediato. Ao observarem a animação da tarefa se deslocando da esquerda para a direita, os coordenadores deixaram claro que aquela seria a dinâmica do dia a dia e que utilizariam o quadro para acompanhar o andamento durante todo o projeto. Assim, os gestores definiram uma maneira de se utilizar o quadro digital de Kanban (QP1).

6.1.5 Extração das informações

As ferramentas utilizadas no projeto (*Jira* e *Bitbucket*) disponibilizam APIs para consulta, possibilitando que alguns scripts fossem desenvolvidos para se extraírem os dados de evolução do projeto, a fim de serem analisados. O atributo "*id*" (abreviado do inglês - *identification*) foi escolhido para identificar as tarefas, pois é o mesmo em ambas as ferramentas. A coleta de dados do estudo de caso, portanto, foi feita de maneira automatizada após o desenvolvimento do script na etapa final do estudo.

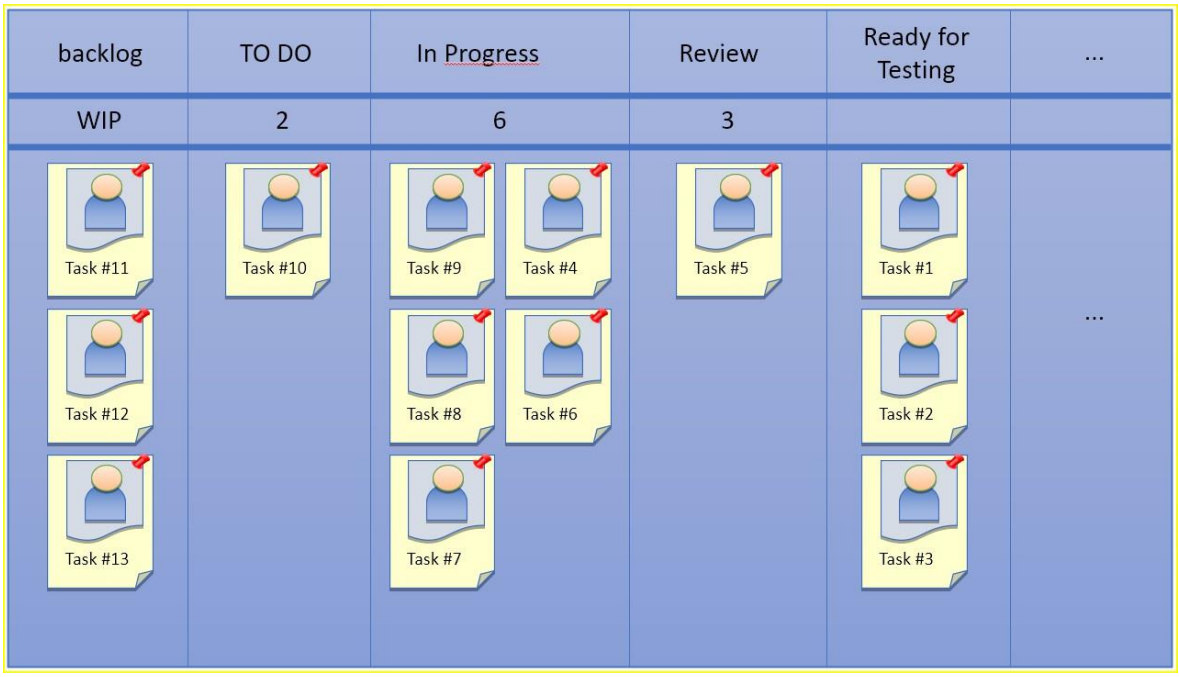
6.1.6 Estratégia de seleção

Foi elaborada uma estratégia de seleção para cada tipo de dado em sua fonte específica: Dados do quadro digital (*Jira*) e dados relativos ao desenvolvimento efetivo das atividades (*Git*).

6.1.6.1 Extração dos dados do quadro digital

O quadro digital de Kanban (Figura 14) foi montado na ferramenta *Jira*, seguindo a premissa de que as diversas colunas do quadro representam o estágio de desenvolvimento da atividade. Para efeito deste estudo, as colunas: *In Progress* e *Ready for Testing* foram as escolhidas como foco de referência para marcação de início e fim. Em outras palavras, quando a tarefa deixa a coluna *TO DO* e é movida para *In Progress* no conceito do projeto, o time de desenvolvimento se compromete com sua execução e o relógio começa a contar. Quando a tarefa é movida para *Ready for Testing*, significa que o desenvolvimento foi concluído e já foi revisado. Os scripts criados utilizando APIs do *Jira* focaram em extrair estas informações para cada tarefa.

Figura 14 - Representação simplificada do quadro kanban



Fonte: elaborado pelo autor

6.1.6.2 Extração dos dados relativos ao desenvolvimento das tarefas

Devido ao fato de o código ser versionado utilizando-se padrões de desenvolvimento com a criação de *branches*, foi possível utilizar seus nomes e atributos como marcadores de início e fim de desenvolvimento pelo software de controle de versão (*Git*). Assim, associa-se a data de criação dos *branches* como "início do desenvolvimento" e o "*merge*" desta branch na "*develop*" (nome da *branch* que contempla o conjunto de *features* estáveis e que farão parte da entrega seguinte), como a sua conclusão. Os scripts criados utilizando as APIs do *Bitbucket* focaram em extrair estas informações para cada tarefa. Posteriormente estes dados puderam ser comparados pois a convenção de nomes das branches continham o "*Id*" da tarefa do software de controle de atividades (*Jira*).

6.2 Preparação para a coleta de dados

O Nível de coleta de dados adotada neste estudo, conforme a divisão de (Lethbridge, Sim, & Singer, 2005), é a de segundo grau (*Second degree*), sem

interação direta com os indivíduos, mas com acesso aos dados originais armazenados pelos softwares. O método de coleta utilizado foi o de Métricas (*Metrics*), dado que o estudo está interessado em dados quantitativos (Runeson & Höst, 2008) para a questão de pesquisa QP2.

6.2.1 Controle de versão de código fonte

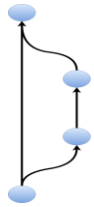
O versionamento de código fonte é assunto bastante antigo e já amplamente utilizado, sobretudo em projetos de grande escala. No entanto, apresenta-se uma breve descrição para clarificar como o software de controle de versão foi utilizado para coleta de dados do estudo de caso deste projeto. O software de controle de versão define como “Master” a “linha” imaginária do tempo que contém o código consolidado. Quando uma tarefa é iniciada, um ramo (*branch*) é criado a partir do código consolidado (*Master*) e as alterações são registradas com segurança e de maneira isolada. O *software* não registra o momento de criação do *branch*, e sim, o primeiro registro (*commit*) a ele associado. Esta informação pode variar muito conforme a maneira como o desenvolvedor trabalha, ou seja, ele pode fazer muito trabalho em sua máquina local, sem enviar para o servidor, por vários dias e enviar tudo quando estiver prestes a terminar a tarefa. Dessa forma, adotou-se a mesma data registrada pelo quadro de atividades, como a data de criação do ramo (*Branch*).

A segunda etapa que foi observada neste software não está representada na tabela, mas figura nos registros: a data de abertura do pedido de revisão (*merge-request*). Quando o desenvolvedor faz esta requisição, significa que já terminou o desenvolvimento da tarefa e agora solicita a aprovação de um revisor - no quadro Kanban, é quando se altera a atividade para a coluna “Aguardando Revisão”.

A terceira etapa é o “*merge*” ou união do código desenvolvido na tarefa com o código fonte principal consolidado e que representa a última atividade de desenvolvimento. A tarefa, agora, aguarda os testes ou é concluída se for uma tarefa técnica ou não-funcional do ponto de vista de negócio.

A Figura 15, a seguir, mostra os possíveis estados da tarefa representados no software de controle de versão:

Figura 15 - Fluxo de uma tarefa no software de controle de versão

	*Merge* (aprovação das alterações por um revisor)
	Registro (atualizações da atividade no *branch*)
	Registro (criação do *branch* que representa uma tarefa #)
	Master (código consolidado anterior à tarefa)

Fonte: elaborado pelo autor

6.2.2 Estrutura dos dados

Tanto o sistema *Git*, quanto o *Jira,* utilizados no projeto, fornecem APIs para extração de dados, com possibilidade de se escolher quais dados sobre cada elemento se quer extrair. Para o *Git*, cada registro significa um *commit*, ou seja, uma alteração em um ou mais arquivos, que consiste numa mudança de código que pode representar a resolução de um problema ou a adição de uma nova funcionalidade. Essas alterações se relacionam com o estado do código, antes e depois, relação fundamental para este estudo, pois é ela que expõe se uma alteração está sendo considerada como finalizada ou não. Para a ferramenta *Jira*, cada registro identifica uma tarefa, enquanto alguns dos atributos representam seu estado. Se um determinado estado ainda não foi atingido, o atributo é nulo e, quando é atingido, o atributo apresenta a data em que a tarefa entrou naquele estado. Os dados foram extraídos do *Jira* considerando os atributos dispostos na Figura 16.

Figura 16 - Estrutura dos dados extraídos da ferramenta de controle de projeto

Id – Número de identificação da tarefa no Jira
IssueType – Tipo de tarefa (correção de bug ou nova funcionalidade)
CreatedAt – Data da criação da tarefa no Jira
InProgress – Data de início de desenvolvimento.
WaitingReview – Data do término do desenvolvimento (envio para revisão)
UnderReview – Data do início da revisão.
Closed – Data do término da tarefa
ReadyForTesting – Data do término da tarefa (enviada para outra equipe).

Fonte: elaborado pelo autor

A extração dos registros do *GIT* considerou os atributos apresentados na Figura 17.

Figura 17 - Estrutura dos dados extraídos da ferramenta de controle de versão

Name – Nome da branch
Project – Nome do projeto
Repo – Nome do repositório
Title – Título da branch
Description – Descrição da branch
Id – identificação da branch
Created – Data de criação
Updated – Data do último update
FromRef – Nome da branch que originou o último *pull-request* deste branch.
ToRef – Nome da branch de destino do último *pull-request* desta branch.
From – Nome de exibição da branch que originou o último *pull-request* deste branch.
To – Nome de exibição da branch de destino do último *pull-request* desta branch.

Fonte: elaborado pelo autor

De acordo com os processos pré-definidos, o nome (*display-id*) das *branches* (*Git*) contém o mesmo padrão de nome das tarefas cadastradas no *Jira*. A coleta dos dados foi feita de maneira automatizada, considerando o Id para efeito de comparação desta estrutura.

6.3 Coleta de dados

Durante todo o projeto, os coordenadores, juntamente com seus desenvolvedores, realizavam reuniões diárias nas quais o progresso de cada desenvolvedor era reportado e os avanços registrados no *Jira*. Nestas reuniões, também era possível conversar sobre dúvidas particulares e priorizar tarefas. Cada desenvolvedor permanecia em sua própria estação de trabalho durante as reuniões e a tela do projeto era compartilhada para todos, com a comunicação entre os membros ocorrendo via VOIP (*voice over IP*) através da ferramenta *Skype*. É importante ressaltar que, nos dias em que as reuniões não aconteciam, algumas atividades eram

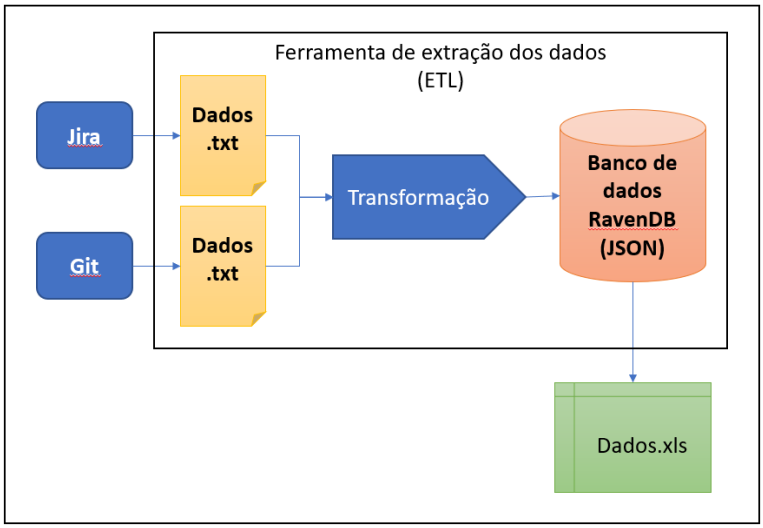
atualizadas com atraso no dia seguinte, quando a reunião se iniciava e o desenvolvedor reportava ou era lembrado pelos colegas ou pelo coordenador. Estas iterações com a ferramenta geravam dados sobre a movimentação, assim como as ações executadas no software de controle de versão. Ao que se chama aqui de “final do projeto”, é na verdade o final do estudo. Realiza-se uma extração do início do projeto até o momento presente, de todo o andamento até então para servir como base para este estudo.

6.3.1 Ferramentas de coleta e análise dos dados

As ferramentas de coleta de dados foram as seguintes:

a) APIs do Jira para ETL dos dados das reuniões*;*
b) APIs do github para ETL dos dados do desenvolvimento*;*
c) Ferramenta criada para tratamento e exportação dos dados.

Figura 18 - Fluxo da coleta de dados



Fonte: elaborado pelo autor

O banco de dados RavenDB foi usado como repositório dos dados estruturados, por ser um dos bancos que permitem armazenar, de forma direta, uma estrutura de dados em formato correspondente ao JSON (*java script object notation*).

As APIs do Jira e do Github foram utilizadas para extrair os dados e foram disponibilizadas em um diretório para importação.

Os dados foram importados, transformados para estruturas em memória e exportados no formato de planilha eletrônica para permitir análise posterior conforme mostrado na Figura 18.

6.4 Análise dos dados coletados

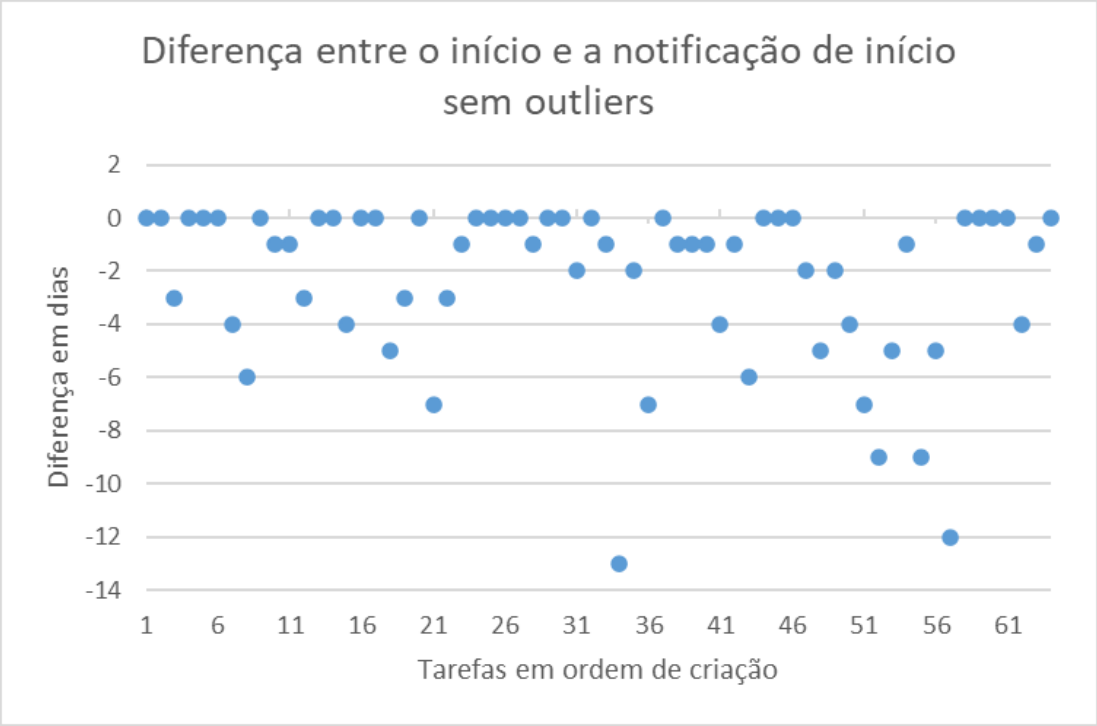
O momento para extração dos dados, cerca de dois anos após o seu início, precisou ser antecipado por motivos de mudanças organizacionais que dificultariam a continuidade do seu acompanhamento. Contudo, foi possível extrair 212 atividades identificadas como tarefas de novas funcionalidades pelo *Jira* e 436 *branches* no Bitbucket (*Git*). O número maior de *branches,* em relação ao número de tarefas, está relacionado ao fato de, às vezes, uma tarefa desencadear mais de uma *branch* de trabalho, sendo apenas uma *branch* principal e as demais, *branches* secundárias. Todas as secundárias foram ignoradas, pois a tarefa só é concluída quando a *branch* principal é finalizada.

Após a curadoria, 97 tarefas apresentaram dados completos em relação às propriedades que se faziam necessárias em ambas as bases de dados. Os atributos que eliminaram as tarefas desprezadas foram, por exemplo, a falta de uma data de término, indicando que a tarefa ainda estava em execução ou a ausência da sua existência na outra base.

Depois de realizado o processamento dos dados colhidos, foi possível observar que as tarefas, em sua maioria, não foram iniciadas no mesmo dia em que foram marcadas como iniciadas (71%). Em alguns casos, o atraso foi maior e apareceram alguns *outliers* que, em uma análise pontual posterior, foram identificados como assuntos não relevantes ou que deveriam ser postergados (duas tarefas foram excluídas do conjunto de dados após esta análise). Essas informações foram encontradas nos comentários das próprias tarefas. Apesar da alta diferença para iniciar as tarefas, o término se mostrou mais conciso: cerca de 61% das atividades finalizaram no mesmo dia ou com até 1 dia de atraso de notificação. Os próximos dois gráficos (Figura **19** e Figura **20**) ilustram os registros destas diferenças para início e término.

No eixo X, cada ponto representa uma tarefa e o tempo avança da esquerda para a direita de forma que as tarefas à direita começaram antes que uma tarefa à esquerda. O eixo Y representa a diferença em dias a partir do momento em que houve a decisão de iniciar a tarefa, ou seja, o comprometimento de iniciá-la e o seu início efetivo registrado pelos desenvolvedores. Os valores ficam negativos porque os dados tratam da diferença de dias entre a marcação registrada no quadro digital (jira) e a marcação registrada no software de controle de versionamento do código fonte (*Git*).

Figura 19 - Diferença entre início e notificação de início das atividades (sem *outliers*)

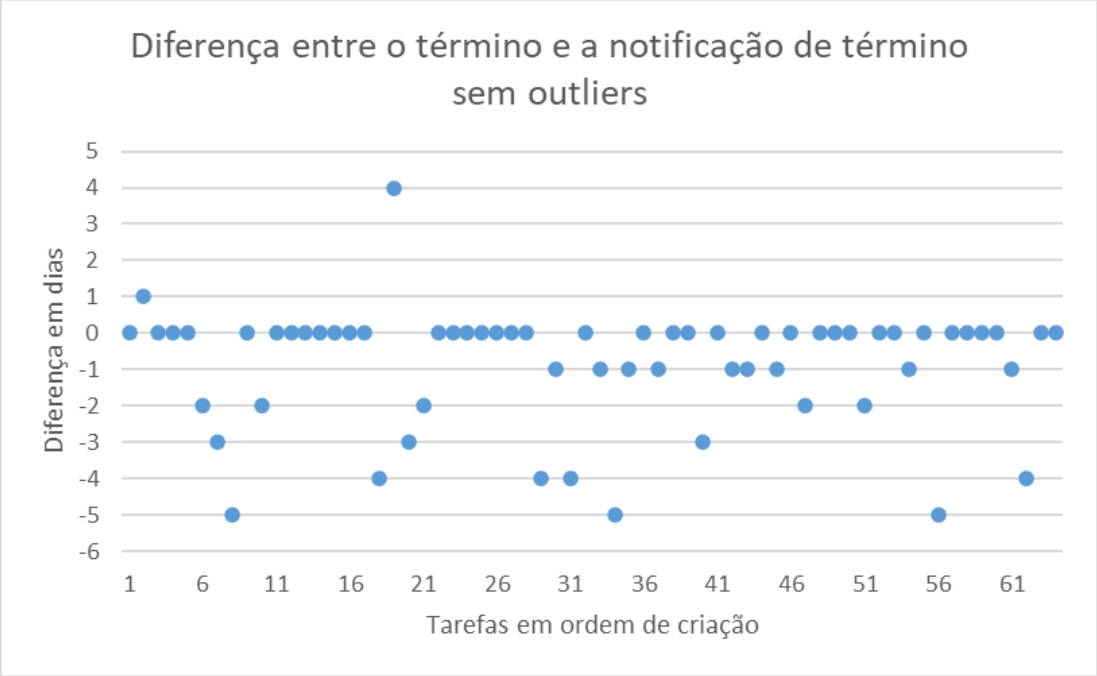


Fonte: elaborado pelo autor

A Média e desvio padrão das marcações de início das atividades foram, respectivamente:

a) Média: -2.297
b) Desvio padrão: 3.08

Figura 20 - Diferença entre término e notificação de término das atividades (sem *outliers*)



Fonte: elaborado pelo autor

Média e desvio padrão das marcações de término das atividades foram, respectivamente:

a) Média: -0.844
b) Desvio padrão: 1.63

Aplicação do erro quadrático médio (EQM):

$$EQM = \frac{1}{N} \times \sum_{i=1}^{n} (y_i - \bar{y}_i)^2$$

Legenda:
N: número de registros
y: valores observados
Ȳ: valores esperados

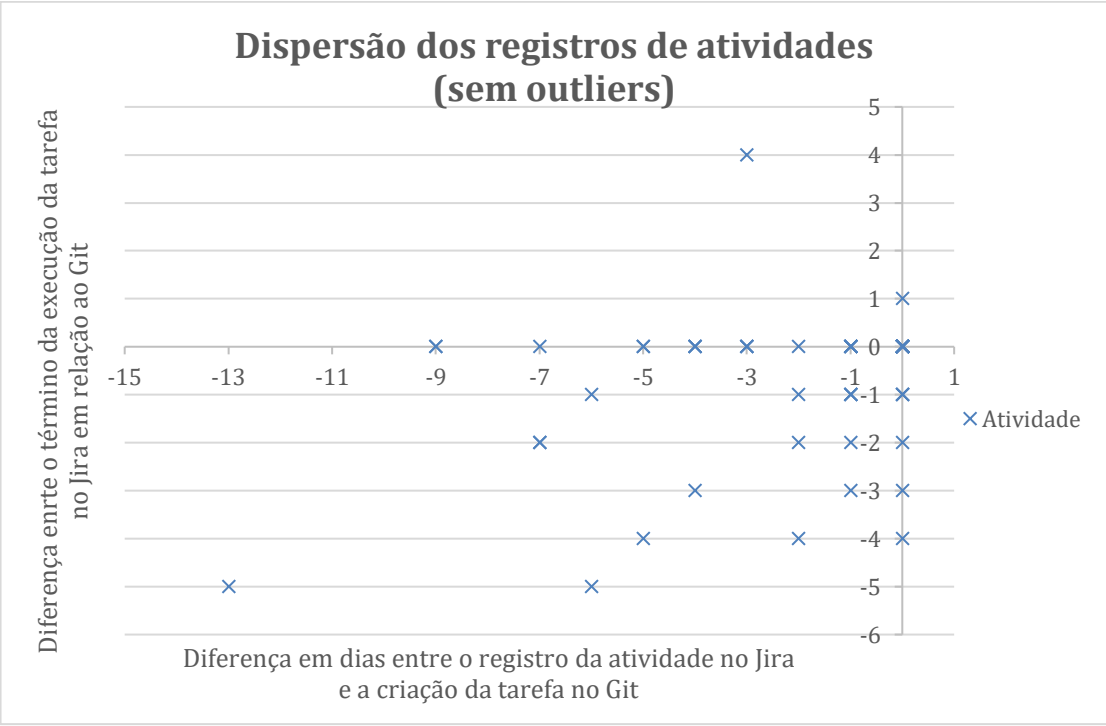
a) Início das atividades:
   0.386
b) Término das atividades:
   0.203

Considerando que as tarefas acordadas em iniciar em certo dia, se iniciadas no dia planejado resultaria em um gráfico linear com todos os pontos sobre o eixo X,

o erro quadrático médio seria 0 (zero). O erro quadrático médio inferior a 0.4 dias para os registros de início e término indica que as marcações foram aderentes ao esperado que era de 0 a 1 dia.

Outro modo de observar os resultados é pela plotagem da dispersão (Figura 21) entre as diferenças de marcações de início e término. No eixo X, para cada tarefa, a diferença em dias entre o registro no *Jira* em relação ao *Git*. No eixo Y, a diferença para o registro do término.

Figura 21 - Comparação entre as marcações efetuadas manualmente pela ferramenta de controle de projeto (JIRA) e os registros efetivamente observados durante o projeto (Git) (sem *outliers*)



Fonte: elaborado pelo autor

Com estes resultados, foi possível concluir que o uso do quadro digital de acompanhamento de atividades (*Jira*) mostrou, na maioria das vezes, que as atividades marcadas como iniciadas ou finalizadas estavam, de fato com este status (QP1).As atividades que foram registradas com atraso não impactaram no controle da gestão do projeto, pois as entregas ao cliente se deram em intervalos muito longos, aproximadamente, de seis em seis meses, o que mitigou eventuais problemas que tais atrasos pudessem causar. Além disso, o uso do quadro digital permitiu a extração

automatizada das informações supracitadas, além da extração de dados para geração de gráficos de CFD e *throughput*, que foram usados internamente dentro do fluxo do Kanban (QP2).

Como estas diferenças podem ser mais impactantes em equipes que possuem entregas mais dinâmicas, propõe-se que o sistema de controle de versão seja integrado com o quadro digital Kanban de modo a avançar a tarefa automaticamente conforme os respectivos eventos no desenvolvimento ocorrerem (QP2).

Ou seja, no contexto deste estudo de caso, a proposta seria: mover automaticamente a tarefa no quadro digital (*Jira*) no momento em que a tarefa for criada no software de controle de versão (*Git*) e bloquear a inicialização manual no quadro.

Ao mesmo tempo, bloquear o movimento de encerramento da tarefa pelo quadro e vincular este movimento à finalização da tarefa (merge na branch develop no repositório central) do *Gitb*.

Algumas ferramentas de mercado como o *github*, *gitlab* e *bitbucket* já possuem webhooks para os eventos de merge e, na outra ponta, o *Jira, Trello e Asana*, por exemplo, oferecem APIs para receberem estes eventos e viabilizar essa integração.

6.5 Divulgação dos resultados

Os resultados coletados no estudo de caso foram disponibilizados na plataforma *Zenodo* sem a remoção dos *outliers* de forma que podem ser acessados, validados e, até, utilizados por trabalhos futuros (Alexandre, 2020).

7 Conclusão

Idealizado por (Anderson, 2010), o uso do método Kanban aplicado ao desenvolvimento de software obteve diversos resultados positivos e tem evoluído ao longo dos anos. O tema tem atraído autores de livros como (Leopold, 2018) e (Burrows, 2014) sendo, talvez, o método que mais cresce dentro do movimento ágil (Corona & Pani, 2013).

O Kanban induz as equipes a atuarem de maneira espontânea nas tarefas de maior prioridade mesmo em situações em que há muitas alterações nas priorizações de atividades como, por exemplo, na manutenção de software (Ahmad, Kuvaja, Oivo, & Markkula, 2016).

Muitos usuários do Kanban relataram que o método é fácil de usar, de aprender e que melhora a produtividade e a qualidade dos seus trabalhos (Ahmad, Markkula, & Oivo, 2016)

A troca do quadro físico pelo quadro digital é mais um importante marco evolutivo para o método, pois viabiliza a extração automatizada de métricas e a redução do erro humano durante as atualizações diárias das atividades (Wan & Chen, 2008) e tem ajudado diversas empresas globais a desenvolverem software (Tanner & Dauane, 2017).

Outra importante contribuição da digitalização é a criação de oportunidades para integrações entre o quadro e as ferramentas de versionamento de código fonte e de entrega contínua, além de outras ferramentas de gestão para extração das métricas.

As diferenças apresentadas entre as marcações de início e fim das atividades do ciclo de desenvolvimento no estudo de caso mostram que há espaço para melhorias. Assim, foi proposto que o quadro digital Kanban seja integrado à ferramenta de controle de versão de modo que as atividades sejam iniciadas no quadro mediante a criação de suas respectivas *branches* no software de controle de versão e finalizadas no quadro, apenas quando forem integradas (mergeadas) no repositório central.

Com informações mais consistentes, os gestores e equipes poderão ter maior confiança e segurança para tomarem decisões.

7.1 Trabalhos futuros

É possível criar integrações entre as ferramentas de versionamento de código e a ferramenta de controle de projeto, de modo que a tarefa mude de status automaticamente, conforme algumas etapas do desenvolvimento são cumpridas. Dessa forma, haveria uma consistência fiel entre os cartões das atividades que estão vinculadas ao desenvolvimento de código e a conclusão efetiva do trabalho, mitigando, assim, o risco do erro humano no registro das atividades.

Acompanhar um projeto com esta integração seria uma oportunidade para verificar se esta proposta elimina as defasagens encontradas e pesquisas com os gestores podem mostrar se essa redução poderia trazer benefícios reais na gestão do projeto.

8 Referências

Abrahamsson, P., Salo, O., Ronkainen, J., & Warsta, J. (25 de 09 de 2002). Agile software development methods: Review and analysis. *arXiv preprint arXiv:1709.08439*. doi:10.48550/arXiv.1709.08439

Ahmad, M. O., Dennehy, D., Conboy, K., & Oivo, M. (03 de 2018). Kanban in software engineering: A systematic mapping study. *The Journal of Systems and Software [s.l], 137*, pp. 96-113. doi:10.1016/j.jss.2017.11.045

Ahmad, M. O., Kuvaja, P., Oivo, M., & Markkula, J. (2016). Transition of Software Maintenance Teams from Scrum to Kanban. *2016 49th Hawaii International Conference on System Sciences (HICSS)* (pp. 5427-5436). Hawaii: IEEE. doi:10.1109/HICSS.2016.670

Ahmad, M. O., Liukkunen, K., & Markkula, J. (2014). Student perceptions and attitudes towards the software factory as a learning environment. *2014 IEEE Global Engineering Education Conference (EDUCON)* (pp. 422-428). Istanbul, Turkey: IEEE. doi:10.1109/EDUCON.2014.6826129

Ahmad, M. O., Markkula, J., & Oivo, M. (2013). Kanban in software development A systematic literature review. *2013 39th Euromicro Conference on Software Engineering and Advanced Applications* (pp. 9-16). Santander, Spain: IEEE. doi:10.1109/SEAA.2013.28

Ahmad, M. O., Markkula, J., & Oivo, M. (2016). Insights into the perceived benefits of Kanban in software companies: practitioners' views. *INTERNATIONAL CONFERENCE ON AGILE SOFTWARE DEVELOPMENT* (pp. 156-168). Finland: Springer, Cham. doi:10.1007/978-3-319-33515-5_13

Al-Baik, O., & Miler, J. (01 de 12 de 2015). The kanban approach, between agility and leanness: a systematic review. *Empir Software Eng 20, 20*, pp. 1861–1897. doi:10.1007/s10664-014-9340-x

Alexandre, T. M. (11 de 12 de 2020). Kanban tasks opening and close dates differences between git and Jira. São Paulo, SP, Brasil. doi:10.5281/zenodo.4317101

Alsaqqa, S., Sawalha, S., & Abdel-Nabi, H. (10 de 07 de 2020). Agile Software Development: Methodologies and Trends. *International Journal of Interactive Mobile Technologies (iJIM)*, pp. 246-270. doi:10.3991/ijim.v14i11.13269

Anderson, D. J. (2010). *Kanban: Successful Evolutionary Change for Your Technology Business. [s.l.]:.* Blue Hole Press. doi:9780984521401

Anderson, D. J., & Carmichael, A. (2016). *Essential Kanban condensed.* Seattle, Washington: Lean Kanban University press. Fonte: ISBN: 978-0-9845214-2-5

Beck, K., & Andres, C. (2004). *Extreme Programming Explained: Embrace* (2 ed.). Massachusetts: Addison Wesley Professional.

Birkeland, J. O. (2010). From a Timebox Tangle to a More Flexible Flow. *Agile Processes in Software Engineering and Extreme Programming* (pp. 325-334). Berlin: Springer Berlin Heidelberg. doi:10.1007/978-3-642-13054-0_35

Burrows, M. (2014). *Kanban from the Inside: Understand the Kanban Method, connect it to what you already know, introduce it with impact.* Blue Hole Press.

Corona, E., & Pani, F. E. (01 de 2013). A Review of Lean-Kanban Approaches in the Software Development. *WSEAS Transactions on Information Science and Applications*, pp. 1-13.

Dennehy, D., & Conboy, K. (Out de 2016). Going with the flow: An activity theory analysis of flow techniques in software development. *Journal of Systems and Software, 133*, pp. 160-173. doi:10.1016/j.jss.2016.10.003z

Easterbrook, S., Singer, J., Storey, M.-A., & Damian, D. (01 de 2008). Selecting Empirical Methods for Software Engineering Research. *Guide to advanced empirical software engineering*, pp. 285-311. doi:10.1007/978-1-84800-044-5_11

Fitzgerald, B., Musiał, M., & Stol, K. (2014). Evidence-based decision making in lean software project management. *36TH INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING COMPANION, . , [s.n.], p.DOI:* (pp. 93-102). Hyderabad: Association for Computing Machinery. doi:10.1145/2591062.2591190

Greaves, K. (2011). Taming the customer support queue: a Kanban experience report. *IEEE AGILE CONFERENCE* (pp. 154-160). Salt Lake City: IEEE. doi:10.1109/AGILE.2011.9

Greening, D. R. (2010). Enterprise Scrum: Scaling Scrum to the Executive Level. *2010 43rd Hawaii International Conference on System Sciences* (pp. 1-10). Honolulu: IEEE. doi:10.1109/HICSS.2010.186

Hammarberg, M., & Sundén, J. (2014). *Kanban in Action.* Manning Publications.

Harzl, A. (2016). Combining FOSS and Kanban: An Action Research. *IFIP International Conference on Open Source Systems* (pp. 71-84). Springer, Cham. doi:10.1007/978-3-319-39225-7_6

Heikkilä, V., Paasivaara, M., & Lassenius, C. (2016). Teaching University Students Kanban with a Collaborative Board Game. *2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C)* (pp. 471-480). Austin, TX: IEEE.

Highsmith, J., & Cockburn, A. (2001). Agile software development: The business of innovation. *Computer, 34*, pp. 120-127.

Hiranabe, K. (14 de 01 de 2008). *Kanban applied to software development: From agile to lean.* Acesso em 14 de set de 2022, disponível em InfoQ: https://www.infoq.com/articles/hiranabe-lean-agile-kanban/#theCommentsSection

Ikonen, M., Pirinen, E., Fagerholm, F., Kettunen, P., & Abrahamsson, P. (2011). On the Impact of Kanban on Software Project Work: An Empirical Case Study Investigation. *2011 16th IEEE International Conference on Engineering of Complex Computer Systems* (pp. 305-314). Las Vegas, NV, USA: IEEE. doi:10.1109/ICECCS.2011.37

Laanti, M., & Kangas, M. (2015). Is agile portfolio management following the principles of large-scale agile? Case study in Finnish Broadcasting Company Yle. *IEEE Agile Conference* (pp. 92-96). National Harbor, MD, USA: IEEE. doi:10.1109/Agile.2015.9

Law, E. L., & Lárusdóttir, M. K. (02 de Set de 2015). Whose Experience Do We Care About? Analysis of the Fitness of Scrum and Kanban to User Experience. *International Journal of Human–Computer Interaction, 31*(9), pp. 584-602. doi:10.1080/10447318.2015.1065693

Leopold, K. (2018). *Practical Kanban: From Team Focus to Creating Value.* LEANability PRESS. Fonte: ISBN: 978-3-903205-05-5

Lethbridge, T. C., Sim, S. E., & Singer, J. (01 de 07 de 2005). Studying Software Engineers: Data Collection Techniques for Software Field Studies. *Empirical Software Engineering, 10*(3), pp. 311-341. doi:10.1007/s10664-005-1290-x

Maassen, O., & Sonnevelt, J. (2010). Kanban at an Insurance Company (Are You Sure?). *In: International Conference on Agile Software Development. Agile Processes in Software Engineering and Extreme Programming. XP 2010.*

Lecture Notes in Business Information Processing. 48, pp. 297-306. Berlin, Heidelberg: Springer Berlin Heidelberg. doi:10.1007/978-3-642-13054-0_32

Mahnič, V. (01 de Jan de 2015). From Scrum to Kanban : introducing lean principles to a software engineering capstone course. *International journal of engineering education, 31*(4), pp. 1106-1116.

Middleton, P., & Joyce, D. (fev de 2012). Lean Software Management: BBC Worldwide Case Study. *IEEE Transactions on Engineering Management, 59*(1), pp. 20-32. doi:10.1109/TEM.2010.2081675

Muniz, A., Irigoyen, A., Mafra, C., Trierveiler, F., & Villanova, G. (2021). *Jornada Kanban na prática: unindo teoria e prática com o objetivo de acelerar o aprendizado do Kanban para quem está iniciando.* Rio de Janeiro: Brasport. Fonte: ISBN: 978-65-88431-18-4

Neely, S., & Stolt, S. (2013). Continuous Delivery? Easy! Just Change Everything (Well, Maybe It Is Not That Easy). *2013 Agile Conference* (pp. 121-128). Nashville, TN, USA: IEEE. doi:10.1109/AGILE.2013.17

Nevenka, K., & Saso, K. (01 de 2015). Usage of Kanban Methodology at Software Development Teams. *Journal of applied economics and business, 3*(3), pp. 25-34.

Nikitina, N., & Kajko-Mattsson, M. (2011). Developer-driven big-bang process transition from Scrum to Kanban. *In: Proceedings of the 2011 International Conference on Software and Systems Process (ICSSP '11)* (pp. 159-168). Honolulu, Wiley: Association for Computing Machinery. doi:10.1145/1987875.1987901

Ohno, T. (1988). *Toyota Production System: Beyond Large-Scale Production* (1st ed.). New York: Productivity Press. doi:10.4324/9780429273018

Ozkan, N., Bal, S., Erdogan, T. G., & Gok, M. S. (2022). Scrum, Kanban or a Mix of Both?A Systematic Literature Review. *17th CONFERENCE ON COMPUTER SCIENCE AND INTELLIGENCE SYSTEMS* (pp. 881-891). Sofia, Bulgaria: FEDCSIS. doi:10.15439/2022F143

Poppendieck, M., & Poppendieck, T. (2003). *Lean Software Development: An Agile Toolkit.* Upper Saddle River, NJ: Addison Wesley.

Raunak, M. S., & Binkley, D. (09 de 2017). Agile and other trends in software engineering. *2017 IEEE 28th Annual Software Technology Conference (STC)* (pp. 1-7). Gaithersburg, MD, USA: IEEE. doi:10.1109/STC.2017.8234457

Rodríguez, P., Markkula, J., Oivo, M., & Turula, K. (2012). Survey on agile and lean usage in finnish software industry. *In: Proceedings of the ACM-IEEE international symposium on Empirical software engineering and measurement (ESEM '12)* (pp. 139–148). Lund, Sweden: Association for Computing Machinery. doi:10.1145/2372251.2372275

Rodríguez, P., Partanen, J., Kuvaja, P., & Oivo, M. (2014). Combining Lean Thinking and Agile Methods for Software Development: A Case Study of a Finnish Provider of Wireless Embedded Systems Detailed. *2014 47th Hawaii International Conference on System Sciences* (pp. 4770-4779). Waikoloa, HI, USA: IEEE. doi:10.1109/HICSS.2014.586

Runeson, P., & Höst, M. (19 de dez de 2008). Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering, 14*(2), pp. 131-164. doi:10.1007/s10664-008-9102-8

Rutherford, K., Shannon, P., Judson, C., & Kidd, N. (2010). From Chaos to Kanban, via Scrum. *Agile Processes in Software Engineering and Extreme Programming* (pp. 344-352). Berlin: Springer.

Salma, F., & Gómez, J. M. (01 de jan de 2021). Challenges and Trends of Agile. *In M. Mora, J. Gómez, R. O'Connor, & A. Buchalcevová (Ed.), Balancing Agile and Disciplined Engineering and Management Approaches for IT Services and Software Products*, pp. 189-204. doi:10.4018/978-1-7998-4165-4.ch010

Santos, P. S., Beltrão, A. C., de Souza, B. P., & Travassos, G. H. (19 de out de 2018). On the benefits and challenges of using kanban in software engineering: a structured synthesis study. *Journal of Software Engineering Research and Development, 13*(6), pp. 1-29. doi:10.1186/s40411-018-0057-1

Schwaber, K. (2004). *Agile Project Management with Scrum.* Redmond, WA: Microsoft Press.

Schwaber, K., & Sutherland, J. (2020). *The scrum guide*. Acesso em 28 de maio de 2022, disponível em Scrum Alliance: https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-US.pdf

Seikola, M., Loisa, H.-M., & Jagos, A. (2011). Kanban Implementation in a Telecom Product Maintenance. *In: 2011 37th EUROMICRO Conference on Software Engineering and Advanced Applications, Proceedings 37th EUROMICRO Conference on Software Engineering and Advanced Applications* (pp. 321-329). Oulu, Finland: IEEE. doi:10.1109/SEAA.2011.56

Senapathi, M., & Drury-Grogan, M. (05 de 10 de 2021). Systems Thinking Approach to Implementing Kanban: A Case Study. *Journal of Software: Evolution and Process*, p. 33:e2322. doi:10.1002/smr.2322

Senapathi, M., Middleton, P., & Evans, G. (2011). Factors Affecting Effectiveness of Agile Usage – Insights from the BBC Worldwide Case Study. *International Conference on Agile Software Development. 77*, pp. 132-145. Berlin: Springer, Berlin, Heidelberg. doi:10.1007/978-3-642-20677-1_10

Shinkle, C. M. (2009). Applying the Dreyfus Model of Skill Acquisition to the Adoption of Kanban Systems at Software Engineering Professionals (SEP). *2009 Agile Conference* (pp. 186-191). Chicago, IL, USA: IEEE. doi:10.1109/AGILE.2009.25

Siderovza, S. (2018). *Maximise Customer Satisfaction: Kanban Cycle Time*. Acesso em 21 de abril de 2020, disponível em Nave Blog - Expert tips and guidelines for Kanban teams: https://getnave.com/blog/kanban-cycle-time/

Sjøberg, D. I., Johnsen, A., & Solberg, J. (03 de jul de 2012). Quantifying the Effect of Using Kanban versus Scrum: A Case Study. *IEEE Software, 29*(5), pp. 47-53. doi:10.1109/MS.2012.110

Taipale, M. (2010). Huitale – A Story of a Finnish Lean Startup. *Lean Enterprise Software and Systems LESS 2010. Lecture Notes in Business Information Processing. 65*, pp. 111-114. Berlin, Heidelberg: Springer. doi:10.1007/978-3-642-16416-3_16

Tanner, M., & Dauane, M. E. (01 de jan de 2017). The use of Kanban to alleviate collaboration and communication challenges of global software development. *Issues in Informing Science and Information Technology, 14*, pp. 177-197. doi:10.28945/3716

Tripathi, N., Rodríguez, P., Ahmad, M. O., & Oivo, M. (2015). Scaling Kanban for Software Development in a Multisite Organization: Challenges and Potential Solutions. *International Conference on Agile Software Development. 212*, pp. 178-190. Springer, Cham. doi:10.1007/978-3-319-18612-2_15

Vacanti, D. S. (2015). *Actionable Agile Metrics for Predictability: An Introduction.* Actionable Agile Press.

Wan, H.-d., & Chen, F. F. (20 de 07 de 2008). A Web-based Kanban system for job dispatching, tracking, and performance monitoring. *The International Journal of*

Advanced Manufacturing Technology, 38(9), pp. 995–1005. doi:10.1007/s00170-007-1145-2

Wang, X., Conboy, K., & Cawley, O. (Jun de 2012). An experience report analysis of the application of lean approaches in agile software development. *Journal of Systems and Software. Galway, 85*(6), pp. 1287-1299. doi:10.1016/j.jss.2012.01.061

Wangenheim, C. G., Savi, R., & Borgatto, A. F. (01 de out de 2013). SCRUMIA—An educational game for teaching SCRUM in computing courses. *Journal of Systems and Software, 86*(10), pp. 2675-2687. doi:10.1016/j.jss.2013.05.030

Willeke, E. R. (2009). The Inkubook Experience: A Tale of Five Processes. *2009 Agile Conference* (pp. 156-161). Chicago, IL, USA: IEEE. doi:10.1109/AGILE.2009.34

Womack, J., Jones, D., & ROOS, D. (1990). *The Machine That Changed the World.* New York: Scribner Book Company.

Yin, R. K. (2017). *Case Study Research and Applications: Design and Methods* (6th ed.). Thousand Oaks: Sage Publications.

Glossário

Asana: Ferramenta web proprietária de gestão de projetos e tarefas.

Branch: Estrutura no *Git* de onde se desenvolve uma tarefa de maneira isolada sem afetar o restante do sistema.

BitBucket: plataforma de gerenciamento de desenvolvimento de software que oferece recursos de hospedagem de repositório Git ou Mercurial.

Bug: Defeito encontrado em um software entregue ao cliente.

Feature: Funcionalidade.

GIT: Sistema distribuído para controle de versionamento de código-fonte de software desenvolvido para facilitar o gerenciamento de código fonte, principalmente, em equipes de desenvolvimento.

GitHub: plataforma de desenvolvimento de software que permite o controle de versão e colaboração em projetos através do uso do Git.

GitLab: plataforma de gerenciamento de ciclo de vida de desenvolvimento de software.

Hot-Fixes: Correções urgentes. Aplicadas em produção.

JIRA: Ferramenta web proprietária para controle de atividades em projetos de desenvolvimento de software.

Kaizen: Palavra japonesa que significa melhoria.

Kanban: Quadro físico marcados com faixas verticais que representam etapas dentro de um fluxo de atividades ou processo. Possui cartões autoadesivos que representam as tarefas que percorrem o fluxo da direita para a esquerda.

Kanban: Método proposto por David J. Anderson, adaptado do Lean que propõe o uso de kanban para auxiliar o desenvolvimento de software aderindo-se ao processo de desenvolvimento existente, expondo os gargalos e permitindo a aplicação de *kaizens* para melhoria contínua.

Lead Time: Tempo total de execução de uma tarefa em um fluxo kanban.

Lean: Conjunto de práticas de gestão com foco em melhoria da eficiência e eficácia através da redução do desperdício.

Lean Thinking: Processo de decisão de negócios baseado no Lean.

Merge: Processo de fundir um código que representa uma tarefa no código principal do sistema (processo do Git).

Pair-programming: Programação em pares.

Pull-request: Processo do Git que consiste na solicitação de uma revisão e aprovação de merge de desenvolvimento de uma tarefa no código principal.

Merge-request: O mesmo que pull-request. É utilizado por algumas ferramentas Git.

Scrum: Método de desenvolvimento de software.

Swarming: Técnica que envolve designar mais de um desenvolvedor para atacar um mesmo problema de modo a buscar uma solução mais rápida.

Throughput: Taxa de entregas de um fluxo Kanban.

Trello: Ferramenta web proprietária de gerenciamento de projetos que permite organizar tarefas e projetos visualmente em um formato de lista de cartões.

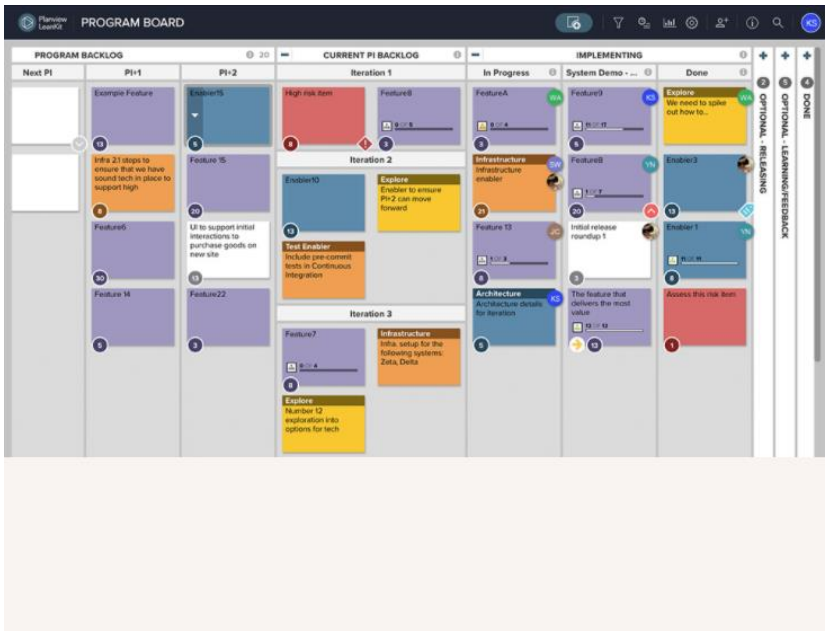
9 APÊNDICE A – Ferramentas de digitalização do Kanban

Há diversas ferramentas que podem ser utilizadas como quadros virtuais de Kanban no mercado, algumas pagas, outras gratuitas. Algumas são aplicações, enquanto outras são *plugins* de outras aplicações. A seguir, apresentam-se algumas aplicações sugeridas e atualizadas a partir das contribuições de Hammarberg e Sunden (2014). Cumpre também destacar que o mercado é muito dinâmico e estas indicações podem ficar brevemente defasadas, sendo por isso mesmo, recomendável pesquisar as ferramentas de mercado disponíveis no momento oportuno de implementação.

A.1 Leankit Kanban

Trata-se de uma ferramenta leve, que suporta diversas práticas do Kanban e é indicado tanto para simples utilização pessoal, quanto para organizações. A versão gratuita suporta até 25 usuários e 10 quadros. Disponível em: https://www.planview.com/products-solutions/products/leankit/

Figura A1 – Demonstração do Leankit Kanban

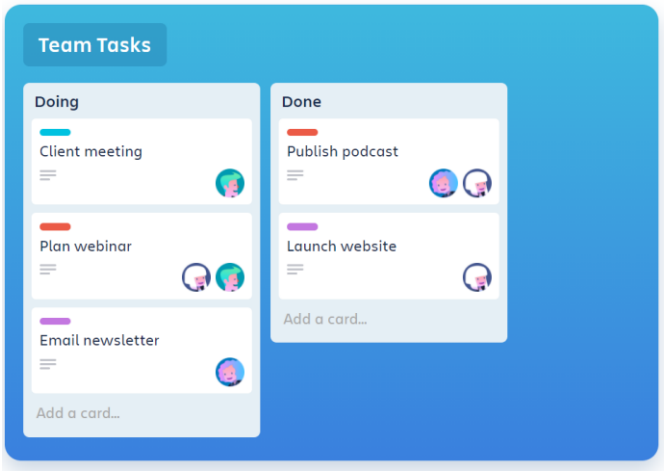


Fonte: captura de tela obtida pelo autor

A.2 Trello

Ferramenta completamente gratuita, que pode ser usada para visualizar o fluxo, apesar de algumas limitações como não permitir limitar o WIP. Disponível em: https://trello.com/

Figura A2 – Demonstração do Trello

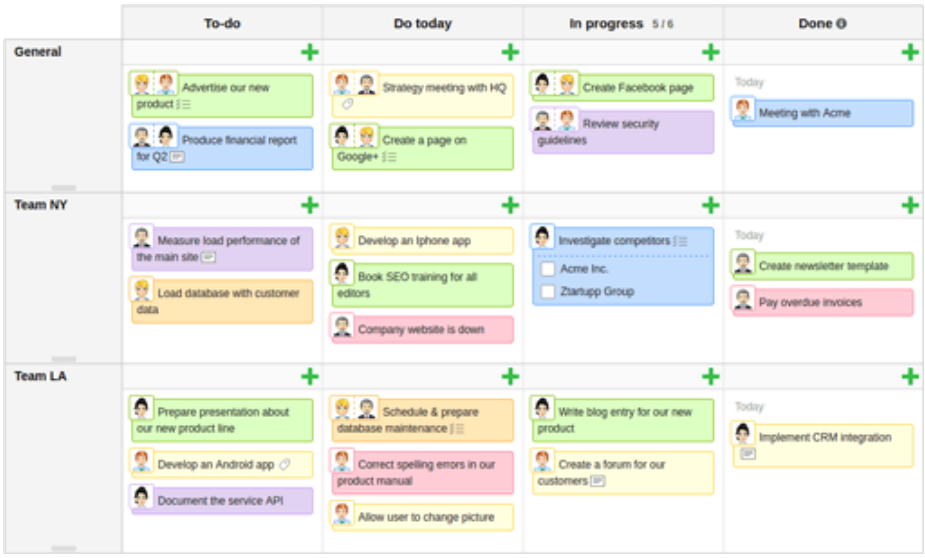


Fonte: captura de tela obtida pelo autor

A.3 KanbanFlow

Também uma ferramenta totalmente gratuita, pode ser usada para visualizar o fluxo e suporta muitas práticas do Kanban, como *timers* Pomodoro, por exemplo. Disponível em: https://kanbanflow.com

Figura A3 – Demonstração do KanbanFlow

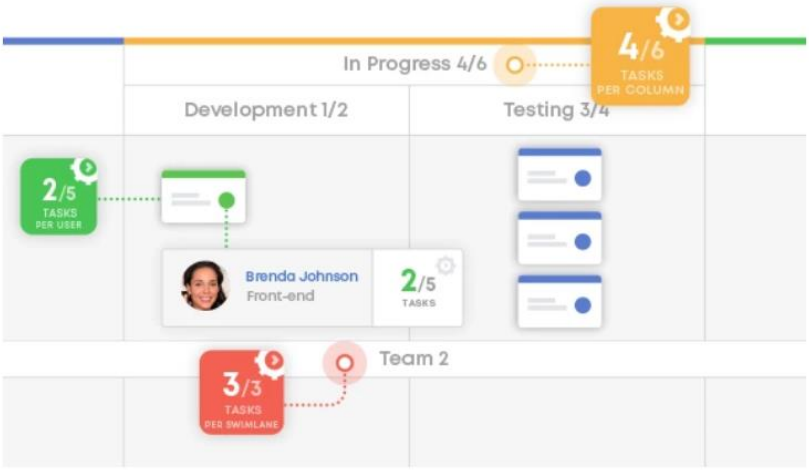


Fonte: captura de tela obtida pelo autor

A.4 Kanbanize

Esta aplicação em nuvem oferece uma vasta gama de funcionalidades, incluindo 2FA e relatórios de análise aprimorados. Não há versões gratuitas disponíveis, entretanto. Disponível em: https://kanbanize.com

Figura A4 – Demonstração do Kanbanize

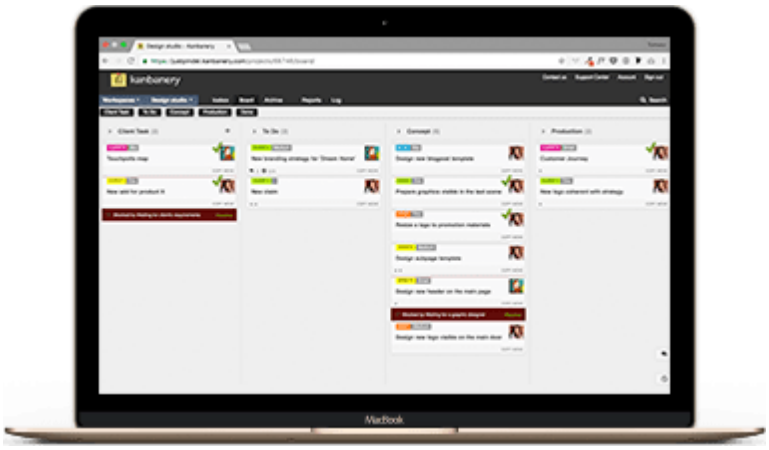


Fonte: captura de tela obtida pelo autor

A.5 Kanbanery

Esta aplicação em nuvem oferece integração com GitHub, visualização em *smartphones* e relatórios de análise aprimorados. Não há versões gratuitas disponíveis. https://kanbanery.com

Figura A5 – Demonstração do Kanbanery



Fonte: kanbanery.com