UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO DE JOINVILLE
CURSO DE ENGENHARIA DE TRANSPORTES E LOGÍSTICA

RODRIGO PALUDETTO SILVA DE PAULA LOPES

MODELO DE GESTÃO ÁGIL DE PROJETO BASEADA NA PRÁTICA DE PROGRAMAÇÃO EM PARES PARA DESENVOLVIMENTO DE SOFTWARE

JOINVILLE
2021

RODRIGO PALUDETTO SILVA DE PAULA LOPES

MODELO DE GESTÃO ÁGIL DE PROJETO BASEADA NA PRÁTICA DE PROGRAMAÇÃO EM PARES PARA DESENVOLVIMENTO DE SOFTWARE

Trabalho apresentado como requisito para obtenção do título de bacharel no Curso de Graduação em Engenharia de Transportes e Logística do Centro Tecnológico de Joinville da Universidade Federal de Santa Catarina.

Orientador: Dr. Cristiano Vasconcellos Ferreira

Joinville

2021

RODRIGO PALUDETTO SILVA DE PAULA LOPES

MODELO DE GESTÃO ÁGIL DE PROJETO BASEADA NA PRÁTICA DE PROGRAMAÇÃO EM PARES PARA DESENVOLVIMENTO DE SOFTWARE

Este Trabalho de Conclusão de Curso foi julgado adequado para obtenção do título de bacharel em Engenharia Transportes e Logística, na Universidade Federal de Santa Catarina, Centro Tecnológico de Joinville.

Joinville, 05 de maio de 2021.

Banca Examinadora:

_____

Dr.

Orientador

Cristiano Vasconcellos Ferreira

_____

Dra.

Elisete Santos da Silva Zagheni

_____

Dr.

Modesto Hurdado Ferrer

AGRADECIMENTOS

Agradeço primeiramente aos meus pais, que não mediram esforços para que eu conseguisse concluir essa etapa da vida.

A todos meus familiares, por me darem suporte em momentos difíceis que envolveram desistência e retomada da graduação. Obrigado vocês foram fundamentais para mais essa conquista.

À minha namorada, Larissa Bagini, por me fazer a cada dia ser uma pessoa melhor, e estar sempre ao me lado, fonte de inspiração e motivação, estando presente comigo nos momentos mais difíceis da graduação, me apoiando e não me deixando desanimar.

A todo corpo docente, diretores e servidores que souberam repassar o conhecimento para que minha formação fosse a melhor possível. Agradeço em especial a meu orientador e coordenador de curso professor Dr. Eng. Cristiano Vasconcellos Ferreira que me mostrou a definição do que é ser um orientador durante este ano que me auxiliou neste trabalho.

Agradeço a empresa Rota Exata Software Ltda, por confiar em meu trabalho e dando a viabilidade de pôr em prática o trabalho aqui apresentado. Meu muito obrigado a todos envolvidos.

Por fim agradeço meus amigos que tornaram a minha passagem pela universidade repleta de boas histórias que levarei pra vida. Vocês são parte integrante de tudo isso.

RESUMO

Em um ambiente empresarial cada vez mais competitivo as empresas de software necessitam ser cada vez mais eficientes para se manterem no mercado. A utilização de metodologias ágeis vem se mostrando uma ótima alternativa para processos de desenvolvimento de software reduzindo o tempo de entrega de novas funcionalidades sem perder a qualidade do produto. Este trabalho tem seu foco na indústria de software, e propõe a implementação de uma melhoria no processo de desenvolvimento ágil da empresa Rota Exata Software Ltda, baseado na Programação em Pares. O trabalho apresenta, inicialmente, uma revisão bibliográfica e um estudo de caso na empresa de software. E, na sequência, com base nestes estudos, é implementado a Programação em Pares. Ao final, são apresentados os resultados da implementação do modelo proposto, e um questionário de avaliação da experiência dos desenvolvedores participantes da implementação.

Palavras-chave: Programação em Pares. Métodos Ágeis. Gerenciamento de Projeto. Empresas de software.

ABSTRACT

In an increasingly competitive business environment, software companies need to be increasingly efficient to stay in the market. The use of agile methodologies is proving to be an excellent alternative for software development processes, reducing the delivery time of new functionalities without losing the quality of the product. This work focuses on the software industry and proposes an improvement in the agile development process of Rota Exata Software Ltda, based on Pair Programming. The work initially presents a bibliographic review and a case study at the software company. Then, based on these studies, the Pair Programming method is implemented. In the end, the implementation of this proposed model is presented, and a questionnaire evaluating the developer's experience participating in this study.

Keywords: Pair Programming. Agile methods. Project Management. Software companies

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

IC – Integração Contínua

PO – Product Owner

SaaS – Software as a Service

TI – Tecnologia de Informação

UFSC – Universidade Federal de Santa Catarina

WIP – Work in progress

XP – eXtreaming Program

SUMÁRIO

1 INTRODUÇÃO

Com o avanço da internet e as novas tecnologias de armazenamento de dados, entramos na era da transformação digital. A indústria de software está evoluindo tão rapidamente que pode ser encontrada em vários segmentos e em níveis diferentes. Os exemplos vão desde softwares simples como controlar a temperatura do ambiente pelo celular até softwares mais robustos que tomam decisões de investimentos, baseado em análises de base de dados com proporções astronômicas.

Neste contexto vem se desenvolvendo rapidamente um modelo de comercialização de software, conhecido como Software como Serviço ou Software as a Service (SaaS). O software consiste em uma funcionalidade ou aplicação que é oferecida por meio de um modelo de assinatura pela Internet. O cliente que pode ser tanto pessoa física quanto empresa não se tornam donos do software, ao invés disso, eles alugam a solução total que é oferecida remotamente.

A crescente competitividade empresarial e o aumento das exigências do mercado colocaram o mercado SaaS em destaque. Segundo Pressman (2011) embora a indústria caminhe para a construção com base em componentes, a maioria dos softwares continuam a ser construídos de forma personalizada (sob encomenda).

Para suprir a demanda e exigências do mercado as empresas desenvolvedoras de software tem o desafio de criar produtos que se adaptem ao mercado e suas mudanças constantes. Neste cenário os processos e metodologias para gerenciamento e desenvolvimento de produtos precisam acompanhar as mudanças repentinas e ao mesmo tempo atingir o maior número de clientes.

O objetivo do gerenciamento de projetos é produzir um produto completo que atenda as demandas do cliente. Em muitos casos, o objetivo é também moldar ou reformar o resumo do cliente para poder, de maneira viável, abordar suas principais exigências. Uma vez que os objetivos do cliente estejam claramente estabelecidos, eles devem influenciar todas as decisões tomadas na concepção projeto. (FARIA, 2009).

Por tratar-se de produtos complexos e de melhoria contínua, as abordagens tradicionais de desenvolvimento de produto não acompanharam a evolução do mercado de desenvolvimento de software. Em 2001 um grupo de profissionais experientes do ramo de produtos de software se reuniram para criar uma nova forma de gerenciar e desenvolver

projetos. Foi definido um conjunto de valores e princípios, conhecido como Manifesto Ágil. E desde então este manifesto vem sendo seguido por diversas organizações em todo mundo.

Os métodos ágeis são modelos de gerenciamento de projetos com raízes na TI (Tecnologia da Informação). Refere-se ao desenvolvimento ágil de software, estudando um conjunto de comportamentos, práticas e processos de ferramentas que são utilizadas para a criação de produtos. Este, está associado a uma mentalidade produtiva e está focado em segmentar o escopo de tarefas para antecipar entregas, gerando uma percepção mais rápida do valor do cliente. O Scrum, Kanban e XP (Extreme Programming), são os métodos ágeis mais populares usados para desenvolver produtos e sistemas complexos (NETO, 2014).

A empresa Rota Exata Software Ltda, está inserida no contexto apresentado. Atualmente a empresa está lançando novos produtos e aumentando sua base de clientes rapidamente. Para atender às novas demandas a empresa está evoluindo seu modelo de desenvolvimento de produtos, baseando-se nos métodos ágeis, como Scrum, Kanban e XP. O presente trabalho pretende contribuir com uma proposta do modelo de desenvolvimento de software da empresa descrevendo seus processos e implementando possíveis melhorias para serem testadas pelo time de desenvolvimento.

A empresa está inserida no competitivo mercado de transportes e tem o desafio de entregar novas funcionalidades a seu produto, mantendo a mesma qualidade já conhecida nesse mercado. Para isso, o time de desenvolvimento ganhou novos membros recentemente e há uma projeção de dobrar o número de funcionários no decorrer do ano. O presente trabalho buscou implementar uma prática primordial do método XP, que é a Programação em Pares.

Essa prática requer que dois programadores trabalhem juntos no mesmo código, onde ambos trabalham ao mesmo tempo e no mesmo computador, enquanto o primeiro desenvolvedor se concentra na escrita, o outro revisa o código, sugere melhorias e corrige erros ao longo do caminho (MEDEIROS, 2007). Esta técnica foi implementada com o objetivo de melhorar os indicadores do time de desenvolvimento.

1.1 OBJETIVOS

Este trabalho tem como objetivo geral propor um modelo de gestão ágil de projeto baseada na prática de Programação em Pares para desenvolvimento de software.

Considerando o objetivo geral proposto foram estabelecidos os seguintes objetivos específicos:

- Revisar bibliograficamente as abordagens clássicas e ágeis de gerenciamento de projetos;
- Aprofundar conhecimentos sobre o tema métodos ágeis, evidenciando definições, variedades e possíveis abordagens;
- Descrever o modelo da empresa estudada e as métricas utilizadas;
- Implementar possíveis melhorias no modelo de gestão de projeto de software para a empresa Rota Exata; e,
- Comparar os resultados obtidos após a implementação.

1.2 JUSTIFICATIVA

Segundo a Associação Catarinense de Tecnologia (2020), o Brasil encerrou o ano de 2019 com 306,4 mil empresas atuando no setor de tecnologia. São cerca de 7 mil a mais em comparação a 2018. Apesar do crescimento no último biênio, este número caiu 8,5% entre 2015 e 2019. Na contramão dos números nacionais destacou-se o estado de Santa Catarina sendo o estado que registrou maior crescimento (11,8%) no mesmo período chegando a 12.138 empresas de tecnologia (ACATE, 2020).

Esses números podem ser explicados devido ao ecossistema de inovação criado nas principais cidades do estado ao longo dos últimos anos. O ecossistema consiste em empresas de tecnologia e universidades de ponta. Nacionalmente, três cidades catarinenses se destacam entre as dez cidades com a maior taxa de empresas de tecnologia por habitante do país: Florianópolis, Blumenau e Joinville (ACATE, 2020).

Neste contexto, esse trabalho se justifica com o pretexto de aproximar a Academia de empresas de tecnologia, contribuindo com o ecossistema inovador da cidade de Joinville. Trazendo o conhecimento acadêmico do autor aplicando um modelo de desenvolvimento em uma empresa de base tecnológica da cidade.

Além disso, corroboram para a justificativa do trabalho, os desafios do gerenciamento ágil de projetos como: manter equipes engajadas, particionar problemas complexos em tarefas curtas e focadas no cliente e aumentar a produtividade das equipes de desenvolvimento. Este trabalho se posiciona como um veículo de conhecimento não só para o meio acadêmico, como também uma fonte de contribuição para o conhecimento em seu meio social a partir de um material coeso e estruturado, possibilitando também o entendimento do tema por leitores que não sejam especialistas sobre a temática abordada.

1.3 METODOLOGIA

Buscando tornar possível a realização dos objetivos propostos neste trabalho, considerou-se uma metodologia de trabalho de pesquisa aplicada, que tem como característica o interesse na aplicação, utilização e consequências práticas dos conhecimentos (Gil, 2010).

Os conceitos são apresentados de forma a representar a cronologia em que serão utilizados ou considerando a melhor forma para compreensão.

Inicialmente é realizada a pesquisa bibliográfica sobre os temas que construíram o referencial teórico do trabalho, a fim de dar embasamento no entendimento dos capítulos seguintes.

Em seguida, é apresentado o estudo de caso, com o objetivo de apresentar a empresa bem como seu modelo de desenvolvimento de software atual, e seus principais indicadores.

Na etapa seguinte é feito um diagnóstico do modelo atual e proposto e implementado a Programação em Pares. Na sequência foi enviado um questionário dissertativo aos participantes da aplicação, com a finalidade de compreender a experiência dos envolvidos.

Por fim, analisou-se os mesmos indicadores apresentados no estudo de caso, para comparação dos modelos. E ainda se discutiu a experiência dos desenvolvedores com o a Programação em Pares por meio das respostas do questionário.

1.4 ESTRUTURA DO DOCUMENTO

Este trabalho está organizado em 5 capítulos.

No Capítulo 1 são apresentadas uma breve introdução sobre o tema a ser discutido, os objetivos gerais e específicos, as questões que levaram à realização deste trabalho e a metodologia e estrutura escolhida para o trabalho.

O Capítulo 2 trata do referencial teórico. São apresentadas e analisadas as referências dentro do escopo deste trabalho, abordados os termos, gerenciamento de projetos e processo, metodologias ágeis e programação em pares.

Já o Capítulo 3 apresenta o estudo de caso da empresa Rota Exata Software Ltda e seu processo de desenvolvimento de software.

No Capítulo 4, é proposto, implementado e em seguida avaliado um modelo de desenvolvimento de software baseado na Programação em Pares

No Capítulo 5 são apresentadas as conclusões deste trabalho e apontamentos para trabalhos futuros.

2 REFERENCIAL TEÓRICO

Neste capítulo serão abordados os principais conceitos relacionados ao tema do trabalho. Buscando gerar embasamento teórico a respeito dos métodos ágeis de gerenciamento de projeto, serão apresentados a seguir metodologias para adequar o modelo teórico por meio de um estudo de caso e avaliá-lo. A apresentação destas metodologias terá como foco a indústria de software de alta competitividade.

2.1 GERENCIAMENTO DE PROJETOS: MODELO DE DESENVOLVIMENTO TRADICIONAL

O gerenciamento de projetos é a prática de iniciar, planejar, executar, controlar e fechar o trabalho de uma equipe para atingir metas específicas e atender a critérios específicos de sucesso no tempo especificado. Seu desafio é atingir todas as metas do projeto com as restrições dadas. Esta informação é geralmente descrita em sua documentação, criada no início do processo de desenvolvimento. As principais restrições são escopo, tempo, qualidade e orçamento. O desafio secundário - e mais ambicioso - é otimizar a alocação dos insumos necessários e aplicá-los para atender aos objetivos pré-definidos (VARGAS, 2016).

Como mencionado por Amaral (2011), objetivos de gerenciamento de projetos mal definidos são prejudiciais para a tomada de decisões. Um projeto é um empreendimento temporário projetado para gerar um produto, serviço ou resultado exclusivo com início e fim definidos (normalmente restritos no tempo, por financiamento ou contratação de pessoal) para atingir metas e objetivos específicos, geralmente para trazer mudanças benéficas ou valor acrescentado.

A natureza temporária dos projetos contrasta com as operações usuais, que são atividades funcionais repetitivas, permanentes ou semipermanentes para produzir produtos ou serviços. Na prática, o gerenciamento de tais abordagens de produção distintas requer o desenvolvimento de habilidades técnicas e estratégias de gerenciamento distintas (MENDES, 2012).

A abordagem tradicional de projetos é uma prática universal que inclui um conjunto de técnicas desenvolvidas usadas para planejar, estimar e controlar atividades. O objetivo dessas técnicas é atingir o resultado desejado dentro do prazo, do orçamento e de acordo com as

especificações. A abordagem tradicional de projetos é utilizada principalmente em projetos em que as atividades são concluídas em uma sequência e, raramente, ocorrem alterações (KERZNER, 2011).

O conceito de gerenciamento de projetos tradicional é baseado em experiência previsível e ferramentas previsíveis. Cada projeto segue o mesmo ciclo de vida, que inclui cinco estágios: iniciação, planejamento, execução, monitoramento e controle, encerramento. (KERZNER, 2011).

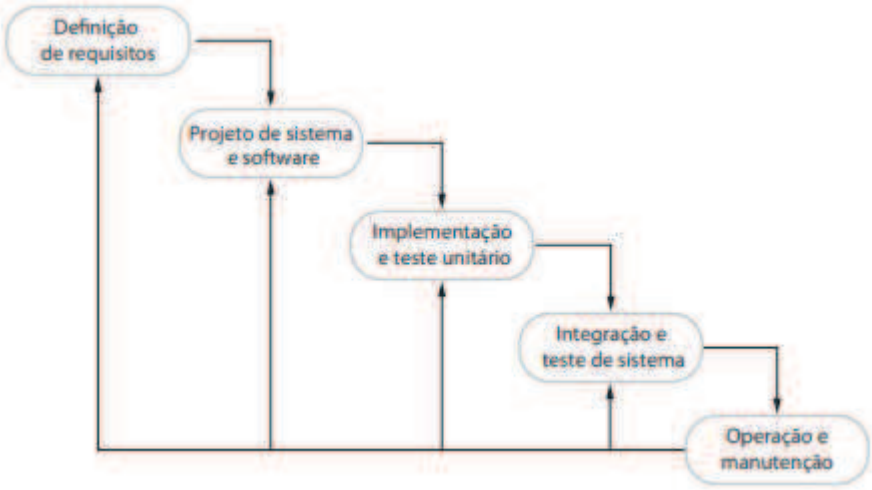
Embora o gerenciamento de projetos tenha sido introduzido pela primeira vez como disciplina nos anos 50, ele existe há milhares de anos e tem sido usado na criação de alguns dos maiores projetos, desde as Grandes Pirâmides até a Ferrovia Transcontinental. Esses projetos de larga escala mudaram a face da história e da humanidade para sempre. No entanto, com o passar do tempo, empresários e empreendedores achavam difícil acompanhar o ritmo acelerado do desenvolvimento tecnológico e as crescentes demandas do mercado (PETERS, 2011).

Os processos têm se tornado mais complexos e exigentes, e o gerenciamento tradicional de projetos não oferece mais as melhores soluções para os problemas de negócios. O conceito de gerenciamento de projetos tradicional foi alterado e estendido por diferentes metodologias e estruturas de gerenciamento de projetos. No entanto, o gerenciamento tradicional ainda é considerado a base de todas as abordagens modernas e ainda é a metodologia predominante ao trabalhar em grandes projetos de construção (LOOSEMORE, M.; RAFTERY, J.; REILLY, C.; HIGGON, D., 2011).

Para manter o controle sobre o projeto do início ao fim, um gerente de projetos utiliza várias técnicas, dentre as quais se destacam: planejamento de projeto, análise de valor agrafado, gerenciamento de riscos de projeto, cronograma, melhoria de processo.

As metodologias tradicionais de desenvolvimento de software são baseadas em fases/estágios previamente organizados do ciclo de vida de desenvolvimento de software. Aqui, o fluxo de desenvolvimento é unidirecional, dos requisitos ao design e, em seguida, ao desenvolvimento, aos testes e à manutenção. Em abordagens clássicas como o modelo em cascata, cada fase possui resultados específicos e documentação detalhada que foram submetidos a um processo de revisão completo (SOMMERVILLE, 2011). Modelo em cascata é apresentado na Figura 1.

Figura 1: Modelo em Cascata



Fonte: SOMMERVILLE (2011)

As abordagens tradicionais são adequadas quando os requisitos são bem compreendidos - por exemplo, em indústrias como a construção, onde todos compreendem claramente o produto. Por outro lado, em indústrias que mudam rapidamente, como a TI, os procedimentos tradicionais de desenvolvimento podem falhar em atingir as metas do projeto. Abaixo estão as principais desvantagens dos métodos tradicionais (SOMMERVILLE, 2011).

- A declaração do problema / necessidade comercial deve ser definida com bastante antecedência. A solução também precisa ser determinada com antecedência e não pode ser alterada ou modificada;
- Todo o conjunto de requisitos deve ser fornecido na fase inicial sem chance de alterá-los ou modificá-los após o início do desenvolvimento do projeto. Por exemplo, o usuário pode ter fornecido requisitos iniciais para analisar seus produtos em termos de vendas. Após o início do projeto, se o usuário desejar alterar o requisito e analisar os dados sobre o movimento regional de produtos, o usuário poderá esperar até a conclusão dos requisitos iniciais ou iniciar outro projeto (BLASCHEK, 2011);
- O usuário não pode realizar avaliações intermediárias para garantir que o desenvolvimento do produto esteja alinhado para que o produto final atenda aos requisitos de negócios;
- O usuário obtém um sistema baseado no entendimento do desenvolvedor e isso nem sempre pode atender às necessidades do cliente;
- A documentação assume alta prioridade e se torna cara e demorada para criar.
- Há menos chances de criar / implementar componentes reutilizáveis;

- Os testes podem começar somente após o término do processo de desenvolvimento. Quando o aplicativo está no estágio de teste, não é possível voltar e editar nada que possa ter um impacto adverso nas datas de entrega e nos custos do projeto;
- Ocasionalmente, os projetos são descartados, o que leva à impressão de ineficiência e resulta em esforço e gasto desperdiçados.

Essas desvantagens dificultam a entrega do projeto em termos de custo, esforço, tempo e acabam tendo um grande impacto no relacionamento com os clientes.

As metodologias tradicionais de desenvolvimento são adequadas apenas quando os requisitos são precisos, ou seja, quando o cliente sabe exatamente o que deseja e pode afirmar com segurança que não haverá grandes mudanças no escopo durante o desenvolvimento do projeto. Não é adequado para grandes projetos, como projetos de manutenção, em que os requisitos são moderados e há uma grande margem para modificações contínuas (MORAES, 2011). A figura 2 ilustra os esforços da metodologia tradicional para se manter no plano pré-determinado.

Figura 2: Objetivo da gestão de projeto - abordagem tradicional.

Fonte: AMARAL et al. (2011)

2.2 O MÉTODO ÁGIL

A metodologia ágil é um tipo de processo de gerenciamento de projetos, usado principalmente para o desenvolvimento de software, em que demandas e soluções evoluem através do esforço colaborativo de equipes auto-organizáveis, multifuncionais e seus clientes. Esse método ajuda as equipes a responder à imprevisibilidade da construção de software. Ele usa sequências de trabalho incrementais e iterativas, comumente conhecidas como sprints (LEITÃO, 2010).

Partindo dos valores e princípios do Manifesto Ágil, o método ágil foi criado como uma resposta às inadequações dos métodos tradicionais de desenvolvimento, como o método em cascata. A indústria de software é um mercado altamente competitivo devido ao fato de que o software é algo que pode ser atualizado continuamente. Isso significa que os desenvolvedores precisam melhorar e inovar constantemente seus produtos para se manterem no topo competitivos e a abordagem linear e sequencial do método em cascata simplesmente não foi suficiente.

2.2.1 Breve História

Muitas das ideias ágeis surgiram na década de 1970. Estudos e revisões foram realizados sobre o Método Ágil, que explica seu surgimento como uma reação às abordagens tradicionais ao desenvolvimento de projetos. Em 1970, o Dr. William Royce publicou um artigo que discutia o gerenciamento e o desenvolvimento de grandes sistemas de software. O artigo delineou suas ideias específicas sobre desenvolvimento sequencial. Sua apresentação afirmou que um projeto poderia ser desenvolvido como um produto em uma linha de montagem. Cada fase do desenvolvimento precisava ser concluída antes que a próxima fase pudesse começar.

A ideia exigia que todos os desenvolvedores primeiro reunissem todos os requisitos de um projeto. O próximo passo foi concluir toda a sua arquitetura e design. Isto é seguido escrevendo o código. As sequências continuam em incrementos completos. Quando essas etapas são concluídas, há pouco ou nenhum contato entre grupos especializados que concluem cada fase do projeto. Os pioneiros do Método Ágil acreditavam que, se os desenvolvedores estudassem o processo, considerariam a solução mais lógica e útil para o desenvolvimento de software (JUBILEU, 2008).

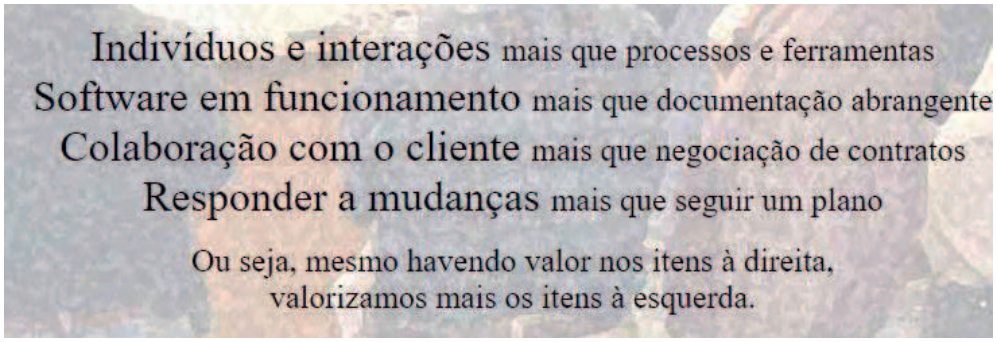
Nos anos 90, o desenvolvimento de software enfrentou um pouco de crise. Referida como 'crise de desenvolvimento de aplicativos' ou 'atraso na entrega de aplicativos', a indústria percebeu que não podia se mover rápido o suficiente para atender às demandas e exigências dos clientes - o tempo estimado entre uma necessidade de negócios e a aplicação real era de cerca de três anos. Veja, os modelos tradicionais de desenvolvimento foram baseados em uma abordagem de linha do tempo, onde o desenvolvimento ocorreu sequencialmente e o produto final não foi revelado aos clientes até a etapa final. Isso deixou pouco espaço para flexibilidade quando se tratava de revisões e alterações em andamento. Portanto, quando uma aplicação real

foi concluída, era altamente provável que os requisitos e sistemas dos objetivos originais do projeto tivessem mudado.

Com o dinheiro e os esforços desperdiçados, e até mesmo alguns projetos cancelados no meio do caminho, os líderes profissionais da comunidade de software pensaram que era hora de uma abordagem nova e atualizada. Então, em 2001, em uma cabana de esqui em Utah (EUA), reuniu 13 indivíduos. Alguns dos quais já estavam pensando na ideia de um novo método de desenvolvimento de software. Todos ansiavam por consolidar um processo que legitimasse o que estava sendo praticado e, assim, veio a criação do Manifesto Ágil (LANDIM, 2012).

O Manifesto Ágil é uma declaração dos valores e princípios expressos nos métodos ágeis. Composto por quatro valores fundamentais e 12 princípios-chave, ele visa ajudar a descobrir melhores maneiras de desenvolver softwares, fornecendo uma estrutura clara e mensurável que promova o desenvolvimento iterativo, a colaboração da equipe e o reconhecimento de alterações (ALBINO, 2013). Na Figura 3 - Valores do Manifesto Ágil estão descritos os valores do manifesto ágil:

Figura 3: Valores do Manifesto Ágil



FONTE: agile (2001)

As metodologias ágeis aplicam uma coleção de práticas, guiadas por 12 princípios-chave que podem ser aplicados por profissionais de software no dia a dia. Tais princípios podem ser observados na figura 4 - Princípios do Manifesto Ágil.

Figura 4: Princípios do Manifesto Ágil.

FONTE: agile (2001)

Métodos ágeis são algumas vezes caracterizados como o oposto de metodologias guiadas pelo planejamento ou disciplinadas. Uma distinção mais acurada é dizer que os métodos existem em um contínuo do adaptativo até o preditivo. Métodos ágeis existem do lado adaptativo deste contínuo (COSTA; PENTEADO; SILVA; BRAGA, 2015).

O método ágil opera iterativamente em contraste com a rigidez do método em cascata. O desenvolvimento de software ocorre em ciclos de entregas recorrentes de tarefas criadas a partir de um planejamento inicial. O conjunto de tarefas a ser entregues neste ciclo é denominado sprint, as entregas estabelecidas de cada sprint são usadas para informar as entregas do próximo sprint. Os fluxos de trabalho ágeis enfatizam o excesso de planejamento e as equipes coordenadas e independentes para evitar paralisia. A vantagem do método ágil é que seus negócios são investidos diretamente e envolvidos no processo de desenvolvimento de software. Isso estabelece um senso de propriedade sobre o projeto e oferece a oportunidade de feedback à medida que o desenvolvimento avança (SOMMERVILLE, 2011).

O método ágil também é mais adaptável à implementação de recursos básicos de automação e ao preenchimento de funções avançadas posteriormente, o que permite uma rápida recuperação do projeto. A desvantagem da abordagem ágil é que esse estilo de desenvolvimento de automação requer recursos humanos significativos da sua empresa - vários membros da sua equipe devem ser totalmente dedicados ao projeto. Além disso, para implementações de automação particularmente complexas, é possível perder de vista a entrega final em larga escala, pois os avanços graduais são iterativamente reunidos.

O método ágil apresenta diversas ferramentas de gerenciamento de projetos para o desenvolvimento de software, como ‘Crystal’, ‘Dynamic Systems Development’, ‘eXtreme Programming’, ‘Feature Driven Development’, 'Kanban', 'Lean Software Development' e o ‘Scrum. Neste trabalho serão abordadas somente três dessas metodologias e, que estão relacionadas ao escopo de estudo. São elas: Scrum, Kanban e Extreme Programming. E ainda será evidenciado a Programação em Pares para compor o modelo de desenvolvimento aplicado.

2.3 SCRUM

Para Schwaber e Sutherland (2013) Scrum é um framework para desenvolver, entregar e manter produtos complexos. Esta definição consiste em papéis, eventos, artefatos e as regras do Scrum que unem os demais e os mantém integrados.

Um dos frameworks mais comuns usados pelos adeptos ao Manifesto Ágil. O Scrum incentiva a tomada de decisão iterativa e reduz o tempo gasto em variáveis desconhecidas que estão sujeitas a alterações.

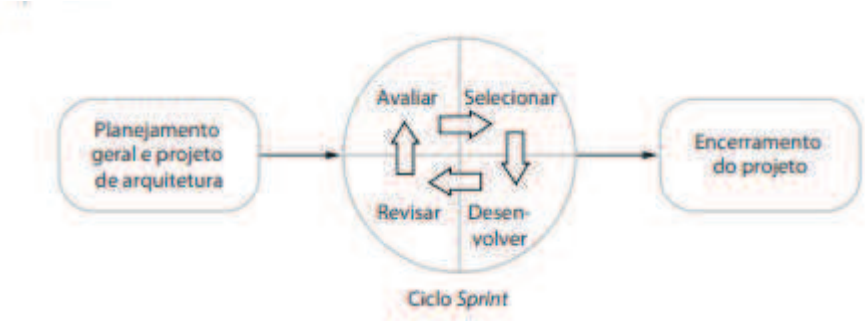
O Scrum é um subconjunto do Ágil. É uma estrutura de processo leve para desenvolvimento ágil e a mais usada (RIBEIRO e SOUZA, 2019):

- Uma “estrutura de processo” é um conjunto particular de práticas que devem ser seguidas para que um processo seja consistente com a estrutura. (Por exemplo, a estrutura do processo Scrum requer o uso de ciclos de desenvolvimento chamados sprints, a estrutura XP requer Programação em Pares e assim por diante).
- “Leve” significa que a sobrecarga do processo é mantida tão pequena quanto possível, para maximizar a quantidade de tempo produtivo disponível para realizar o trabalho útil.

Um processo Scrum se distingue de outros processos ágeis por conceitos e práticas específicas, divididos nas três categorias de funções, artefatos e caixas de tempo. Scrum é mais

frequentemente usado para gerenciar software complexo e desenvolvimento de produtos, usando práticas iterativas e incrementais (RIBEIRO e SOUZA, 2019). Este modelo está esquematizado figura 5:

Figura 5: O Processo Scrum



Fonte: Sommerville (2011)

Segundo Sommerville (2011) a característica inovadora do Scrum é sua fase central, chamada ciclos de sprint. Um sprint do Scrum é uma unidade de planejamento na qual o trabalho a ser feito é avaliado, os recursos para o desenvolvimento são selecionados e o software é implementado. No fim de um sprint, a funcionalidade completa é entregue aos stakeholders. As principais características desse processo são:

1. Sprints são de comprimento fixo, normalmente duas a quatro semanas.

2. O ponto de partida para o planejamento é o backlog do produto. Durante a fase de avaliação da sprint, este é revisto, e as prioridades e os riscos são identificados. O cliente está envolvido nesse processo e, no início de cada sprint, pode requisitar ou tarefas

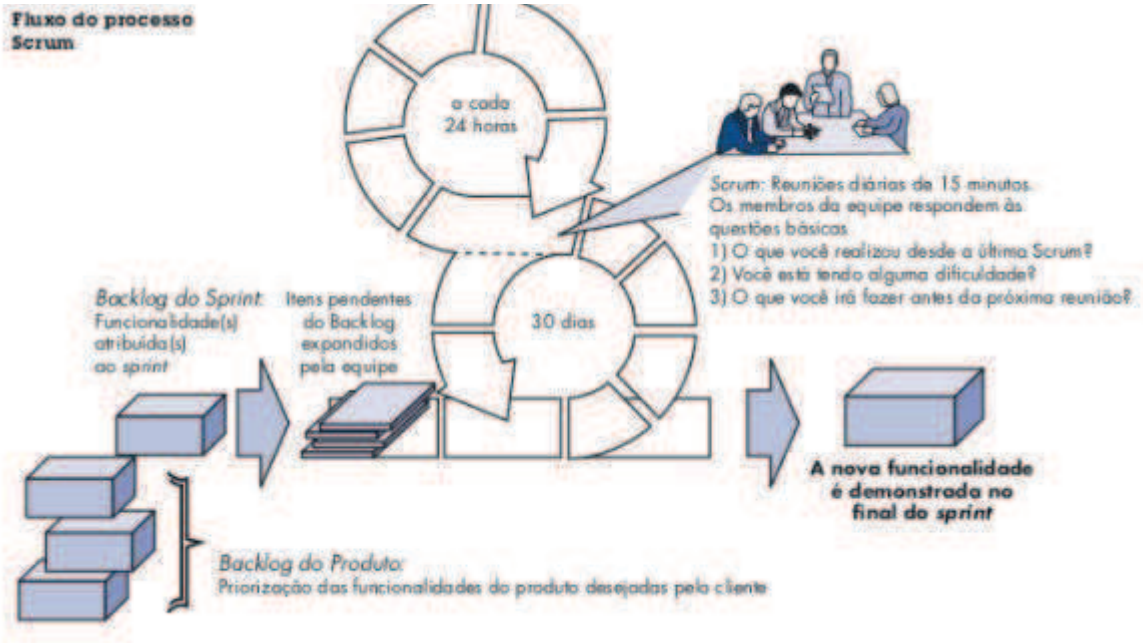
3. A fase de seleção envolve todos da equipe do projeto que trabalham com o cliente para selecionar os recursos e a funcionalidade a ser desenvolvida durante a sprint.

4. Após a seleção a equipe se organiza para desenvolver o software. Reuniões diárias rápidas, envolvendo todos os membros da equipe são realizadas para avaliar o progresso e se necessário mudar as prioridades do projeto

5. No fim da sprint, o trabalho é revisto e apresentado aos stakeholders. O próximo ciclo começa em seguida.

Os princípios e valores do Scrum estão alinhados ao manifesto ágil; as Funções, Artefatos e Caixas de Tempo, podem ser divididas em requisitos, análise, projeto, evolução e entrega. A figura 6 demonstra o fluxo Scrum:

Figura 6: O Fluxo Scrum



Fonte: Pressman (2011)

Scrum, em termos simples, é um processo orientado por valores baseado nos princípios de transparência, inspeção e adaptação. Oferece um conjunto de práticas que otimizam o trabalho em equipe, de forma a garantir um produto final de qualidade que atenda aos requisitos do cliente e proporcione uma experiência de usuário satisfatória e produtiva (SOARES, 2015). Os princípios do Scrum são orientados por 5 valores fundamentais que são eles: Coragem, Foco, Comprometimento, Respeito e Abertura.

Em essência, a equipe Scrum entra em um pacto colaborativo que inspira o mais alto desempenho individual dos membros da equipe e contribui para um fluxo de trabalho tranquilo e produtivo. Uma das principais características do processo de desenvolvimento do Scrum são as funções bem definidas de seus participantes. Existem fundamentalmente 3 funções Scrum (FOGGETTI, 2014).

- Dono do Produto: Ou Product Owner (PO) é a pessoa responsável por maximizar não só o valor do produto, como também o trabalho do time de desenvolvimento, e a gestão de Backlog do produto (SCHWABER, 2014). O PO escreve itens centrados aos clientes e classifica e prioriza esses itens, adicionando-os ao backlog do produto.
- Scrum Master: É função do Scrum master manter-se informado sobre o status do projeto e garantir que a equipe de desenvolvimento siga o processo Scrum. Para

Schwaber e Sutherland (2013) o Scrum Master é um servo-líder para o Time Scrum. O Scrum Master ajuda aqueles que estão fora da equipe de desenvolvimento a entender quais as suas interações com o Time Scrum são úteis e quais não são. Seu papel é manter o time focado ajudando toda equipe a mudarem estas interações para maximizar o valor criado pelo Time Scrum.

- Equipe de Desenvolvimento Scrum: A equipe é formada por 5-9 desenvolvedores, programadores, designers e testadores que colaboram para fazer a mágica acontecer. Eles não têm funções designadas, mas recebem um conjunto específico de tarefas para concluir em um determinado sprint (FOGGETTI, 2014).

O processo Scrum depende de 3 artefatos principais:

- Backlog do produto: O backlog do produto é um compilado de todas as funcionalidades desejadas e um projeto de desenvolvimento de software. É o processo de planejamento inicial do Scrum que define e prioriza as etapas do Scrum necessárias para a conclusão bem-sucedida do projeto. O PO e o Scrum Master colaboram para preparar o backlog do produto antes que o planejamento da sprint comece. É um documento vivo que é continuamente refinado, revisado e reordenado ao longo do processo de desenvolvimento (CRUZ, GONÇALVES e GIACOMO, 2019).
- Sprint Backlog: A sprint backlog é um microcosmo do product backlog que lista as tarefas a serem concluídas em um único sprint. As histórias de usuário que descrevem a funcionalidade desejada do produto são incluídas em um sprint backlog (FOGGETTI, 2014).
- Incremento do produto: um incremento do produto Scrum representa a soma dos itens concluídos durante um sprint. Para que um novo incremento seja “feito”, ele deve estar prontamente utilizável e disponível para liberação, embora seja responsabilidade do PO decidir quando liberá-lo (RIBEIRO e SOUZA, 2019).

Uma vez que o backlog do produto foi construído, o processo de desenvolvimento do Scrum pode começar. Consiste em uma série de sprints para realizar um conjunto específico de tarefas. A função do PO durante um sprint é responder a perguntas, aceitar histórias de usuários de membros da equipe e decidir quando um incremento está “pronto”. Cada sprint é subdividido em 5 fases ou eventos Scrum (CRUZ, GONÇALVES e GIACOMO, 2019):

- Planejamento da sprint: Durante a sessão de planejamento da sprint, o Srum Master se reúne com a equipe de desenvolvimento para planejar os detalhes do

próximo sprint. A equipe decide coletivamente quais itens de prioridade no backlog do produto podem ser razoavelmente concluídos durante a sprint, de acordo com seu tempo e recursos disponíveis. As tarefas são então atribuídas a membros individuais da equipe (RIBEIRO e SOUZA, 2019).

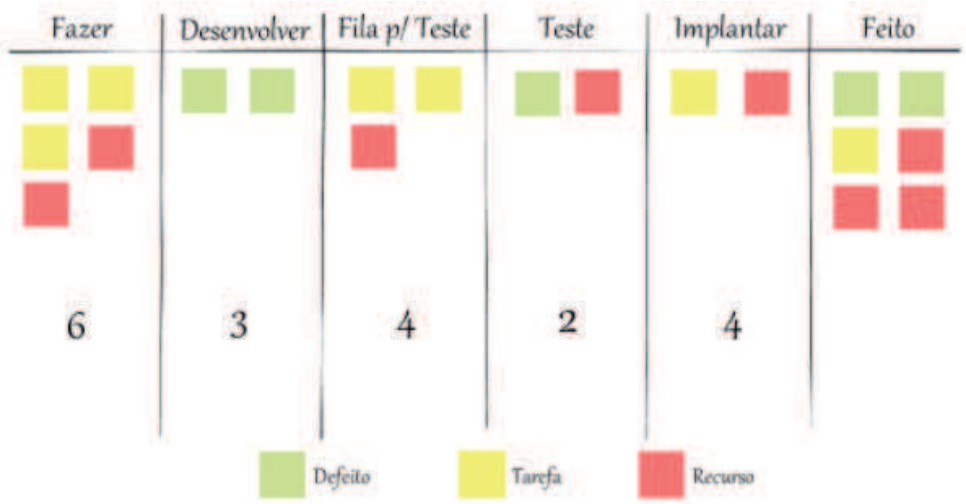
- Daily Scrum: Este briefing diário é limitado a 15 minutos ou menos. O objetivo do Scrum diário é verificar com outros membros da equipe, avaliar o progresso do dia, discutir quaisquer problemas e planejar o trabalho do dia seguinte (LOPES, 2017).
- Revisão da sprint: a revisão da sprint é uma sessão de ajuste que normalmente ocorre no último dia de um sprint. Ele analisa o que deu certo, o que deu errado e o que pode ser feito melhor. Clientes, gerentes e outras partes interessadas podem participar da revisão da sprint para ver os incrementos "concluídos" e fornecer feedback (CRUZ, GONÇALVES e GIACOMO, 2019).
- Retrospectiva da Sprint: A retrospectiva é a reunião final no final de um Sprint, com a presença do PO, Scrum Master e equipe de desenvolvimento. Durante a reunião, os participantes discutem quais melhorias podem ser feitas e como implementá-las em sprints futuros (FOGGETTI, 2014).
- Sprint: o próprio sprint é considerado um elemento do processo de desenvolvimento. Abrange todo o trabalho e eventos que ocorreram durante seu incremento de tempo limitado (CRUZ, GONÇALVES e GIACOMO, 2019).

2.4 KANBAN

O Kanban é um método de gerenciamento de fluxo de trabalho visual. Na verdade, a maioria das equipes de projeto usam Kanban para visualizar e gerenciar ativamente a criação de produtos com ênfase na entrega contínua, sem sobrecarregar a equipe de desenvolvimento. Como o Scrum, Kanban é um processo projetado para ajudar as equipes a trabalharem juntas de forma mais eficaz (AHMAD, 2013).

O quadro Kanban pode ser implementado conforme as necessidades de organização do projeto, não se limita ao número de colunas específicas. Na figura 7 está exemplificado um quadro Kanban utilizado por equipes de desenvolvimento de software, e na sequência serão explicados os elementos que compõem esse quadro.

Figura 7: Quadro Kanban



Fonte: Bernardo (2014)

O quadro do exemplo acima contém 6 colunas (Fazer, Desenvolver, Fila p/ Teste, Teste, Implantar e Feito) e 3 tipos de cartão que representam diferentes tipos de atividades. Os números na parte inferior de cada coluna representam o limite de tarefas por etapa.

No Kanban, as colunas são usadas para indicar os status pelos quais um item de valor passou antes de ser concluído. Quanto mais a direita no quadro, mais valor e tempo investidos.

Os cartões contêm uma ou múltiplas informações em forma de atributos, podendo variar de acordo com as necessidades do projeto. A identificação pode ser feita por cores conforme a figura 7 ou por pontos. Essa priorização faz com que a equipe saiba em que trabalhar primeiro, agregando mais valor ao processo.

Segundo Ahmad, 2013, O Kanban é baseado em 3 princípios básicos:

- Visualizar o fluxo de trabalho: Todos colaboradores podem enxergar o contexto do outro, levando instantaneamente o aumento da comunicação e colaboração. A visibilidade permite maior percepção ao impacto das mudanças;
- Limitar a quantidade de trabalho em andamento (WIP): Isso ajuda a equilibrar a abordagem baseada em fluxo para que as equipes não comecem e se comprometam com muito trabalho de uma vez;
- Melhorar o fluxo: quando algo for concluído, o próximo item de maior prioridade da lista de pendências precisa ser iniciado.

O Kanban promove a colaboração contínua e incentiva o aprendizado e a melhoria contínua e ativa, definindo o melhor fluxo de trabalho de equipe possível (AHMAD, 2013). As práticas a seguir são atividades essenciais para gerenciar um sistema Kanban:

- Visualizar - Os sistemas Kanban usam mecanismos como um quadro Kanban para visualizar o trabalho e o processo pelo qual ele passa. Para que a visualização seja mais eficaz, ela deve mostrar: onde no processo uma equipe trabalhando em um serviço concorda em fazer um item de trabalho específico (ponto de compromisso); onde a equipe entrega o item de trabalho a um cliente (ponto de entrega); políticas que determinam qual trabalho deve existir em um determinado estágio; Limites WIP; Limite o trabalho em andamento (LOPES, 2017). Quando se estabelece limites para a quantidade de trabalho em andamento em um sistema e usa esses limites para orientar quando iniciar novos itens, pode suavizar o fluxo de trabalho e reduzir os prazos de entrega, melhorar a qualidade e entregar com mais frequência (RIBEIRO e SOUZA, 2019).
- Gerenciar fluxo - O fluxo de trabalho em um serviço deve maximizar a entrega de valor, minimizar os prazos de entrega e ser o mais previsível possível. As equipes usam o controle empírico por meio da transparência, inspeção e adaptação para equilibrar esses objetivos potencialmente conflitantes. Um aspecto-chave do gerenciamento de fluxo é identificar e resolver gargalos e bloqueadores (LOPES, 2017).
- Tornar as políticas explícitas - Políticas explícitas ajudam a explicar um processo além da simples listagem de diferentes estágios do fluxo de trabalho. As políticas devem ser esparsas, simples, bem definidas, visíveis, sempre aplicadas e facilmente alteráveis pelas pessoas que trabalham no serviço. Os exemplos de políticas incluem: Limites de WIP, alocação de capacidade, definição de pronto e outras regras para itens de trabalho existentes em vários estágios do processo (RIBEIRO e SOUZA, 2019).
- Implementar loops de feedback - Os loops de feedback são um elemento essencial em qualquer sistema que busca fornecer mudanças evolutivas. Os loops de feedback usados no Kanban são descritos na seção Ciclo de vida. Melhore colaborativamente, evolua experimentalmente. O Kanban começa com o processo como ele existe atualmente e aplica melhorias contínuas e incrementais ao invés de tentar alcançar uma meta final predefinida (FOGGETTI, 2014).

2.5 EXTREME PROGRAMMING (XP)

Na década de 1990, o surgimento da Internet exigiu uma mudança no desenvolvimento de software. Se o sucesso de uma empresa dependesse da velocidade com que ela poderia crescer e trazer produtos ao mercado, as empresas precisariam reduzir drasticamente o ciclo de vida de desenvolvimento de software (TELES, 2005).

Foi nesse ambiente que Kent Beck criou a Extreme Programming (XP), uma metodologia ágil de gerenciamento de projetos que oferece suporte a lançamentos frequentes em curtos ciclos de desenvolvimento para melhorar a qualidade do software e permitir que os desenvolvedores respondam às mudanças nos requisitos do cliente (BEZERRA e CONCEIÇÃO, 2012).

O XP é mais do que apenas uma série de etapas para gerenciar projetos, ele segue um conjunto de valores capazes de ajudar qualquer equipe a trabalhar com mais rapidez e colaborar com mais eficácia (WÄYRYNEN et al., 2005).

- Simplicidade - As equipes realizam o que foi pedido e nada mais. O XP divide cada etapa de um processo principal em metas menores e alcançáveis para os membros da equipe cumprirem.
- Comunicação simplificada - As equipes trabalham juntas em todas as partes do projeto, desde a coleta de requisitos até a implementação do código, e participam de reuniões diárias para manter todos os membros da equipe atualizados. Quaisquer preocupações ou problemas são resolvidos imediatamente (BEZERRA e CONCEIÇÃO, 2012).
- Feedback consistente e construtivo - No XP, as equipes adaptam seus processos ao projeto e às necessidades do cliente, e não o contrário. A equipe deve demonstrar seu software com antecedência e com frequência para que possam obter feedback do cliente e fazer as alterações necessárias (WÄYRYNEN et al., 2005).
- Coragem - Os membros da equipe se adaptam às mudanças conforme elas surgem e assumem a responsabilidade por seu trabalho. Eles falam a verdade sobre seu progresso, não há “mentiras inocentes” ou desculpas para deixar de fazer as pessoas se sentirem melhor. Não há razão para temer porque ninguém trabalha sozinho (RAMOS, 2013).

A estrutura XP normalmente envolve 5 fases ou estágios do processo de desenvolvimento que se repetem continuamente (WÄYRYNEN et al., 2005):

- O planejamento, o primeiro estágio, é quando o cliente encontra a equipe de desenvolvimento e apresenta os requisitos na forma de histórias de usuário para descrever o resultado desejado. A equipe então estima as histórias e cria um plano de liberação dividido em iterações necessárias para cobrir a funcionalidade necessária, parte após parte. Se uma ou mais histórias não puderem ser estimadas, os chamados picos podem ser introduzidos, o que significa que mais pesquisas são necessárias
- O design é, na verdade, parte do processo de planejamento, mas pode ser separado para enfatizar sua importância. Está relacionado a um dos principais valores do XP que discutiremos a seguir - simplicidade. Um bom design traz lógica e estrutura ao sistema e permite evitar complexidades e redundâncias desnecessárias (BEZERRA e CONCEIÇÃO, 2012).
- Codificação é a fase durante a qual o código real é criado pela implementação de práticas XP específicas, como padrões de codificação, Programação em Pares, integração contínua e propriedade coletiva do código (a lista inteira é descrita abaixo) (TELES, 2005).
- O teste é o núcleo da programação extrema. É a atividade regular que envolve testes de unidade (teste automatizado para determinar se o recurso desenvolvido funciona corretamente) e testes de aceitação (teste do cliente para verificar se o sistema geral foi criado de acordo com os requisitos iniciais) (RAMOS, 2013).
- Ouvir tem a ver com comunicação e feedback constantes. Os clientes e gerentes de projeto são envolvidos para descrever a lógica de negócios e o valor esperado (BEZERRA e CONCEIÇÃO, 2012).
- Esse processo de desenvolvimento envolve a cooperação entre vários participantes, cada um com suas próprias tarefas e responsabilidades. A programação extrema coloca as pessoas no centro do sistema, enfatizando o valor e a importância de habilidades sociais como comunicação, cooperação, capacidade de resposta e feedback. Portanto, essas funções são comumente associadas ao XP (TELES, 2005).

As práticas do XP são um conjunto de regras e métodos específicos que o distinguem de outras metodologias. Quando usados em conjunto, eles se reforçam, ajudam a mitigar os

riscos do processo de desenvolvimento e levam ao resultado esperado de alta qualidade (RAMOS, 2013). O XP sugere o uso de 12 práticas durante o desenvolvimento de software entre elas a Programação em Pares que é descrita ao final desta seção.

- Desenvolvimento Orientado a Testes - As equipes XP praticam a técnica de desenvolvimento orientado a teste que envolve escrever um teste de unidade automatizado antes do próprio código (RAMOS, 2013). De acordo com essa abordagem, cada pedaço de código deve passar no teste para ser lançado. Portanto, os engenheiros de software se concentram em escrever código que possa realizar a função necessária. É assim que a técnica de desenvolvimento orientado permite que os programadores usem feedback imediato para produzir software confiável (BEZERRA e CONCEIÇÃO, 2012).
- O Jogo do Planejamento - Esta é uma reunião que ocorre no início de um ciclo de iteração. A equipe de desenvolvimento e o cliente se reúnem para discutir e aprovar os recursos de um produto. No final do jogo de planejamento, os desenvolvedores planejam a próxima iteração e lançamento, atribuindo tarefas para cada um deles (TELES, 2005).
- Cliente no local - Como já mencionado, de acordo com o XP, o cliente final deve participar integralmente do desenvolvimento. O cliente deve estar presente o tempo todo para responder às perguntas da equipe, definir prioridades e resolver disputas, se necessário (RAMOS, 2013).
- Refatoração de código - Para agregar valor aos negócios com software bem projetado em cada iteração curta, as equipes XP também usam refatoração. O objetivo desta técnica é melhorar continuamente o código. Refatorar é remover redundância, eliminar funções desnecessárias, aumentar a coerência do código e, ao mesmo tempo, desacoplar elementos. Desta maneira é possível manter o código limpo e simples, para que se possa facilmente entendê-lo e modificá-lo quando necessário (SONIA et al., 2014).
- Integração contínua - Os desenvolvedores sempre mantêm o sistema totalmente integrado. As equipes XP levam o desenvolvimento iterativo a outro nível porque confirmam o código várias vezes ao dia, o que também é chamado de entrega contínua (BEZNOSOV, 2003). Os praticantes de XP entendem a importância da comunicação. Os programadores discutem quais partes do código podem ser reutilizadas ou compartilhadas. Dessa forma, eles sabem exatamente quais

funcionalidades precisam desenvolver. A política de código compartilhado ajuda a eliminar problemas de integração. Além disso, o teste automatizado permite que os desenvolvedores detectem e corrijam erros antes da implantação (SONIA et al., 2014).

- Pequenos lançamentos - Essa prática sugere liberar o software rapidamente e desenvolver ainda mais o produto fazendo pequenas atualizações incrementais. Versões pequenas permitem que os desenvolvedores recebam feedback com frequência, detectem bugs antecipadamente e monitorem como o produto funciona na produção. Um dos métodos para fazer isso é a prática de integração contínua (IC) (BEZNOSOV, 2003).
- Design simples - O melhor design de software é o mais simples que funciona. Se alguma complexidade for encontrada, ela deve ser removida. O design correto deve passar em todos os testes, não ter código duplicado e conter o menor número possível de métodos e classes. Também deve refletir claramente a intenção do programador (SONIA et al., 2014). Os usuários do XP destacam que as chances de simplificar o design são maiores depois que o produto já está em produção há algum tempo. Don Wells aconselhou a escrever o código para os recursos que se planeja implementar imediatamente, em vez de escrevê-lo com antecedência para outros recursos futuros. A melhor abordagem é criar código apenas para os recursos que se está implementando enquanto busca conhecimento suficiente para revelar o mais simples Projeto (GHANI e YASIN, 2013).
- Padrões de codificação - Uma equipe deve ter conjuntos comuns de práticas de codificação, usando os mesmos formatos e estilos de escrita de código. A aplicação de padrões permite que todos os membros da equipe leiam, compartilhem e refatorem o código com facilidade, rastreiem quem trabalhou em certas partes do código, além de tornar o aprendizado mais rápido para outros programadores. O código escrito de acordo com as mesmas regras incentiva a propriedade coletiva (BOSTRÖM, 2006).
- Propriedade do código coletivo - Esta prática declara a responsabilidade de uma equipe inteira pelo design de um sistema. Cada membro da equipe pode revisar e atualizar o código. Os desenvolvedores que têm acesso ao código não entrarão em uma situação em que não conheçam o lugar certo para adicionar um novo recurso (SONIA et al., 2014). A prática ajuda a evitar a duplicação de código. A

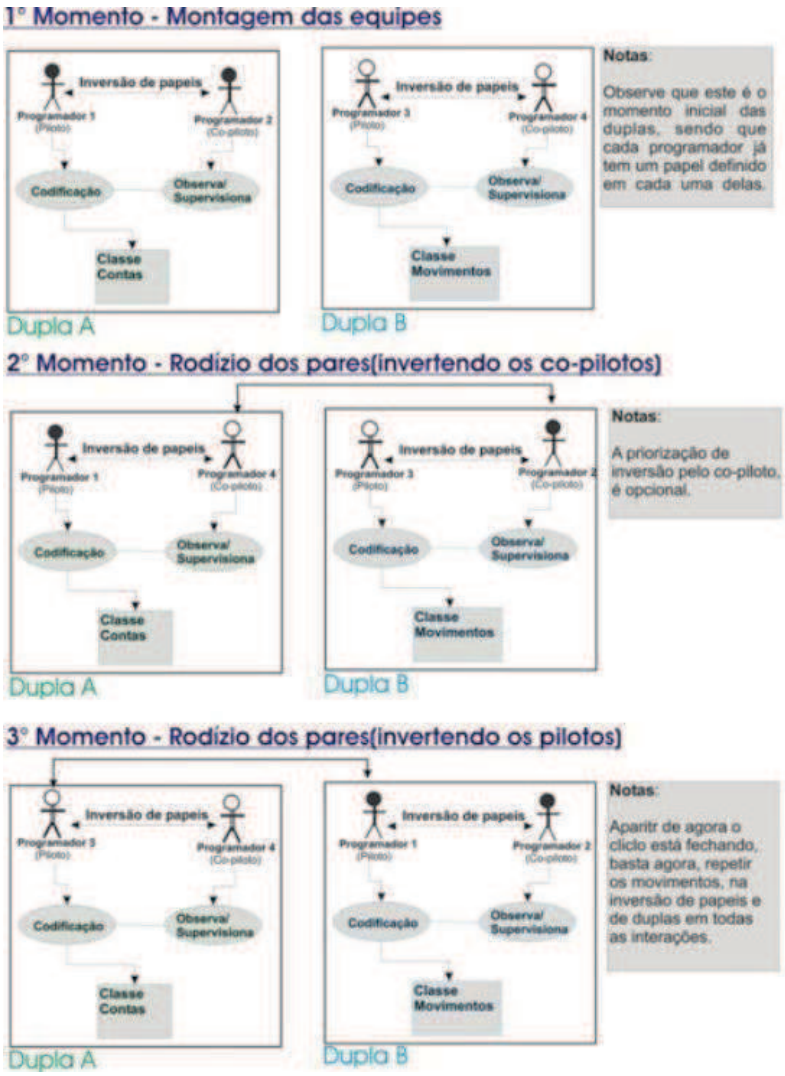
implementação da propriedade coletiva do código incentiva a equipe a cooperar mais e ficar à vontade para trazer novas ideias (GHANI e YASIN, 2013).

- Metáfora do Sistema - A metáfora do sistema representa um design simples que possui um conjunto de certas qualidades. Primeiro, um design e sua estrutura devem ser compreensíveis para novas pessoas. Eles devem ser capazes de começar a trabalhar nisso sem gastar muito tempo examinando as especificações. Em segundo lugar, a nomenclatura de classes e métodos deve ser coerente. Os desenvolvedores devem ter como objetivo nomear um objeto como se ele já existisse, o que torna o design geral do sistema compreensível (BOSTRÖM, 2006)
- Semana de 40 horas - Os projetos XP requerem que os desenvolvedores trabalhem rápido, sejam eficientes e mantenham a qualidade do produto. Para cumprir esses requisitos, eles devem se sentir bem e descansados. Manter o equilíbrio entre vida pessoal e profissional evita o esgotamento dos profissionais (GHANI e YASIN, 2013). No XP, o número ideal de horas de trabalho não deve exceder 45 horas por semana. Uma hora extra por semana só é possível se não houver nenhuma na semana seguinte (BOSTRÖM, 2006).
- Programação em Pares segundo Sommerville (2011), leva ao profundo conhecimento de um programa, pois ambos desenvolvedores precisam compreender seu funcionamento em detalhes para continuar o desenvolvimento em conjunto. Assim, possíveis bugs são detectados com mais frequência o que aumenta a qualidade das tarefas entregues.

Para aplicar a Programação em Pares desenvolvedores são divididos em duplas, cada dupla contém um piloto e um copiloto que trabalham em um único computador, enquanto um dos programadores está codificando, o outro acompanha seu trabalho, observando se as boas práticas e se o padrão de codificação está sendo atendido, auxiliando e corrigindo possíveis erros. De maneira geral, os programadores trabalham em par para compartilhar conhecimentos para melhor implementar as tarefas definidas (MEDEIROS, 2007).

A figura 9 demonstra um modelo de aplicação da prática de Programação em Pares, no qual os programadores revezam suas funções.

Figura 8- Modelo de Programação em Pares



Fonte: Medeiros (2007)

Segundo Wildt et al. (2015), os seguintes benefícios são observados utilizando o pareamento de desenvolvedores:

- Revisão do código continua: muitos erros são pegos quando eles estão sendo codificados, em vez de serem descoberto por um testador, o que torna a quantidade de erros consideravelmente menor;
- Discussão contínua da solução: o design é aprimorado e o código fica mais sucinto;

- Afinamento do par: o time como um todo resolvem os problemas mais rapidamente por está coeso também em pares;
- Gestão do conhecimento: o processo acaba com o conhecimento distribuído no qual várias pessoas entendem cada pedaço do sistema.

A Programação em Pares é primordial para entregas de qualidade método XP, podendo ser praticada em conjunto com outros métodos. Programar em par sugere que duas pessoas trabalhando em um único computador produzirá mais que duas pessoas trabalhando separadamente, com as premissas que o foco nas conclusões das tarefas aumenta e também gere mais qualidade no código e troca de conhecimento (WILDT; MOURA; LACERDA; HELM, 2015).

2.6 COMPARAÇÃO ENTRE OS MÉTODOS APRESENTADOS

O XP, como o Scrum e Kanban fazem parte da metodologia ágil e compartilham os principais princípios ágeis, ou seja, lançamentos frequentes, ciclos de desenvolvimento curtos, comunicação constante com o cliente, equipes multifuncionais e assim por diante (BEZERRA e CONCEIÇÃO, 2012).

O Scrum é comumente associado a equipes auto-organizadas. Geralmente também tem sprints de 2 a 4 semanas de duração, enquanto as iterações de XP são mais curtas, levando de 1 a 2 semanas. Além disso, o XP é muito mais flexível com possíveis mudanças dentro das iterações, enquanto o Scrum não permite nenhuma modificação depois que o backlog da sprint é definido. Outra diferença é que no XP o cliente prioriza os recursos e decide a ordem de seu desenvolvimento, mas no Scrum a própria equipe determina no que trabalhar primeiro (BOSTRÖM, 2006).

As principais funções do Scrum são Product Owner, Scrum Master e Scrum Team, que são diferentes daquelas do XP. No entanto, não há necessidade de escolher entre XP e Scrum. Incorporar práticas XP e técnicas Scrum é considerado bastante eficaz com XP focando em aspectos de engenharia e Scrum organizando o processo (SANTOS, 2013).

O Kanban coloca muito foco na visualização do processo de desenvolvimento e limita estritamente o número de recursos desenvolvidos por vez. Também é caracterizado por um fluxo de trabalho contínuo, enquanto o XP tem iterações separadas, embora ambos sugiram pequenos lançamentos frequentes e um alto nível de flexibilidade e adaptabilidade aos requisitos em mudança. As funções no Kanban não são estritamente definidas (RAMOS, 2013).

Após discutir os métodos, conclui-se que podem ser adotados de acordo com as necessidades de cada projeto. Por exemplo, um projeto de suporte à produção deve adotar Kanban, um projeto de desenvolvimento de melhorias pode adotar Scrum incrementando a cada sprint, se as iterações forem muito curtas devido a mudanças frequentes de requisitos, então a metodologia XP se adequa melhor. E ainda, os métodos e seus atributos podem ser usados de maneira conjunta, criando um mix de acordo com as necessidades do desenvolvimento.

3 ESTUDO DE CASO

Neste capítulo apresentam-se as informações provenientes de um estudo de caso realizado na empresa Rota Exata Software Ltda. A partir deste estudo teve-se uma descrição do cenário atual e, com base nos conceitos apresentados anteriormente, fez-se uma proposta de desenvolvimento de software. Esta proposta está descrita no Capítulo 4.

Neste Capítulo 3, inicialmente, é apresentada a caracterização da empresa. Na segunda seção detalham-se o modelo e métricas utilizados para desenvolvimento de software da empresa. Para esse estudo de caso foram analisados os resultados de 7 sprints.

3.1 CARACTERIZAÇÃO DA EMPRESA

A Rota Exata Software LTDA é uma empresa de rastreamento e gestão de frotas, a qual tem o foco em entregar soluções que reduzam os custos operacionais que os seus clientes têm durante o dia a dia, ao mesmo tempo que facilita a gestão veicular. A empresa conta três principais produtos para atingir esses objetivos, que são divididos em três módulos.

- Rastreamento - Visa o fornecimento e gerenciamento sistemático da posição e do estado dos veículos em tempo real, com variados níveis de exatidão e intervalos de tempo.
- Gestão - Controle de gastos incluindo manutenções, abastecimentos, multas, a conformidade com os serviços prestados, políticas de segurança, leis municipais, estaduais e federais.
- Operação - Planejamento e roteirização de serviços e entregas.

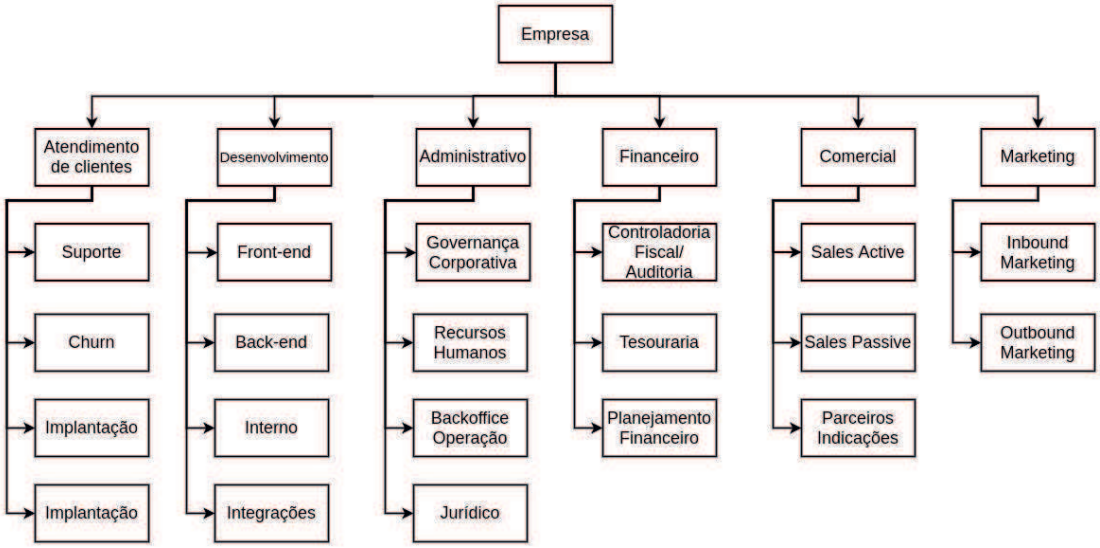
Atualmente, a Rota Exata conta com cerca de 1.000 empresas clientes, e possui por volta de 10.000 veículos rastreados por todo o Brasil. Além disso, conta com três empresas parceiras, as quais são revendedoras de seu software.

A empresa tem como meta desenvolver um novo módulo chamado Automação, que propõe automatizar a gestão do time de campo de motoristas, criando regras e formulários condicionais, enviando mensagens automáticas aos motoristas de acordo com as regras previamente definidas pelo usuário, como controle de excesso de velocidade, aceleração e frenagem bruscas, desvios de rotas, entre outras. E também gerar um ranking entre os motoristas baseado nas quebras dessas regras.

O novo módulo gerou um aumento de demanda para o time de desenvolvimento. Tendo em vista esse novo cenário a empresa decidiu mudar o seu modelo de desenvolvimento de software com a finalidade de dar mais velocidade aos processos e aumentar a entregabilidade de novas funcionalidades. O modelo utilizado anteriormente era o Kanban, e o modelo atual será descrito na próxima seção.

A estrutura organizacional está dividida em 6 setores, são eles Administrativo, Atendimento de clientes, Comercial, Desenvolvimento, Financeiro e Marketing, conforme ilustrado na figura 9.

Figura 9: Estrutura organizacional Rota Exata



Fonte: Desenvolvido pelo autor (2021)

Cada um dos setores engloba áreas menores, as quais realizam tarefas específicas, e o conjunto de todas essas tarefas garante a entrega contínua do produto.

O setor de Desenvolvimento é responsável por desenvolver e assegurar a qualidade técnica do produto no mercado. A equipe de desenvolvimento é composta por 8 pessoas sendo um Product Owner, um Scrum Master e Time Scrum formado por seis desenvolvedores. A fim de preservar a identidade dos desenvolvedores vamos chamá-los de Dev1, Dev2, Dev3 e Dev4 responsáveis pelo desenvolvimento do front-end, Dev5 e Dev6 responsáveis pelo desenvolvimento do back-end. Para a realização do trabalho consideraremos os desen

3.2 PROCESSO DE DESENVOLVIMENTO DE SOFTWARE NA ROTA EXATA

A empresa optou por utilizar métodos descritos no capítulo 2, adaptando práticas do Scrum e do Kanban para o processo de desenvolvimento. Serão apresentados nessa seção, o modelo de desenvolvimento de software e as métricas utilizadas pela empresa.

Dada uma listagem de demandas, vindas das mais diversas origens, tanto do mercado quanto uma necessidade de atualização interna, um comitê formado por um membro de cada área da empresa levanta todas as possíveis melhorias do software a serem implementadas e passam essa informação para o backlog do produto, o Product Owner define a priorização dessas tarefas de acordo com o seu conhecimento do que agrega mais valor para o produto.

Definida a ordem de priorização o Scrum Master junto com o desenvolvedor mais experiente montam a sprint com ciclo de uma semana, por um sistema pontuação nas tarefas de acordo com o tempo médio a serem realizadas pelos desenvolvedores. Além dos pontos referentes às tarefas, os desenvolvedores podem somar mais pontos com correções de bugs no software e auxílio para os mais novos do time, mediante a justificativa ao Scrum Master.

A figura 10 ilustra o modelo de controle do sprint, já revisado pelo Scrum Master, praticado na empresa. Exemplificando algumas tarefas a serem cumpridas pelos desenvolvedores front-end.

Na figura 10, pode-se observar os pontos distribuídos entre os desenvolvedores Dev1, Dev2, Dev3 e Dev4 no canto superior esquerdo. No canto superior direito está a pontuação total obtida pelos desenvolvedores em relação a quantos pontos foram previstos para aquele sprint. Logo abaixo estão discriminados:

- Os auxiliares do time de desenvolvimento;
- O cronograma que marca o início e o final do sprint;
- Definição de tempo, que se refere ao peso da pontuação de cada tarefa, que serão explicados no decorrer deste capítulo.

O controle do sprint é feito em duas etapas, na etapa inicial o Scrum Master lista as tarefas, determina o responsável pela tarefa e define o peso de cada tarefa. Nesta etapa são preenchidas as seguintes colunas: “Atividade”, “Dono” e “Pontos”.

A etapa final ocorre no término do sprint, o Scrum Master verifica o status da tarefa no Kanban, e se a tarefa foi concluída corretamente. Nessa etapa são preenchidas as colunas “Kanban”, “Retrabalho” e “Realizado”. A figura 10 apenas ilustra o modelo, não listando a totalidades das tarefas determinadas naquele sprint.

Figura 10: Modelo de sprint da empresa Rota Exata

Sprint

			Total de Pontos **67/90**

Time Front-end:
Dev1 (6/dia) **= 30**
Dev2 (4/dia) **= 20**
Dev3 (4/dia) **= 20**
Dev4 (4/dia) **= 20**

Auxiliares dos Times
Product Owner
Scrum Master:

Cronograma
Início: 15/03/2020 - Segunda
Previsto:19 /03/2020 - Sexta
Término:

Definições de Tempo
1 hora: 1 ponto
¼ dia: 2 pontos
½ dia: 3 pontos
1 dia: 6 pontos

Sprint - Time Front-end.

Kanban	Dono	Atividade	Pontos	Retrabalho	Realizado
Sim	Dev1	Roteirização de Visitas - Não aparece a variável de tipo de carga em todas as tabelas que tem nos modais	6	Não	Sim
Sim	Dev1	Minha Rota - Mudar a label do botão para `Formulário ${nome do formulário} `	1	Não	Sim
Sim	Dev3	Minha Rota - Após dar check-in, a visita já pode ser marcada como realizada na página principal.	2	Não	Sim
Sim	Dev3	Roteirização de Visitas - Quando a API de planejamento dá erro, deve aparecer a mensagem retornada em um toast - (Entre em contato com o suporte).	2	Não	Sim
Sim	Dev2	Agendamento de Visitas - Ao cadastrar um destino do zero, preencher o campo endereço não preenche latitude e longitude.		Não	Sim
Sim	Dev4	Roteirização de Visitas - Avisar quando alguma visita for retirada da rota e enviada de volta para roteirização. - MARCAR EM AMARELO NA SEGUNDA ABA AS VISITAS QUE TEM ROTA_ID_RECUSADA	3	Não	Sim

Fonte: Desenvolvido pelo autor (2021)

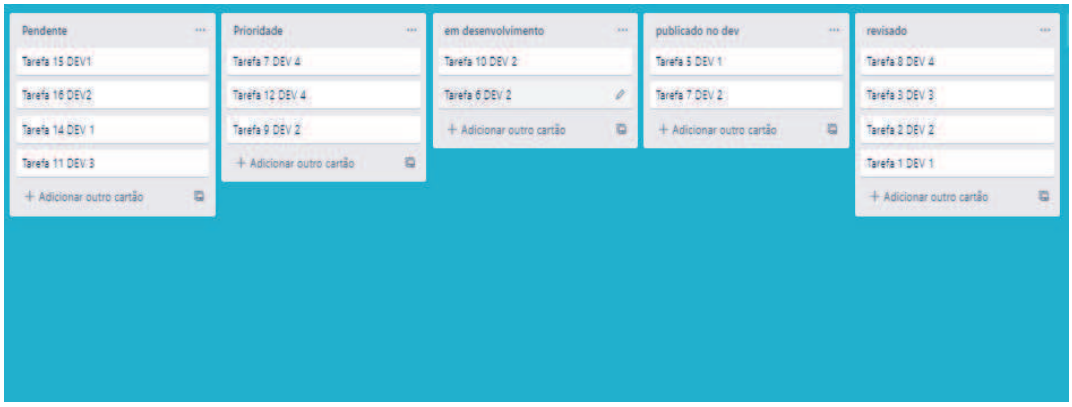
Após definido o sprint as tarefas e suas pontuações são colocadas pelo Scrum Master no fluxo de trabalho do quadro Kanban, onde cada desenvolvedor consegue visualizá-las e atualizar o status para: “pendente”, “prioridade”, “em desenvolvimento”, “publicado no dev” e no final todas tarefas são revisadas pelo Scrum Master. Após a revisão a tarefa poderá ser concluída ou recusada. Cada desenvolvedor tem o tempo do ciclo do sprint para realizar as

tarefas designadas, ficando a cargo deles a ordem de execução. As tarefas ou retrabalhos não realizados durante o ciclo voltam para o backlog do sprint. Abaixo a explicação de cada status das tarefas do fluxo de trabalho.

- Pendente - A tarefa ainda não foi priorizada pelo desenvolvedor, caso encerre o ciclo Scrum e a tarefa ainda esteja em pendente, ela voltará para o backlog do sprint.
- Prioridade - A tarefa foi analisada pelo desenvolvedor e será concluída até o final do ciclo dessa sprint, caso o desenvolvedor não a concluir completamente, ela voltará como retrabalho no próximo sprint.
- Em desenvolvimento – A tarefa está progredindo e sendo desenvolvida, caso encerre o sprint nessa etapa a tarefa voltará como retrabalho na próxima o sprint, com uma pontuação menor.
- Publicado no dev – O termo dev nesse status refere-se ao ambiente de programação onde os desenvolvedores publicam a conclusão de suas tarefas, após a publicação encerra-se a participação do desenvolvedor nessa tarefa.
- Revisada – Nessa etapa o Scrum Master faz o papel de qualidade e testa e revisa a tarefa publicada no status anterior. Caso a tarefa esteja aceitável ela é concluída, caso não, retornará para o backlog do sprint.

A figura 11 ilustra o quadro Kanban na visão dos desenvolvedores:

Figura 11: Quadro Kankan desenvolvedores



Fonte: Desenvolvido pelo autor (2021)

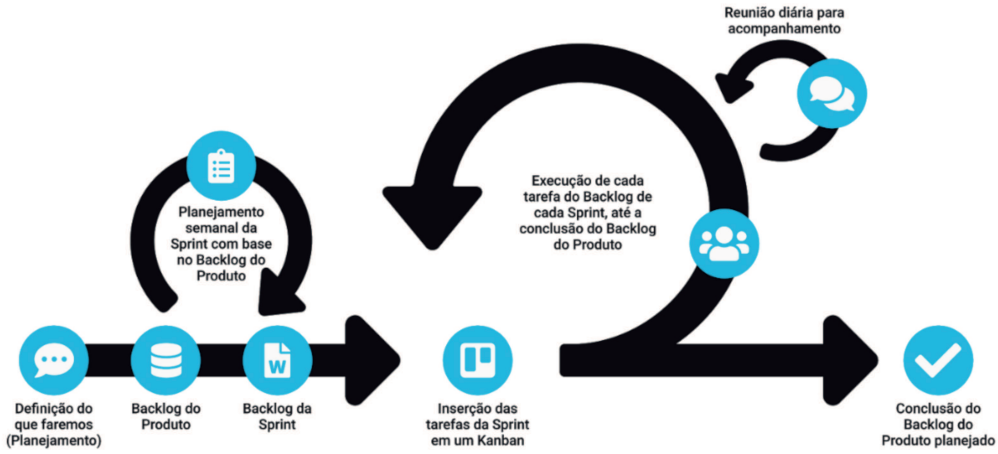
Todos os dias do ciclo do sprint são realizadas as Dailly Scrum, para acompanhamento da realização das tarefas, onde cada desenvolvedor aponta o que foi feito desde a última reunião e o que pretende fazer em seguida, e se houve algum impedimento. Essa reunião é mediada pelo

Scrum Master e se torna importante na troca de experiência entre os desenvolvedores e tem uma duração de apenas 15 minutos.

Durante o ciclo, o Product Owner e o Scrum Master assumem o papel de garantir a qualidade do produto desenvolvido, realizando testes e verificando o que foi feito nas atividades que estão no quadro Kanban dos desenvolvedores, e definem essas atividades de acordo com o resultado do teste como concluída quando a atividade já pode ser publicada, necessita de edição quando necessita de alguma correção e recusada quando não está de acordo com o solicitado.

A figura 12 exemplifica o fluxo do desenvolvimento de novas funcionalidades do produto da empresa:

Figura 12: Fluxo de desenvolvimento do produto - ROTA EXATA



Fonte: Desenvolvido pelo autor (2021)

Ao final do fluxo é incrementado as melhorias e novas funcionalidades propostas no planejamento, e a etapa de planejamento é revisitada para dar origem ao novo backlog do produto.

Após o encerramento do ciclo de cada sprint o Scrum Master quantifica o resultado individual de cada desenvolvedor baseado nos pontos distribuídos em cada tarefa e os pontos justificados pelo desenvolvedor que não estavam previstos nas sprints.

Os desenvolvedores são avaliados individualmente de acordo com suas entregas em cada sprint, sendo assim possível uma avaliação semanal. São apurados indicadores gerais, os quais resumem como está toda a produtividade do desenvolvedor desde o início do acompanhamento. Além do tempo eles são avaliados pela qualidade de suas entregas pelos índices de tarefas concluídas e retrabalho.

As pontuações de cada tarefa são medidas em pontos de acordo com o tempo definido para ser cumprida, usando como base um dia de 6 horas de trabalho. Os pontos variam entre uma composição de 1,2,3 e 6 pontos sendo que:

- 1 ponto refere-se a 1 hora de trabalho do desenvolvedor, utilizado para tarefas mais simples.
- 2 pontos referem-se a ¼ de dia trabalhado.
- 3 pontos referem-se ½ dia trabalhado.
- 6 pontos referem-se a 1 dia trabalhado.

Cada desenvolvedor recebe um saldo de pontos estipulados pelo Scrum Master no início de cada semana. Este saldo é estipulado de acordo com a previsão do que o desenvolvedor irá fazer nos sprints e possíveis tarefas não previstas. Ao longo da semana esse saldo é debitado com as seguintes definições:

- Concluídas - Tarefas previstas no sprint que foram concluídas com sucesso;
- Retrabalho - tarefas previstas no sprint que foram concluídas, mas que terão retrabalho em próximas sprints;
- Não previsto - tarefas não previstas no sprint que foram concluídas e justificadas;
- Não metrificado - Número de pontos em que não se sabe o que o desenvolvedor fez.

A diferença entre o saldo e os pontos debitados geram o indicador Não Metrificado, caso a diferença retorne a um número de pontos negativo, significa que o desenvolvedor trabalhou mais que o previsto. Esse indicador é importante para que o Scrum master defina a redistribuição de pontos, o objetivo é deixá-lo o mais próximo de zero possível. A Tabela 1 mostra o resultado de cada sprint por desenvolvedor.

Tabela 1: Resultados sprint, por desenvolvedor

RESULTADOS DA SPRINT						
	Sprint	Saldo	Concluido	Retrabalho	Não previsto	Não metrificado
Dev1	1	30	8	0	7	15
	2	24	9	1	10	4
	3	30	14	2	13	1
	4	27	16	0	3	8
	5	30	16	2	7	5
	6	30	22	0	12	-4
	7	24	20	0	0	4
Dev2	1	20	8	6	0	6
	2	20	11	0	13	-4
	3	20	16	0	6	-2
	4	17	10	0	2	5
	5	12	4	0	10	-2
	6	16	11	0	0	5
	7	14	9	0	0	5
Dev3	1	20	11	6	4	-1
	2	20	12	0	2	6
	3	20	18	0	2	0
	4	5	3	0	0	2
	5	20	15	2	2	1
	6	20	17	0	3	0
	7	16	5	4	4	3
Dev4	1	20	8	3	4	5
	2	20	11	0	3	6
	3	20	20	0	5	-5
	4	17	12	0	7	-2
	5	20	15	0	0	5
	6	20	18	0	1	1
	7	16	7	0	7	2

Fonte: Desenvolvido pelo autor (2021)

Pode-se perceber que o Dev1 é o que possui mais pontos não previsto e não metrificado, e também o que recebe o maior saldo e consequentemente é o que recebe mais tarefas. O Dev2 também possui um alto número de tarefas não previstas. O Dev3 é o que possui menos pontos não previstos e o maior índice de retrabalho, e no sprint 4 esteve ausente 3 dias, por isso o baixo saldo. Já o Dev4 teve um baixo índice de retrabalho e índice não previsto abaixo da média.

A partir desses resultados são mensurados os indicadores globais relativos para entender a distribuição de pontos gastos pelo desenvolvedor em relação aos saldos previstos pelo Scrum Master, conforme a Tabela 2.

Tabela 2: Indicadores Globais relativos ao Saldo

Desenvolvedor	Saldo	Concluídas	Retrabalho	Não Previsto	Não metrificado
Dev 1	195	53,85% 105	2,56% 5	27% 52	16,92% 33
Dev 2	119	57,98% 69	5,04% 6	26% 31	10,92% 13
Dev 3	121	66,94% 81	9,92% 12	14% 17	9,09% 11
Dev 4	133	68,42% 91	2,26% 3	20% 27	9,02% 12
Global	568	60,92% 346	4,58% 26	22% 127	12,15% 69

Fonte: Desenvolvido pelo autor (2021)

E ainda são medidos os indicadores relacionados a sprint, que mensuram o desempenho das entregas nas sprints, medindo as tarefas concluídas do sprint e o retrabalho gerado, sem levar em consideração os índices de Não Previstos e Não Metrificado, conforme a Tabela 3:

Tabela 3: Indicadores Globais relativos ao sprint

Desenvolvedor	Pontos Sprint	Concluídas Sprint	Retrabalho Sprint
Dev 1	142	73,94% 105	3,52% 5
Dev 2	90	76,67% 69	6,67% 6
Dev 3	93	87,10% 81	12,90% 12
Dev 4	94	96,81% 91	3,19% 3
Global	419	82,58% 346	6,21% 26

Fonte: Desenvolvido pelo autor (2021)

4 MODELO PROPOSTO PARA GESTÃO ÁGIL DE PROJETOS BASEADA NA PRÁTICA DE PROGRAMAÇÃO EM PARES PARA O DESENVOLVIMENTO DE SOFTWARE

Neste capítulo será apresentado uma proposta de melhoria no desenvolvimento de software da empresa, baseada nos índices individuais de cada desenvolvedor analisado, com o objetivo de diminuir os indicadores de retrabalho e números de tarefas não previstas. Desta forma, ter-se-á um novo modelo de gestão para o processo de desenvolvimento de software.

4.1 DIAGNÓSTICO DE PROBLEMAS

O diagnóstico foi realizado por meio de observações in loco em participações de sete (07) Dally Scrum, entrevistas com o Time Scrum e análise dos indicadores que a empresa possui. Com base nestas ações, foram identificados dois problemas principais:

1) Falta de priorização entre tarefas novas e correções de bugs ou auxiliar outros desenvolvedores. Esse é um dos fatores que contribuem para o aumento do índice de tarefas não previstas.

2) Estimativa do restante do trabalho ao longo da sprint errada. Como o processo foi implantado recentemente não é feito um acompanhamento da conclusão de tarefas no decorrer da sprint, apenas após a sua conclusão.

Após analisar o resultado individual de cada sprint levantou-se a hipótese de que o índice de Não Previsto, aumenta de acordo com o tempo em que o desenvolvedor está na empresa. O Dev1 e o Dev2 possuíam um ano e meio de casa contando da primeira sprint observada, e o Dev3 e Dev4 tinham dois meses de casa na primeira sprint observada.

Uma das justificativas que levam ao auto índice de Não Previsto para os desenvolvedores mais experientes com o software, é que eles auxiliam os menos experientes. Contudo, a empresa tem uma política de contratar desenvolvedores iniciando como estagiários para que cresçam e façam carreira na empresa, por isso não abre mão dessa troca de conhecimento entre os funcionários.

Outro fato observado é que o tamanho do sprint seja curto, o que é característico do modelo XP. Neste cenário uma das práticas do XP foi proposta com objetivo de trazer uma

troca de experiência entre os desenvolvedores, ao mesmo tempo em que mais tarefas são concluídas no decorrer da semana.

Para melhorar a gestão do processo de desenvolvimento de software da empresa estudada foi proposta a implementação de uma prática do método XP que é a Programação em Pares.

4.2 Implementação da Programação em Pares

Adequou-se o modelo a realidade da empresa, determinando inicialmente a montagem das duplas. Os desenvolvedores 1 e 2 como os copilotos de cada dupla e os desenvolvedores 3 e 4 como pilotos, portanto, a dupla 1 composta por Dev 1 e Dev 3 e a dupla 2 composta por Dev2 e Dev 4. A seguir as funções de cada papel da dupla:

- Piloto – Assume o teclado e digita os comandos que farão parte do programa.
- Copiloto – Envolvido ativamente na tarefa de programação, mas se concentrando na direção geral, fazendo um trabalho estrategista.

Em decorrência da pandemia Covid-19, o modelo de Programação em Pares precisou ser adaptado à realidade do distanciamento social onde os desenvolvedores trabalharam em modelo de Home Office. Para a adaptação criou-se ambiente virtual, onde cada programador tinha acesso por áudio e vídeo a sua dupla e a tela de codificação pode ser espelhada simultaneamente por meio do software Google Mets.

O ciclo da sprint foi mantido e para sua montagem, o Scrum master junto ao Product Owner e o desenvolvedor mais experiente definiram as tarefas e pontuações conforme a sprint descrita no APÊNDICE A. É importante ressaltar que os pontos das tarefas foram descritos levando em consideração que o peso de cada tarefa diminui quando executadas em dupla. Cada dupla recebeu um saldo equivalente a 25 pontos, que serão gastos no decorrer da semana.

Cada desenvolvedor foi orientado a seguir boas práticas para aprimorar a Programação em Pares seguindo 4 abordagens descritas segundo Wildt et al. (2015).

- Vamos tentar sua ideia primeiro: alguém da dupla sugere começar pela ideia da solução do outro integrante. Essa abordagem cria um clima de respeito, entendo que as soluções de cada um serão experimentadas;
- Pensar alto: o piloto fala o que está pensando em quanto escreve o código. Isso ajuda a compreensão do copiloto no entendimento do que está sendo feito;

- Regra dos dez segundos: muitas vezes o piloto está em um raciocínio aprimorado e não linear. O copiloto deve respeitar esse momento e aguardar cerca de dez segundos para intervir;
- Revezar por ciclo de tempo: revezar o papel do piloto e copiloto é essencial. A sugestão é que cada par define um ciclo de tempo e coloque um alarme para despertar para fazer a troca. Esse ciclo não pode ser nem muito curto, nem muito grande. O par saberá o tempo certo, que normalmente varia de dez minutos a uma hora.

Foram mantidas as reuniões Dally Scrum, na qual cada dupla de desenvolvedores aponta o que foi feito desde a última reunião e o que pretende fazer em seguida, e se houve algum impedimento.

Cada dupla foi responsável por alterar o status das tarefas no quando Kankan, definindo a próxima tarefa a ser cumprida e certificando que a tarefa foi concluída com sucesso. A responsabilidade de atualizar o status de cada tarefa ficou por conta do copiloto momentâneo em cada ocasião, podendo alterar os status em seu próprio computador.

Para simplificar e dar mais fluidez a implementação as duplas foram mantidas em todo o ciclo da sprint e não foi metrificado quantas trocas ocorreram entre pilotos e copilotos no decorrer do desenvolvimento.

4.3 Avaliação da implementação

Neste capítulo serão abordados os resultados obtidos com a implementação da Programação em Pares proposta.

Após a implantação foram levantados os indicadores por dupla e globais, para serem comparados com o modelo atual e aplicado um questionário dissertativo com perguntas qualitativas e objetivo exploratório para melhor compreender a experiência que cada participante obteve com a implantação da Programação em Pares. A seguir o Quadro 1 apresenta as questões aplicadas no questionário:

Quadro 1: Questões do Questionário

Questionário
1. Você já teve alguma experiência anterior com a Programação em Pares?
2. No desenvolvimento das tarefas, você aprendeu algo novo com seu par?
3. Você teve alguma dificuldade ou desconforto em programar em par?
4. No desenvolvimento das tarefas, você teve algum conflito com seu par?
5. Na sua visão, a implantação da Programação em Pares traria benefícios ao modelo de desenvolvimento de software atual da empresa?
6. Na sua visão como você descreve a experiência de programar em pares?

Fonte: Desenvolvido pelo autor (2021)

Também foram analisados os resultados observados na implementação, comparando o modelo atual com o proposto. E, por fim, será discutida a experiência dos desenvolvedores que participaram da implementação. Avaliou-se os percentuais devido ao modelo implementado de Programação em Pares ter ocorrido em um período menor ao modelo atual.

Com base nos indicadores do modelo de desenvolvimento atual da empresa, adaptou-se para fins de comparação a coleta de dados no modelo em pares. A tabela 4 descreve os resultados dos Indicadores Globais relativos ao Saldo do sprint da semana observada:

Tabela 4: Indicadores Globais relativos ao Saldo do modelo implementado

Desenvolvedor	Saldo	Concluídas	Retrabalho	Não Previsto	Não metrificado
Dupla 1	25	76% 19	4% 1	8% 2	12% 3
Dupla 2	25	72% 18	8% 2	12% 3	8% 2
Global	50	74% 37	6% 3	10% 5	10% 5

Fonte: Desenvolvido pelo autor (2021)

Com a implementação do modelo houve um aumento de aproximadamente 13% nas tarefas concluídas em relação ao modelo atual. Já o retrabalho aumentou cerca de 1,4%. Tarefas não previstas tiveram uma queda de 12% e não metrificados caíram 2,15%.

Um dos fatos que contribuiu com o aumento das tarefas concluídas é a responsabilidade mútua em concluir as tarefas, o que gera uma pressão em cada desenvolvedor em entregar as tarefas no prazo para não prejudicar seu parceiro nas entregas. Por outro lado, houve um aumento no índice de retrabalho, o que demonstra que apesar de mais tarefas serem concluídas, ainda há erros propagados que precisam ser observados e minimizados no próximo sprint.

Conforme esperado o índice de não previstos teve queda. Isso ocorreu devido ao fato de não terem pontos justificados como auxílio a outro desenvolvedor, ficando apenas a cargo de correção de bugs. Entretanto, houve uma diminuição da comunicação entre as duplas, por estarem focadas em suas tarefas, restringindo a troca de informações apenas nas Daily Scrum.

Os pontos não metrificados por sua vez tiveram uma pequena queda quando comparado ao modelo atual. Essa queda pode ser justificada pelo fato de que os desenvolvedores tiveram menos interações com o restante da equipe de desenvolvimento e ainda foi atenuado pelo fato de estarem em Home Office.

Outros indicadores medidos são relacionados ao sprint, que mensuram o desempenho das entregas, medindo as tarefas concluídas no sprint e o retrabalho gerado:

Tabela 5: Indicadores globais relativo ao sprint

Desenvolvedor	Pontos Sprint	Concluídas Sprint	Retrabalho Sprint
Dupla 1	20	95,00% 19	5,00% 1
Dupla 2	20	90,00% 18	10,00% 2
Global	40	92,50% 37	7,50% 3

Fonte: Desenvolvido pelo autor (2021)

Observando os indicadores relativos ao sprint, há um aumento de cerca de 10% de concluídas no sprint, e o retrabalho de aproximadamente 1,3%. Foram concluídas todas as tarefas, porem 7,5% das tarefas concluídas foram recusadas e marcadas como retrabalho. É importante salientar que tanto a dupla 1 quanto a dupla 2 tiveram comprometimento na entrega das tarefas no prazo, não havendo assim tarefas não visitadas, demonstrando uma boa distribuição dos pontos pelo Scrum Master.

No que se refere ao **questionário**, apresentado no quadro 1, as respostas do time de desenvolvedores back-end, estão disponíveis no APÊNDICE B. Nesta seção é apresentado os principais pontos sobre a experiência em que os participantes obtiveram em relação à Programação em Pares.

Todos os desenvolvedores envolvidos afirmaram que já haviam tido experiências anteriores com a Programação em Pares. E que em nessa aplicação aprendeu algo novo em decorrer das atividades desenvolvidas com seu par. O Dev3 ainda comenta que aprendeu novas maneiras de codar (desenvolver os códigos) e achar os erros no código.

Em relação a dificuldades e desconfortos de trabalhar em par, o Dev1 afirma não ter sentido nenhuma dificuldade ou desconforto, entretanto acredita que seu par pode ter ficado desconfortável em algumas situações. Já o Dev 2 sentiu-se desconfortável, e salientou que em algumas situações seu par fora individualista focando exclusivamente no código, não aceitando opiniões contrárias. Os Dev3 e 4, não afirmaram ter qualquer desconforto ou dificuldade.

Quando perguntados sobre conflitos com seu par, o Dev1 e Dev4 negaram a existência de qualquer conflito. O Dev3 também negou qualquer conflito, porém salientou que fez vários questionamentos para sua dupla, o que em sua opinião, não se enquadra como conflito. Já o Dev2 afirmou ter conflitos em momentos em que seu par focou apenas no código não envolvendo ele na evolução das tarefas e também que seu par não aceitou opiniões contrárias.

Todos os participantes afirmaram que a Programação em Pares traria benefícios para o modelo de desenvolvimento da empresa estudada. O Dev1 afirma que essa prática foi proveitosa e poderia ser mantida. O Dev2 apenas afirmou que traria benefícios, mas não exemplificou. O Dev3 acredita que seria necessário um período de adaptação do time para melhor aproveitamento do tempo. O Dev4 salientou que a empresa tem grande foco no aprendizado de seus colaboradores e por isso essa prática trará benefícios em decorrer da troca de conhecimento entre os desenvolvedores.

E por fim. cada participante descreveu na sua visão a experiência de programar em pares. O Dev1 relatou que traz aprendizados para programadores iniciantes, e quando bem aplicada melhora os resultados de produtividade.

O Dev2 discorreu sobre o tema, trazendo pontos positivos como: encontrar soluções difíceis de maneira rápida, facilidade de existência de insights para o desenvolvimento das tarefas, maior possibilidade de focar no problema e a facilidade de validar informações quando se trabalha em par. Entretanto abordou também pontos negativos como: frustação quando seu par é individualista e possui um nível diferente de programação. E ainda, afirmou que pode ser vantajoso apenas para o desenvolvedor menos experiente, salientando que possíveis erros e códigos ruins por parte de sua dupla, podem atrasar o desenvolvimento das tarefas. No final ressaltou a possibilidade de aproveitar programadores menos experientes na função de copiloto, com a premissa de que há situações onde o desenvolvedor é ruim para construir códigos, porém é muito bom em dar ideias e que, em seu ponto de vista, a principal função do desenvolvedor é dar ideias que permitem resolver os problemas.

O Dev3 relatou que a experiência é enriquecedora, pois há uma troca de conhecimento entre ambos os desenvolvedores. O Dev4 também classificou como enriquecedora,

principalmente para os estagiários, por dar uma maior visão do sistema como um todo e ter a possibilidade de aprender com desenvolvedores mais experientes.

De maneira geral conclui-se a partir das respostas analisadas no questionário, que todos os desenvolvedores participantes tiveram uma experiência satisfatória com a aplicação da Programação em Pares proposta. Destaca-se as respostas do Dev2 que apesar de relatar conflitos e desconfortos, se mostrou interessado no modelo e abordou premissas que trariam mais sucesso para implementações futuras.

A Programação em Pares trouxe aos participantes uma troca de conhecimento e maior agilidade nas conclusões das tarefas. Ficou evidenciado que para os desenvolvedores menos experientes Dev3 e Dev4 a aplicação foi enriquecedora o que contribui com as necessidades da empresa.

5 CONCLUSÃO

O presente trabalho buscou propor um modelo de gestão ágil de projeto para ser implementado ao modelo de desenvolvimento atual de software da empresa Rota Exata Software LTDA. A Programação em Pares possibilitou uma maior troca de conhecimento entre os programadores e trouxe mais fluidez no desenvolvimento e conclusão de tarefas complexas. Quando comparado ao modelo atual da empresa, observou-se um aumento de 13% nas tarefas concluídas e o índice de tarefas não prevista teve queda de 12%.

Primeiramente realizou-se uma revisão da literatura na área de desenvolvimento de software. Com enfoque nos métodos ágeis Kanban, Scrun e Xp, a revisão literária possibilitou conhecer a respeito dos valores e práticas dessas metodologias, entre elas destacou-se a Programação em Pares.

Neste contexto do desenvolvimento do software foi realizado um estudo de caso, apresentado à empresa Rota Exata Software LTDA, e descrevendo seu processo de desenvolvimento, detalhando o modelo e as métricas utilizados pela empresa.

Na sequência foi realizado um diagnóstico por meio das análises do resultado de sete sprints do modelo atual, e levantou-se a hipótese de que o índice de Não Previsto, aumenta de acordo com o tempo em que o desenvolvedor está na empresa, esse índice tem impacto direto com a conclusão das tarefas.

Neste cenário foi proposto a Programação em Pares com objetivo de trazer uma troca de experiência entre os desenvolvedores, ao mesmo tempo em que mais tarefas são concluídas no decorrer da semana.

O modelo proposto foi implementado pelo time de desenvolvedores back-end da empresa, onde organizou-se dois pares de desenvolvedores para executar as tarefas de um ciclo sprint. Apurou-se os resultados da implementação baseando-se nos indicadores usuais da empresa e na sequência foi relatado as experiências dos participantes por meio de um questionário dissertativo.

Desta forma, entende-se que objetivo geral do trabalho de propor um modelo de gestão ágil de projeto baseada na prática de Programação em Pares para desenvolvimento de software foi atingido.

Além disso, os objetivos específicos de: i) Revisar bibliograficamente as abordagens clássicas e ágeis de gerenciamento de projetos; ii) Aprofundar conhecimentos sobre o tema

métodos ágeis, evidenciando definições, variedades e possíveis abordagens; iii) Descrever o modelo da empresa estudada e as métricas utilizadas; iv) Implementar possíveis melhorias no modelo de gestão de projeto de software para a empresa Rota Exata Software Ltda; v) Comparar os resultados obtidos após a implementação, foram realizados.

Este trabalho traz um conteúdo relevante para a área de desenvolvimento de software, e carrega a responsabilidade de evidenciar na prática o contexto dos métodos ágeis tornando-se uma possível fonte de pesquisa pra empresas de software com modelos semelhantes ao apresentado.

Além disso, contribuiu para o conhecimento dos envolvidos na implementação e trouxe a empresa estudada uma possível melhoria aplicável na projeção de crescimento da empresa, principalmente para troca de conhecimento entre os desenvolvedores.

Quanto a limitação do estudo, sugere-se as seguintes possibilidades para trabalhos futuros:

- Aplicação de indicadores relativos ao desdobramento das tarefas nos ciclos do sprint;
- Estudo de práticas que melhorem a resolução das atividades em pares;
- Implementar o modelo proposto e um número maior de ciclos para efeito de comparação mais precisa entre os modelos, possibilitando a troca entre os pares no decorre dos sprints.

Por fim, é importante ressaltar o valor do trabalho de conclusão de curso para a formação do engenheiro. Por meio desse trabalho foi possível aprimorar o conhecimento teórico e aplica-lo na prática trazendo um maior entendimento da realidade das empresas de tecnologia.

REFERÊNCIAS

ASSOCIAÇÃO CATARINENSE DE TECNOLOGIA. **DESTAQUES**: Santa Catarina é o estado que mais cresceu em número de empresas. TECH REPORT 2020. Disponível em: https://www.acate.com.br/noticias/santa-catarina-e-o-estado-que-mais-cresceu-em-numero-de-empresas-de-tecnologia-aponta-tech-report-2020 Acesso em: 28 Mar. 2021.

AHMAD, M. O.; MARKKULA, J.; OVIO, M. **Kanban in software development:** A systematic literature review. 2013.

ALBINO, Raphael Donaire. **Benefícios alcançados através de um modelo de Gestão Ágil de Projeto em uma empresa de jogos eletrônicos**. 2013.

AMARAL, Daniel Capaldo et al. **Gerenciamento Ágil de Projetos:** aplicação em produtos inovadores. 1. Ed. São Paulo: Saraiva, 2011.

BERNARDO, Kleber. **Kanban: Do início ao fim!. Cultura ágil**. 2014. Disponível em: https://www.culturaagil.com.br/Kanban-do-inicio-ao-fim/. Acesso em: 28 Mar. 2021.

BEZERRA, Wendy S.; CONCEIÇÂO, Carla A.. **Utilização da metodologia ágil eXtreme Programming (XP) como ferramenta de gestão**: um estudo de caso numa empresa do ramo de tecnologia e serviços. 2012. Universidade Potiguar: Revista científica da escola e gestão de negócios.

BEZNOSOV, K. **Extreme security engineering:** on employing XP practices to achieve Good Enough Security without defining it. [S.l.]: ACM, 2003.

BLASCHEK, José Roberto. **Gerênciade Requisitos: O principal problema dos projetos de software**. 2011.

BOSTRÖM, G. et al. **Extending XP practices to support security requirements engineering**. In: INTERNATIONAL WORKSHOP ON SOFTWARE ENGINEERING FOR SECURE SYSTEMS, 1., 2006.

COSTA, Edes Garcia da; PENTEADO, Rosângela; SILVA, Júnia Coutinho Anacleto; BRAGA, Rosana Teresinha Vaccare. **Padrões e Métodos Ágeis: Agilidade no Processo de Desenvolvimento de Software**. 2015.

CRUZ, Jossandro Rodrigues da; GONÇALVES, Luciana Schleder; GIACOMO, Ana Paula Magalhães de Abreu. **Agile Scrum Methodology:** implementation by the nurse in an educational game on safe medication management." Revista gaucha de enfermagem 40.SPE (2019).

FARIA, Vivian Maerker. **Manual de carreira: identifique e destaque o talento que existe em você**. São Paulo: Saraiva, 2009.

FOGGETTI, Cristiano. **Gestão ágil de projetos.** São Paulo: Education do Brasil, 2014.

GHANI, I.; YASIN, I. **Extreme programming methodology**: a systematic literature. Science International, [S.l.], v. 25, n. 2, p. 215-21, 2013.

GIL, A. C. **Como elaborar projetos de pesquisa**. 5. ed. São Paulo: Atlas, 2010.

LANDIM, Henrique Farias. **Uma Abordagem de Monitoramento dos Fatores e Condições que influenciam nas Práticas Ágeis**. 2012.

LEITÃO, Michele de Vasconcelos. **Aplicação de Scrum em Ambiente de Desenvolvimento de Software Educativo**. 2010.

JUBILEU, Andrea Padovan. **Modelo de Gestão do Processo de Venda e Desenvolvimento de Software On-Demand para MPE´s**. 2008.

KERZNER, H. **Project management: a systems approach to planning, scheduling, and controlling**. Nova York: John, Wiley, 2011.

LEITÃO, Michele de Vasconcelos. **Aplicação de Scrum em Ambiente de Desenvolvimento de Software Educativo**. 2010.

LOOSEMORE, M.; RAFTERY, J.; REILLY, C.; HIGGON, D., **Risk Management in projects**. 2. ed. Abingdon: Taylor & Francis, 2011.

LOPES, L.P. **Aplicação da Metodologia Scrum em uma Área de Engenharia de Processos**/– Rio de janeiro: UFRJ/ Escola Politécnica, 2017

MEDEIROS, Manoel. **Implementando Pair Programming em sua equipe**. 2007. Disponível em: https://https://www.devmedia.com.br/implementando-pair-programming-em-sua-equipe/1694. Acesso em: 28 Mar. 2021.

MENDES, T. Geração Y: forjada pelas novas tecnologias. **Revista Brasileira de Administração**, São Paulo: CFA, n. 91, p. 52-54, nov./dez. 2012.

MORAES, Janaína Bedani Dixon. **Engenharia de Software 2 - Técnicas para levantamento de Requisitos**. 2011.

NETO, Oscar Nogueira de Souza. **Análise Comparativa das Metodologias de Desenvolvimento de Softwares Tradicionais e Ágeis**. Trabalho de Conclusão de Curso. Universidade da Amazônia. Belém: 2014.

PETERS, T. **Você é igual ao seu projeto**. Revista VOCÊ s/a., São Paulo, ano 2, n. 14, agosto 2011.

PRESSMAN, Roger S.. **Engenharia de Software: Uma Abordagem Profissional.** 7. ed. New York: The McGraw-Hill Companies, Inc, 2011.

RAMOS, Everton A.. **Metodologias Ágeis**: Extreme Programming. Maringá: Universidade Estadual de Maringá, 2013. 42 p.

RIBEIRO, Tayse Virgulino; SOUZA, Cristina D'Ornelas Filipakis. **SIDD–Scrum Iteration Driven Development:** Processo Ágil Para Desenvolvimento E Gerenciamento De Software. Anais Estendidos do XV Simpósio Brasileiro de Sistemas de Informação. SBC, 2019.

SANTOS, Rodrigo. **Estudo de caso sobre a utilização do eXtreme programming em uma pequena empresa de desenvolvimento para web**. Santa Catarina: Universidade do Sul de Santa Catarina, 2013. 53 p.

SCHWABER, Ken; SUTHERLAND, Jeff. **The Scrum Guide: The Definitive Guide to Scrum: The Rules of the Game**. 2013.

SCHWABER, Ken. **Agile Project Management with** Scrum. Redmond, Washington: Microsoft Press, 2014.

SOARES, F. S. F. **Uma estratégia incremental para implantação de gestão ágil de projeto sem organizações de desenvolvimento de software que buscam aderência ao CMMI**. 2015. 281 p. Dissertação (Ciência da Computação) — Universidade Federal de Pernambuco, Recife.

SOMMERVILLE, I. **Engenharia de Software**-9a Edição. [S.l.]: Pearson Education, 2011.

SONIA; SINGHAL, A.; BANATI, H. **FISA-XP:** an agile-based Integration of security activities with extreme programming., New York, v. 39, n. 3, p. 1-14, 2014.

TELES, Vinícius M. Um **Estudo de Caso da Adoção das Práticas e Valores do Extreme Programming.** Rio de Janeiro: Dissertação (Mestrado em Informática) - Núcleo de Computação Eletrônica, Universidade Federal do Rio de Janeiro, 2005.

VARGAS, Letícia Marques. **Gerenciamento Ágil de projetos em desenvolvimento de software: Um estudo comparativo sobre a aplicabilidade do SCRUM em conjunto com PMBOK e/ou PRINCE2.** Revista de Gestão e Projetos – GeP, v. 7, n. 3. 2016.

WÄYRYNEN, J.; BODÉN, M.; BOSTRÖM, G. **Security Engineering and eXtreme Programming:** An Impossible Marriage? In: CONFERENCE ON EXTREME PROGRAMMING AND AGILE METHODS, 4., 2004, Calgary. Proceedings... Berlin: Springer, 2005. p. 117-28.

WILDT, Daniel *et al*. **EXtreme Programming**: práticas para o dia a dia no desenvolvimento ágil de software. São Paulo: Casa do Código, 2015. 128 p.

APÊNDICE A – SPRINT IMPLEMENTADO EM PARES

Sprint Pares

Time Frontend:
Dupla 1 = Dev 1 + Dev 3 (5/dia) **= 25**
Dupla 2 = Dev 2 + Dev 4 (5/dia) **= 25**

Auxiliares dos Times
Product Owner
Scrum Master

Cronograma
Início: 29/03/2020 - Segunda
Previsto:02/04/2020 - Sexta
Término:

Definições de Tempo
1 hora: 1 ponto
¼ dia: 2 pontos
½ dia: 3 pontos
1 dia: 6 pontos

Kanban	Dono	Atividade	Pontos	Retrabalho	Realizado
Sim	Dupla 1	Ao clicar no botão de X, deve-se fazer um delete	2	Não	Sim
Sim	Dupla 1	O intermodal deve ter um botão para fazer os downloads da imagem	2	Não	Sim
Sim	Dupla 1	Painel Gestão	4	Não	Sim
Sim	Dupla 1	Painel Gestão - Aplicar filtro por usuário dos quadros	2	Não	Sim
Sim	Dupla 2	Detalhes das Rotas - Adicionar um botão de editar Rota	1	Não	Sim
Sim	Dupla 2	Detalhes das Rotas - Criar um modal para fazer a edição de uma rota	3	Não	Sim
Sim	Dupla 2	Categorias - Adicionar um campo "número de subcategorias", o qual indica o número de categorias que aquela categoria tem.	1	Sim	Sim
Sim	Dupla 2	Retirar o campo id das tabelas das páginas de: Usuários e Motoristas, Veículos, Grupos, Categorias.	2	Não	Sim
Sim	Dupla 2	Na tela de Meu Perfil, caso algum campo não tenha valor, deve ser preenchido com "Não informado".	2	Não	Sim
Sim	Dupla 2	Multas - Fazer a parte visual da tela	3	Não	Sim
Sim	Dupla 2	Estudo - Conferir como pegar a localização do celular a partir do PWA	1	Não	Sim
Sim	Dupla 2	(Aberto por Milena) Investigar - Conferir se o componente que faz a tabela (r-data) funciona a paginação.	2	Não	Sim
Sim	Dupla 2	Roteirização de Visitas agendadas - Não está aparecendo todos os documentos na listagem de visitas marcadas	3	Não	Sim
Sim	Dupla 2	Estudar - Geração de Links para telas únicas das quais não precisam de autenticação para visualizar. (Histórico de Deslocamento, Detalhes da Rota)	1	Não	Sim
Sim	Dupla 2	Investigar - Consertar Erros que ocorrem ao editar um veículo	1	Não	Sim
Sim	Dupla 1	Roteirização de Visitas Agendadas	2	Não	Sim
Sim	Dupla 1	Relatório - O totalizador de tempo	2	Sim	Sim
Sim	Dupla 1	Investigar - Minha Rota	1	Não	Sim
Sim	Dupla 1	Ao mudar a cor do tema da empresa, a cor volta para o azul.	1	Não	Sim

APÊNDICE B – QUESTIONÁRIO DISSERTATIVO

Questões	Dev1	Dev2	Dev3	Dev4
Você já teve alguma experiência anterior com a Programação em Pares?	Sim, meu primeiro contato com programação em par foi durante um curso técnico que fiz no Senai.	Sim	Sim	Sim
No desenvolvimento das tarefas, você aprendeu algo novo com seu par?	Geralmente eu aprendo algo novo, pode se dizer que em 70% das atividades desenvolvidas em par.	Sim	Sim, acabei aprendendo novos jeitos de codar e achar os erros nos meus códigos	Sim
Você teve alguma dificuldade ou desconforto em programar em par?	Não, a programação em par requisita uma boa comunicação entre os programadores, acredito que já programei em par com alguém que não estava confortável.	Sim, às vezes o outro programador pode ser um pouco individualista, escrevendo muito código sem explicar o que está fazendo ou simplesmente não aceitar opiniões contrárias.	Não, achei tranquilo,	Não
No desenvolvimento das tarefas, você teve algum conflito com seu par?	Não, sempre tive um bom programador ao meu lado durante as atividades.	Sim, quando meu par ficou escrevendo muito código sem explicar o que está fazendo ou simplesmente não aceitou opiniões contrárias	Não, somente fiz várias perguntas(dúvidas), mas acho que isso não se considera como conflito	Não
Na sua visão, a implantação da Programação em Pares traria benefícios ao modelo de desenvolvimento de software atual da empresa?	Acredito que sim, poderiamos manter essa prática ao longo de todo o backlog	Sim	Acredito que sim, porém talvez exigiria um período de adaptação do time para melhor aproveitamento do tempo	Acredito que em uma empresa como a Rota Exata que tem grande foco no aprendizado, a programação em pares é uma grande aliada pois a mesma pode ser uma meio de passagem de experiências entre os parceiros de programação.
Na sua visão como você descreve a experiência de programar em pares?	Uma experiência que traz sempre aprendizado para um dos programadores e quando bem feita traz bons resultados de produtividade.	Pair programming é a melhor maneira de encontrar uma solução difícil de maneira rápida. Quando se tem um parceiro(a) do seu lado para auxiliar o seu código, você facilita a existência de insights para o desenvolvimento da tarefa, além de poder focar unicamente no código porque já existe alguém pensando na solução para você. Ainda que é muito mais fácil validar alguma informação quando se tem alguém do lado. "Qual algoritmo é mais rápido?", "Você considera esse código legível?", "Há um jeito mais fácil de resolver isso?", "O que você acha de fazer dessa maneira?" são frases comuns em bons pair programmings Por fim, pode ser frustrante caso o seu companheiro seja um programador mais individualista ou que esteja num nível diferente do seu. Ele pode escrever códigos muito ruins e acabar atrasando o desenvolvimento, o que é vantajoso para o programador mais iniciante e desvantajoso para o programador mais experiente. É importante ressaltar que dá pra fazer aproveitar programadores menos experientes em pair programmings explicando a situação e pedindo que lhe dê ideias (as vezes o desenvolvedor é ruim para construir códigos, mas é muito bom para dar ideias e esse é a verdadeira função do dev, dar ideias) "O que você está fazendo?", "O meu jeito era o melhor", "Deixa assim mesmo, depois consertamos" ou o silêncio são frases comuns em maus pair programmings.	Acredito que seja enriquecedora, pois há uma troca de experiências muito forte para ambos os desenvolvedores.	Como grande e enriquecedora. Principalmente no meu nível que é o de estagiária a programação em par nos dá uma visão maior do sistema e também a experiência de codar com alguém mais experiente.