Faculdade de Tecnologia de Americana
Análise e Desenvolvimento de Sistema

METODOLOGIA SCRUM

THAYNARA ALVES DA SILVA

Americana, SP
2017

Faculdade de Tecnologia de Americana
Análise e Desenvolvimento de Sistema

METODOLOGIA SCRUM

THAYNARA ALVES DA SILVA

Thay_jt@hotmail.com

Projeto desenvolvendo em cumprimento curricular da disciplina Projeto de Graduação do Curso Tecnólogo em Analise de Desenvolvimento de Sistema da FATEC – Americana, sob orientação do Msc. Luiz Rodolfo Barreto da Silva

Área: Engenharia de Software

Americana, SP
2017

FICHA CATALOGRÁFICA – Biblioteca Fatec Americana - CEETEPS
Dados Internacionais de Catalogação-na-fonte

S584m SILVA, Thaynara Alves da

Metodologia scrum. / Thaynara Alves da Silva. – Americana: 2017.

90f.

Monografia (Curso de Tecnologia em Análise e Desenvolvimento de Sistemas) – Faculdade de Tecnologia de Americana – Centro Estadual de Educação Tecnológica Paula Souza

Orientador: Prof. Ms. Luiz Rodolfo Barreto da Silva

1. Scrum – software de projetos 2. Desenvolvimento de software I. SILVA, Luiz Rodolfo Barreto da II. Centro Estadual de Educação Tecnológica Paula Souza – Faculdade de Tecnologia de Americana

CDU: 681.3.077

THAYNARA ALVES DA SILVA

METODOLOGIA SCRUM

Projeto desenvolvendo em cumprimento curricular da disciplina Projeto de Graduação do Curso Tecnólogo em Analise de Desenvolvimento de Sistema da FATEC – Americana, sob orientação do Prof. Msc. Luiz Barreto

Área: Engenharia de Software

Americana, 27 de junho de 2017

Banca Examinadora

Luiz Barreto da Silva (Orientadora)
Mestre
FATEC – Americana

Antônio Alfredo Lacerda
Especialista
FATEC – Americana

Alberto Martins Júnior
Mestre
FATEC – Americana

AGRADECIMENTOS

Em primeiro lugar, a meu orientador Barreto que, desde o início, acreditou em meu trabalho. Obrigado a todos os professores pela dedicação e ajudar quando mais necessitei, sempre incentivando para a finalizar mais essa etapa da minha vida.

DEDICATÓRIA

Aos meus professores, família e namorado Jhonatan

RESUMO

O crescimento em produções de sistemas nos últimos anos, teve muitas mudanças nas metodologias das clássicas que são chamadas de “metodologias pesadas”, com uma quantidade alta de documentações e sem poder ocorrer mudanças no projeto no meio do percurso pelo custo da alteração. A metodologia ágil de desenvolvimento de software tem tido grande evolução no Brasil, pois serve como ferramenta na busca por melhores resultados em produtos de software, fornecendo um gerenciamento mais prático focado mais na programação, mas não esquecendo da documentação.

Palavras Chave: Metodologia ágil, Scrum, Engenharia de Software.

ABSTRACT

The growth in systems productions in recent years has had many changes in the methodologies of the classic ones that are called "heavy methodologies", with a high amount of documentation and without being able to happen changes in the project in the middle of the route by the cost of the change. The agile methodology of software development has had great evolution in Brazil, since it serves as a tool in the search for better results in software products, providing a more practical management focused more on programming, but not forgetting the documentation.

Keywords: Agile Methodology, Scrum, Software Engineering.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE GRÁFICO

1 INTRODUÇÃO

A evolução da computação contribuiu para o crescimento dos sistemas de informações. Neste processo, para desenvolvermos sistemas passamos pela eletrônica, circuitos integrados e transistores. Na evolução dos computadores, temos a engenharia de *software,* que agregou com a mudança na criação de programas e sistemas operacionais.

Apesar do grande avanço tecnológico, muitos projetos de *software* não conseguem atender os prazos estipulados, e nem contém todas as funcionalidades e especificações requeridas, não sendo finalizados dentro dos prazos combinados, o que resulta em valores maiores, ou menos funcionalidades do que as necessárias para o sistema. Muitos desses problemas são causados devido a metodologia utilizada no desenvolvimento de *software*.

Processos voltados à documentação acabam limitando os desenvolvedores, além disso, muitas empresas não têm recursos suficientes para essas metodologias consideradas pesadas, principalmente as pequenas empresas, que muitas vezes acabam não utilizando metodologia alguma no desenvolvimento do *software*, gerando um verdadeiro caos dentro do sistema da empresa.

A metodologia ágil veio para simplificar o processo de desenvolvimento de *software*, principalmente para pequenos projetos, tendo como foco as pessoas e não os processos, preocupando sempre com a implementação ao invés da documentação. Outra qualidade dessa metodologia, é a maneira como ela se adapta às situações E aos problemas que poderão ocorrer.

Dentre as metodologias ágeis, uma das mais utilizadas é o *Scrum*, voltada para projetos pequenos, e tem como foco encontrar formas flexíveis de desenvolver *softwares* em um ambiente em constante mudança.

O objetivo principal desse trabalho, é realizar a implantação a partir da utilização do método ágil *Scrum.* Vamos desenvolver um sistema utilizando a referida metodologia para o desenvolvimento da documentação, programação e finalização do processo, com um questionário para saber a aceitação da equipe. E um acompanhamento dos processos, a fim de verificar. A implantação de uma metodologia ágil para produção de software.

A implantação de uma metodologia ágil permite um melhor controle sobre as atividades desenvolvidas pela equipe? O objetivo específico é estudar a metodologia ágil *Scrum*, adaptá-la para a realidade do projeto de desenvolvimento, implantar o *Scrum* em um projeto prático e analisar os impactos positivos e negativos que a aplicação da metodologia causou no mesmo.

JUSTIFICATIVA

A justificativa para elaboração desse projeto será realizada em um trabalho acadêmico de desenvolvimento de *software*. A mesma possui uma equipe formada por quatro participantes, que por sua vez desempenham papéis variados para o cumprimento do mesmo.

Será implantado a metodologia *Scrum* para o desenvolvimento de *software,* para gerenciamento de processos, que envolverá desde a análise de requisitos até a implantação do sistema. Sendo assim, será utilizada a metodologia *Scrum* pelo grupo.

Neste contexto, a atividade do projeto estará sendo realizada de forma mais padronizada, nas questões de definição e distribuição das tarefas, com a experiência de cada integrante da equipe, para o maior aproveitamento dos conhecimentos de cada um, resultando em um progresso melhor em relação ao projeto e aos canais de comunicação entre o grupo.

A falta de padronização para guiar a equipe algumas vezes causa problemas na troca de informações, o que é extremamente prejudicial, visto isso será estabelecido a metodologia para verificar seus resultados e as expectativas de cada integrante da equipe para analisar mais sobre a parte prática na metodologia *Scrum,* no projeto.

Outro fator relevante para a realização deste projeto de implantação de uma metodologia de desenvolvimento de software é a falta de documentação do *software* desenvolvido, que não acontece por falta de interesse por parte da equipe, mas sim pela falta de instrução, organização do tempo e conhecimento para desempenhar esse tipo de tarefa. Buscando solucionar esses problemas, foi escolhida a metodologia ágil *Scrum* para ser adotada no projeto, pela boa aceitação pela equipe

METODOLOGIA

Para atingir os objetivos propostos, foram realizadas as seguintes etapas:

A primeira etapa constituiu em uma revisão bibliográfica sobre as metodologias práticas, com foco no *Scrum*, visando identificar quais conceitos e práticas podem e devem ser aplicados;

A segunda etapa objetivou realizar a escolha de uma metodologia ágil para se colocar em prática no projeto;

Durante a terceira etapa foi feito um planejamento de implementação da metodologia no projeto com a equipe;

E, na quarta e última etapa foram avaliados os impactos da metodologia *Scrum* no projeto, usando para isso um questionário para análise sobre a metodologia no desenvolvimento de *software*.

No capítulo 2 é apresentado a história da computação e engenharia de software e sobre a crise de software.

No capítulo 3 abordamos sobre as metodologias clássicas como cascata, evolucionário, espiral e incremental.

No capítulo 4 é apresentado a metodologia ágeis, focando no método Scrum, que vem se destacando cada vez mais.

No capítulo 5 é feito o estudo de caso, utilizando a metodologia ágil Scrum, na implementação do desenvolvimento.

No capítulo 6 é demostrado o resultado na utilização no estudo de caso com pontos positivos e negativos no desenvolvimento.

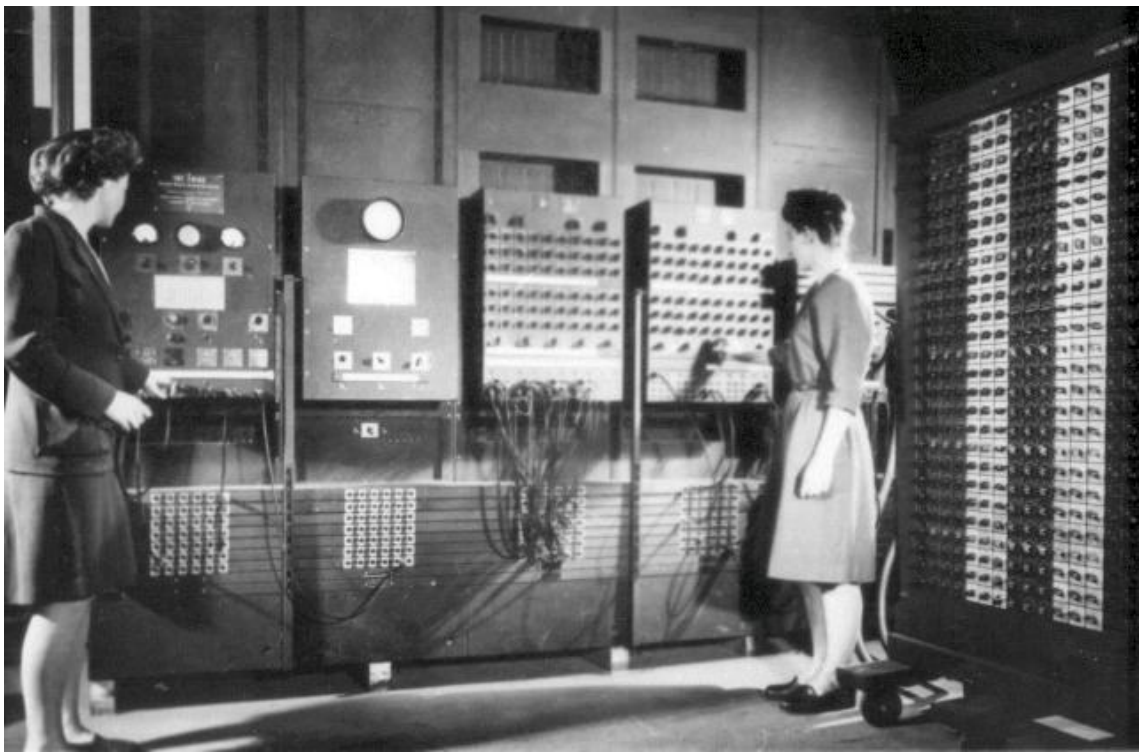
2 PROCESSO DE DESENVOLVIMENTO DE SISTEMAS

Nesse capitulo é dada uma visão geral sobre evolução da computação, história da engenharia *software* e crise de *software*.

2.1 EVOLUÇÃO DA COMPUTAÇÃO

Durante a década de 60, o mundo presenciou grandes evoluções tecnológicas no período da guerra fria. Um dos principais avanços, foi a informática que apresentava seu grandioso e inovador produto, o computador. Como pode ser observado na Figura 1, o computador era robusto, consumia muita energia e tinha pouca funcionalidade. Hoje quando comparamos os computadores que, além de terem dispositivos de *hardware* menores e cada vez mais eficientes, com a evolução da tecnologia, foram agregados, consecutivamente, como ferramenta multiuso para as mais distintas finalidades.

Figura 1 - ENIAC



Fonte: https://sites.google.com/site/historiasobreossitesdebusca/primeiro-computador-do-mundo

O Engenheiro Civil Konrad Zuse 1910 foi o inventor da primeira máquina de cálculo, que controlava automaticamente. E assim foi percebido o primeiro problema, que era a ausência de armazenamento dos dados. Zuse começou a entender que a calculadora precisava de um controlador, para armazenar os dados e um dispositivo de cálculo aritmético. Já o sistema operacional Z3 quando foi lançado, executava de três a quatro adições por segundo e multiplicava dois números em quatro ou cinco segundos. (MAGELA, 2006)

De acordo com Rogério, o UNIVAC I que se observa na Figura 2 foi o primeiro computador a ser produzido para comercialização. Esses eram os computadores da primeira geração. Os computadores de segunda geração só tiveram pequenos avanços com os transistores. Já os da terceira geração foram marcados pelos circuitos integrados. Os computadores da quarta geração até os atuais, tem milhões de artefatos eletrônicos, em um pequeno espaço, que se denomina chip. (MAGELA, 2006)

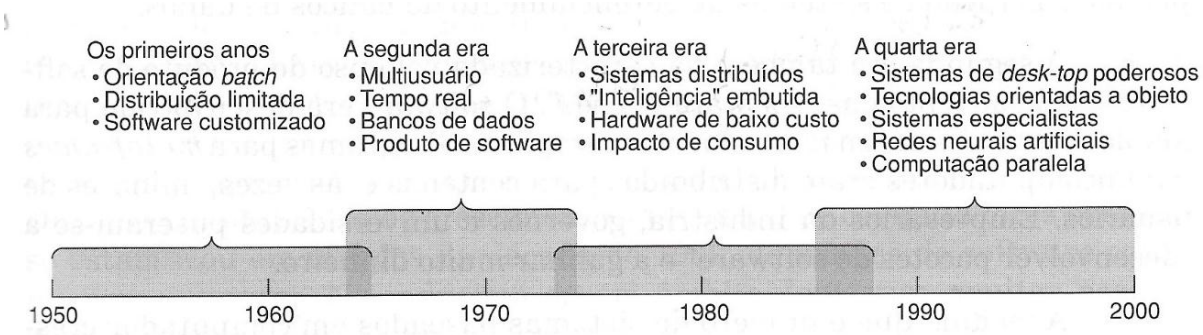
Figura 2 – UNIVAC I

Fonte: https://en.wikipedia.org/wiki/UNIVAC_I#/media/File:Univac_I_at_CHM.agr.jpg

2.2 HISTÓRIA DA ENGENHARIA DE SOFTWARE

Na Figura 3 é descrita a evolução do *software*. No começo do desenvolvimento computacional, o *hardware* teve grandes mudanças, diferentemente do software, que não era visto da mesma maneira naquela época. A programação era complexa e sem grandes métodos para uma aplicação. O desenvolvimento não tinha um gestor para acompanhar a produção do sistema e só começavam a analisar quando já estava sem tempo e com alto custo a entrega do produto. Utilizavam uma orientação batch (em lote) para o desenvolvimento do sistema. Com alguns sistemas sendo interativos, como o primeiro de reservas da American Airlines, com uma interação em tempo real. Mas na maioria das vezes o *hardware* era programado para um programa específico. (PRESSMAN, 2007)

Figura 3 - **A EVOLUÇÃO DO *SOFTWARE***



Fonte: Pressman, 2007, p. 5

Nos primeiros anos, o *hardware* era desenvolvido para todos. Já o software era produzido sob encomenda e limitado. Na criação do *software* não havia documentação e o programador criava e consertava esse erro por não haver muitas demissões dos funcionários. (PRESSMAN,2007)

> Por causa desse ambiente de *software* personalizado, o projeto era um processo implícito realizado no cérebro de alguém, e a documentação muitas vezes não existia. Durante os primeiros anos, aprendemos muito sobre a implementação de sistemas baseados em computador, mas relativamente pouco sobre engenharia de sistemas de computador. Por justiça, entretanto, devemos reconhecer os muitos surpreendentes sistemas baseados em computador

> desenvolvidos durante essa época. Alguns deles permanecem em uso até hoje e constituem feitos que são um marco de referência e que continuam a justificar a admiração. (PRESSMAN,2007, p. 6)

Na segunda evolução, chega-se em uma nova era onde existem vários usuários acessando ao mesmo tempo. Com a parte que já coletava, analisava e transformava os dados. Com um armazenamento *online* em bancos de dados. (PRESSMAN, 2007)

> A segunda era também foi caracterizada pelo uso do produto de software e pelo advento de “software houses”. O software era desenvolvido para ampla distribuição num mercado interdisciplinar. Programas para *mainframes* e minicomputadores eram distribuídos para centenas e, às vezes, milhares de usuários. Empresários da indústria, governos e universidades puseram-se a “desenvolver pacotes de software” e a ganhar muito dinheiro. (PRESSMAN, 2007, p. 6)

Com o crescimento no desenvolvimento de software e com muitas novas instruções na programação. Quando surgia uma falha ou uma mudança para ser adaptada ao hardware do usuário, era necessário algo chamado de manutenção de software. Chegamos ao ponto de alguns sistemas chegarem na fase da "crise de software". (PRESSMAN, 2007)

A terceira geração estava com muitos computadores logado ao mesmo tempo com atividades distintas. Com um aumento da complexidade do sistema por uma interação real com aumento da demanda do desenvolvedor e na complexidade no desenvolvimento do software exigindo mais dos seus programadores. (PRESSMAN, 2007)

> A terceira era também foi caracterizada pelo advento e generalizado uso de microprocessadores, computadores pessoais e poderosas estações de trabalho (workstations) de mesa. O microprocessador gerou um amplo conjunto de produtos inteligentes – de automóveis a fornos de microondas, de robôs industriais a equipamentos para diagnóstico de soro sanguíneo. Em muitos casos, a tecnologia de

> software está sendo integrada a produtos por equipes técnicas que entendem de hardware mas que frequentemente são principiantes em desenvolvimento de software. (PRESSMAN, 2007, p. 7)

Com o aumento do número de computadores pessoais, há um crescimento no número de *softwares* para utilização, não só mais nas empresas na área de negócios, mas, agora para uso pessoal e utilização descontraída com uma acessibilidade do uso.

Na quarta geração, há a criação da programação orientada a objeto, com mais interação com o usuário, também já saindo da realidade que tínhamos na última geração, agora os *desktops* mais poderosos para uso pessoal, com sistemas especialistas para área específicas com uma inteligência artificial e com um conhecimento de sistema nervoso para rede neutra-artificial com capacidade semelhante ao do ser humano. (PRESSMAN, 2007)

E na quinta geração, com um design completamente inovador, com seu software com um alto desempenho de poder. Cada vez mais o software está evoluindo. O hardware, não está tendo o mesmo avanço, o que está gerando softwares sofisticados em equipamentos defasados. A limitação para a construção de um programa é muito pequena para o aumento da demanda do mercado de tecnologia. Por haver muita correria na produção de novos aplicativos, são deixadas muitas falhas e projetos que não agregam toda sua funcionalidade inicial programada para o software que não chegam a ter os recursos necessários.

Conforme a menção feita por Wilson, o software é a peça de um sistema de informação. Ele é um meio central: realiza estruturas complicadas e flexíveis que trazem funções, utilidade e valor ao sistema. Mas outros componentes são indispensáveis: as plataformas de hardware, os recursos de comunicação de informação, os documentos de distintas naturezas, as bases de dados e até os procedimentos manuais que se integram aos automatizados. (PAULA FILHO, 2009)

Segundo SOMMERVILLE, todos os países, hoje em dia, estão sujeitos à sistemas complexos aperfeiçoados em computadores. Infraestruturas e serviços nacionais contam com sistemas baseados em computadores, e a maior parte dos produtos elétricos contém um computador e um software de controle. Mas não se pode aumentar o custo em valores exorbitantes na produção do software para que seja possível manter as economias do mundo todo. (SOMMERVILLE, 2010)

2.3 CRISE DE SOFTWARE

Segundo os estudos para Koscianski e Soares, os fatores negativos na qualidade de um projeto era sua complexidade, que tinha a ver com uma especialidade bastante simples: o tamanho das especificações. (KOSCIANSKI; SOARES, 2007)

Conforme Koscianski e Soares (DIJKSTRA, 1972 apud KOSCIANSKI; SOARES, 2007), com o avanço acelerado da tecnologia aumentando a produção de *software*. Num período pequeno o hardware teve muita modificação, ajudando criar programas mais complexos. Com um crescimento exorbitante em pouco tempo que estava só construindo casas e pequenos prédios de 2 ou 3 andares, que passaram a tarefa de construir grandes arranha-céus como se observassem repentinamente.

> Segundo Koscianski e Soares A maior causa da crise do software é que as máquinas se tornaram várias ordens de magnitude mais potentes! Em termos diretos, enquanto não havia máquinas, programar não era um problema; quando tivemos computadores fracos, isso se tornou um problema pequeno e agora que temos;computadores gigantescos, programar tornou-se um problema gigantesco. (DIJKSTRA, 1972 apud KOSCIANSKI; SOARES, 2007, p. 21)

Para Pressman não adianta só arrumar os programas já criados e sim que muitos já passaram da hora de serem recomeçados do zero para melhor desempenho. E porque “A aplicação ainda funciona” por isso que não temos que melhorar os sistemas. Existe muita concorrência, que será a cada dia mais intensa e está mais perto do que se pode imaginar. Em todos os países, existem profissionais altamente determinados, educados e de valor relativamente baixo com seu conhecimento e uma rapidez para a nova geração de tecnologia no mercado. (PRESSMAN, 2007)

Com o avanço da tecnologia as empresas não podem ficar para trás e tem que evoluir constantemente, mas para isso, elas devem observar seus prós e contra para uma análise melhor do sistema. Segundo Pressman (2007), muitas vezes contratam uma empresa terceirizada para a manutenção que tem um custo mais baixo e chegam a demitir a equipe já formada. Porém, uma vez que essa

intenção tem uma meta com êxito, não demora muito para que as vantagens vão para fora.

O hardware de computadores não são mais a fonte principal para tecnologia e o software continua sendo uma indústria em expansão. De acordo com Feigenbaum e McCorduck os Estados Unidos não podem nem querem perder o lugar na área da tecnologia. Apesar disso, nada fazemos a respeito do envelhecimento de nossa indústria de software. E chega a fazer contado com terceiros do outro lado do mundo por ser mais viável que ser apoiada pelo governo. (PRESSMAN, 2007)

3 METODOLOGIA DE DESENVOLVIMENTO

Conforme Magela (2006, p. 23), “Entendemos como Engenharia a aplicação de um conjunto de técnicas e métodos não necessariamente derivados de uma ciência em particular, como nos prova a história humana”. Já “Software é um conjunto de artefatos gerados na solução de um problema computacional que tem como artefato principal um arquivo binário executável em um computador”. Desta forma “Engenharia de *software* é um conjunto de técnicas, métodos, ferramentas e processos utilizados na especificação, construção, implantação e manutenção de um software que visa a garantir a gerência, controle e a qualidade dos artefatos gerados através de recursos humanos”. Assim “A Engenharia de software visa garantir que o processo empregado seja suficiente e garanta que o software em todo seu ciclo de vida seja passível de gerência e controle e que possua a qualidade esperada”.

Segundo Pressman (2007, p. 31), “A engenharia de software é um rebento da engenharia de sistemas e de hardware. Ele abrange um conjunto de três elementos fundamentais – métodos, ferramentas e procedimentos – que possibilita ao gerente o controle do processo de desenvolvimento do software e oferece ao profissional”. Já a engenharia são várias fases para desenvolver o sistema. Um modelo de engenharia de software é indicado tendo-se como apoio a natureza do projeto e da aplicação, os métodos e as ferramentas a consistir em usados, os controles e os produtos que devem ser entregues.

De acordo com Sommerville (2010, p. 5), “A engenharia de software é uma disciplina de engenharia relacionada com todos os aspectos da produção de software, desde os estágios iniciais de especificação do sistema até sua manutenção, depois que este entrar em operação”.

A Engenharia de software é uma disciplina que agrupa metodologias, métodos e ferramentas a ser empregadas, desde a percepção do problema até o momento em que o sistema desenvolvido deixa de ser operacional, ter em vista resolver problemas essenciais ao processo de desenvolvimento e ao produto de software. (SOMMERVILLE, 2010)

O objetivo da Engenharia de software é auxiliar no processo de produção de software, de forma que o processo de origem a produtos de alta qualidade, produzidos mais rapidamente e a um custo cada vez menor. A Engenharia de software segue o conceito de disciplina na produção de software, fundamentado nas metodologias, que por sua vez seguem métodos que utilizam de ferramentas automáticas para englobar as principais atividades do processo de produção.

> Segundo Sommerville (2010, p.43), diversidade dos processos de software. Não existe um processo ideal, e várias organizações desenvolveram abordagens inteiramente diferentes para o desenvolvimento de software. Os processos evoluíram para explorar as capacidades das pessoas em uma organização e as características específicas dos sistemas que estão sendo desenvolvidos. No caso de alguns sistemas, como os sistemas críticos, é necessário um processo flexível e ágil, é provavelmente mais eficaz.

Já faz alguns anos que o desenvolvimento de software deixou de ser sinônimo apenas de código. Hoje em dia, sabe-se que é imprescindível a utilização de uma metodologia de trabalho. Mas o que é necessariamente uma metodologia de software? Entende-se por metodologia, como a maneira – forma – de se utilizar um conjunto coerente e coordenado de métodos para atingir um objetivo, de modo que se evite, tanto quanto possível, a subjetividade na execução do trabalho. Ministrando um roteiro, um processo dinâmico e interativo para desenvolvimento estruturado de projetos, sistemas ou software, visando à qualidade e produtividade dos projetos.

Segundo Sommerville (2010, p.43),” Embora não exista um processo de software ‘ideal’, existe espaço para aprimoramento do processo de software em várias organizações”. Os processos de software podem ser aprimorados por meio da padronização de processo, na qual a diversidade de processos de software ao longo da organização é reduzida.

Muitas vezes, o uso de uma metodologia é encarado como cerceamento da criatividade dos técnicos, ou como, acréscimo de burocracia, leia-se muita documentação, por muitos tidos como desnecessário a construção de software. Uma

metodologia não deve limitar a criatividade profissional, mas deve ser um instrumento que determine um planejamento sistemático, que harmonize e coordena as áreas envolvidas. O que limita a criatividade não é a metodologia, mas os requisitos de qualidade e produtividade de um projeto.

Para Sommerville (2010, p.43), “A padronização é também um passo inicial importante na introdução de novos métodos, técnicas e boas práticas de engenharia de software”.

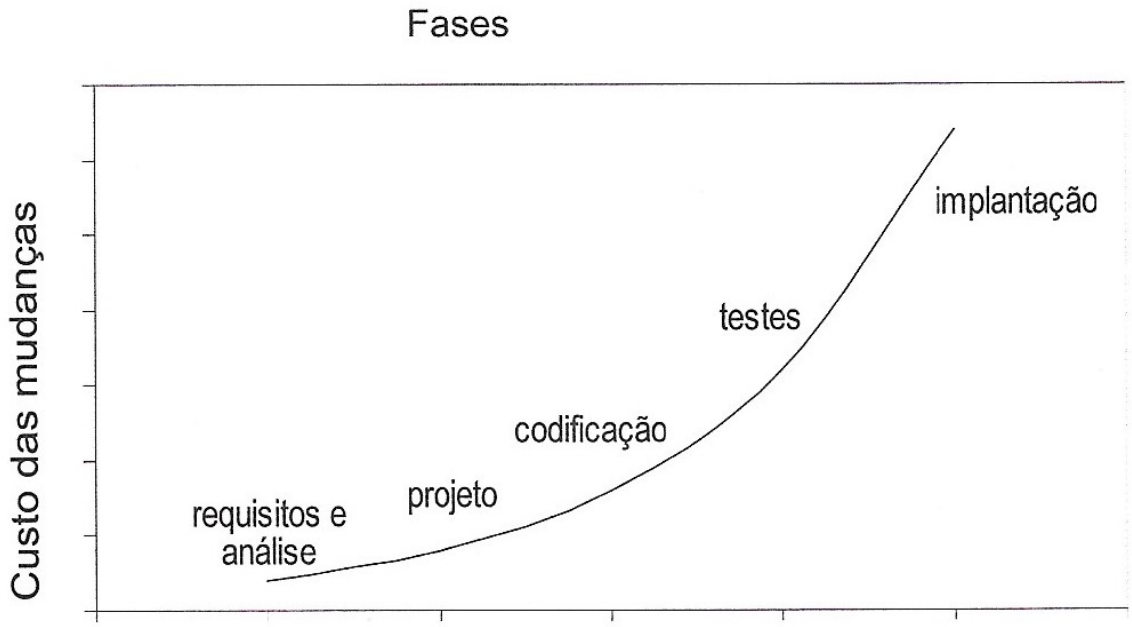
3.1 METODOLOGIAS CLÁSSICAS

A metodologia clássica também conhecida como cascata ou sequencial foi a primeira metodologia publicada de desenvolvimento de software. O modelo tem uma sequência de etapas para o seu desenvolvimento. Que cada etapa tem anexa ao seu final uma documentação que sempre passa pela aprovação antes de passar para próxima etapa. As etapas do modelo clássico são definição de requisitos, analise e projeto do software, implementação e teste unitário e teste do sistema, implantação e manutenção. (Koscianski, Soares, 2007)

De acordo com Koscianski e Soares (2007) as fases do modelo Clássico não tem uma flexibilidade, por exemplo no modelo cascata após a etapa de desenvolvimento não se prevê alterações nas especificações, já o modelo espiral é admitido retorno às etapas anteriores, porém não pode haver a execução paralela entre as etapas. Isto já e permitido em um projeto de engenharia concorrente ou diferentes componentes do produto sejam construídos de maneiras independentes.

Quando ocorre alteração em projetos com a metodologia Clássica os custos podem chegar a cem vezes maiores depois que o software está pronto, por isso para se utilizar o modelo Clássico só quando os requisitos forem estáveis. A figura 4 mostra o custo de uma modificação torna-se mais elevado com o mais final do projeto vamos está maior o custo. (Koscianski, Soares, 2007)

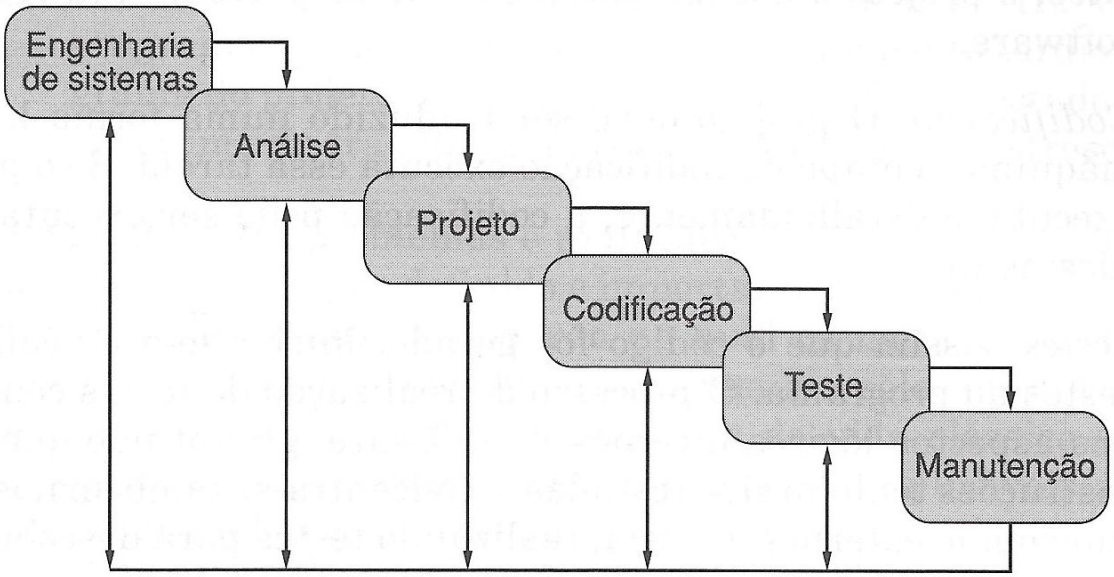
Figura 4 - Ilustração do custo de modificação no modelo clássico



Fonte: Koscianski, Soares, 2007, p. 192

Até o início da década de 1990, o modelo Clássico era quem liderava a forma de desenvolvimento de software, apesar dos julgamentos dos pesquisadores na época, que identificavam muitos problemas com a adoção dessa sequência de tarefas. A figura 5 mostra o ciclo de vida de um modelo Clássico.

Figura 5 - Ciclo de vida clássico



Fonte: Pressman, 2007, p. 33

Segundo Pressman (2007) o modelo Clássico passa por algumas fases para o desenvolvimento de software que são:

- Análise e engenharia de sistemas: Para o software funcionar é imprescindível a integração do software com outros elementos de hardware, banco de dados e pessoas, por isso nesta etapa é feito uma coleta dos requisitos no nível do sistema, uma pequena parte de projeto e também uma análise de alto nível.

- Análise de requisitos de software: Nesta fase é feito uma coleta mais rígida dos requisitos e feito uma análise por parte dos engenheiros “analista”, que por sua vez tem que compreender as informações para preparar o desempenho e interface para o sistema. Os requisitos têm que ser revisados para o cliente e documentador.

- Projeto: Nessa parte do projeto temos vários passos começando com estrutura de dados, arquitetura de software, detalhes procedimentais e caracterização de interface. Nessa parte vamos fazer todo detalhamento do software.

- Codificação: Nessa fase do projeto vamos fazer todo o código do software.

- Testes: Quando termina todo código, começa a realização de testes no software. Nessa fase garantimos que toda codificação e funcionalidade estejam funcionando, corrigindo todos os erros e garantindo que tudo que foi exigido esteja em concordância com os resultados.

- Manutenção: Após a finalização do software, pode ocorrer alguns erros ou alterações que também fazem parte dessa fase, porem as alterações são feitos no software já vivente e não criado um novo.

3.1.1 CASCATA

Segundo Sommerville (2010), o primeiro modelo de processo de desenvolvimento de software revelado originou-se de processos mais gerais na engenharia de sistema. Isso é mostrado na Figura 6. Temos a conexão de uma fase para outra, que esse modelo e conhecido como cascata ou ciclo de vida do software. As principais práticas do modelo demonstram as atividades fundamentais de desenvolvimento:

- Análise e definições de requisitos: Todos requisitos, serviços e objetivos são definidos com uma reunião com o usuário do sistema. Portanto, com toda a definição vai servir para uma especificação do sistema.

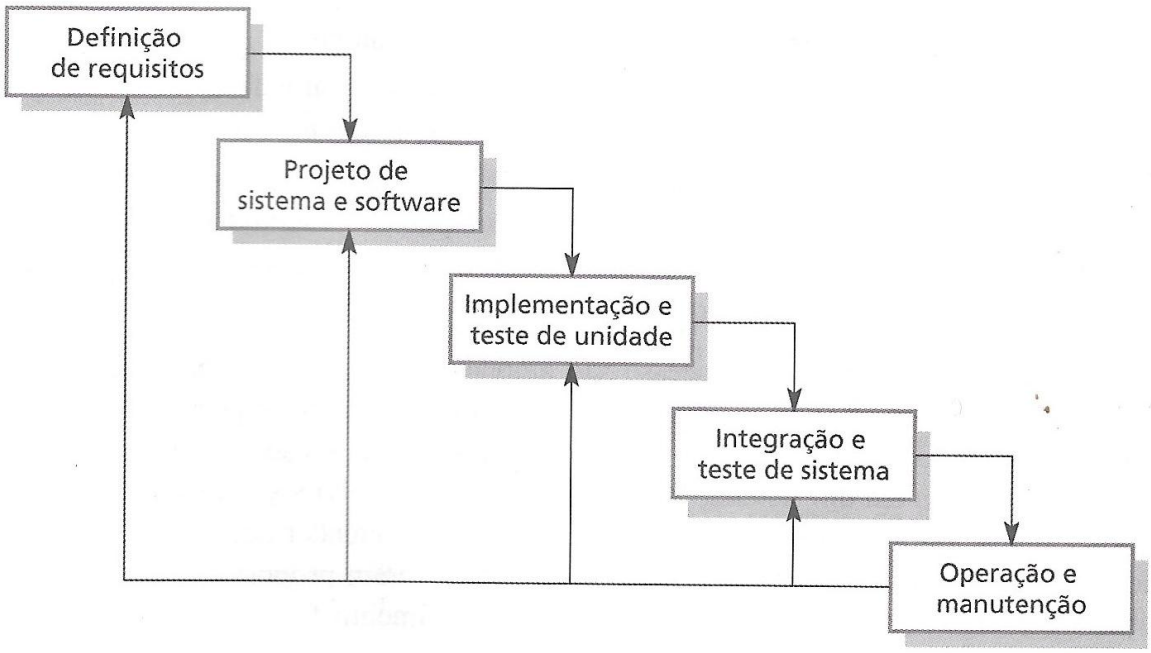
- Projetos de sistema e software: No processo de projeto de sistema divide os requisitos sistemas de hardware ou de essenciais do sistema de software e suas afinidades.

- Implementação e teste de unidade: Durante essa fase, o projeto de software é realizado a implementação ao todo ou por unidade. O teste unitário abrange a comprovação de que cada unidade atende à sua especificação.

- Integração e teste de sistema: As unidades individuais do programa ou os programas são integrados e testados como um sistema completo para avaliar que os requisitos de software foram atendidos. Após os testes, o sistema de software é autorizado para o cliente.

- Operação e manutenção: Essa e a fase mais longa do ciclo de vida, o sistema é instalado e colocado em operação. A manutenção envolve a correção de erros não detectados nos estágios anteriores da fase de planejamento e desenvolvimento, no aperfeiçoamento da implementação das unidades de sistema e no aumento dos serviços de sistema à medida que novos requisitos são identificados.

Figura 6 – Ciclo de vida de software



Fonte: Sommerville, 2010, p. 44

Em princípio, o resultado de cada fase incide de um ou mais documentos aprovados (‘assinados’). A fase seguinte só pode começar quando termina toda a fase anterior. Na prática, esses estágios se juntam e trocam informações entre si. Durante o projeto, são identificados problemas com requisitos, durante a codificação, são encontrados problemas de projeto. O processo de software não é um modelo

simples que envolve uma sequência de atividades de desenvolvimento. (SOMMERVILLE, 2010)

Devido ao alto custo e a aprovação em documentação com o retrabalho em cada fase. Portanto, após algumas alterações, é normal suspender partes do desenvolvimento, como a especificação, e prosseguir com os estágios futuras do desenvolvimento. Os problemas são ignorados ou reprogramados. Com o atraso pode ocorrer mudanças nos requisitos e não atender aos requisitos do usuário. Isso pode levar a sistemas mal estruturados, pois os problemas foram alterados sem um desenvolvimento apropriado.

Durante a fase final do ciclo de vida, o software é colocado em uso. Erros e omissões nos requisitos são descobertos, e novas necessidades são identificadas. O sistema deve evoluir para continuar sendo útil com essa mudança. o que pode implicar em repetição de estágios anteriores do processo.

As vantagens consistem na documentação feita em cada etapa do desenvolvimento do sistema e sua aceitação ao outro modelo de engenharia. Seu problema é a divisão de cada etapa do projeto. O compromisso deve ser assumido no início do processo, o que deixa difícil alteração em requisitos futuros.

O modelo cascata deve ser utilizado quando não ocorrem mudanças radicais nos requisitos e tem um alto conhecimento nos requisitos. Por outro lado o modelo cascata pode ser utilizado com outro modelo de engenharia. Ainda são utilizados nessa abordagem para desenvolvimento de software, particularmente quando fizer parte de um projeto maior de engenharia do sistema. (SOMMERVILLE, 2010)

De acordo com Magela (2006), no modelo cascata e difícil descobrir erros na análise de sistema e na maiores contradições e inconsistência são encontradas na programação. Como está em constante modificação, sendo atualmente quase impossível a utilização dessa abordagem.

3.1.2 EVOLUCIONÁRIO

Segundo Sommerville (2010), O modelo evolucionário tem como implementação inicial, apresentando o resultado ao usuário, e possibilita desenvolver várias versões até que seja desenvolvido um sistema adequado Figura 7. As atividades de especificação, desenvolvimento e validação são interpostas, em vez de serem separadas, com feedback rápido que altera as atividades.

Temos dois tipos fundamentais de desenvolvimento evolucionário:

- Desenvolvimento exploratório: nesse processo temos o contato com o cliente para entender os requisitos e desenvolver o sistema final. O desenvolvimento se inicia com a compreensão do sistema. E a evolução com novas características para compor o sistema pelo cliente.

- Prototipação: no processo de desenvolvimento evolucionário é entender os requisitos do cliente e, a partir disso, compreender melhor as definições de requisitos para o sistema. O protótipo é utilizado quando não temos certeza se os requisitos são mal-entendido pelo cliente.

O modelo evolucionário para desenvolvimento de software é usualmente mais eficaz do que o modelo em cascata na construção de sistemas que aprovem às necessidades imediatas dos clientes. A vantagem da abordagem evolucionária e que pode fazer alteração de forma incremental. Quando o cliente entende melhor o problema, isso ajuda no desenvolvimento do sistema. Na engenharia do processo evolucionário temos dois problemas:

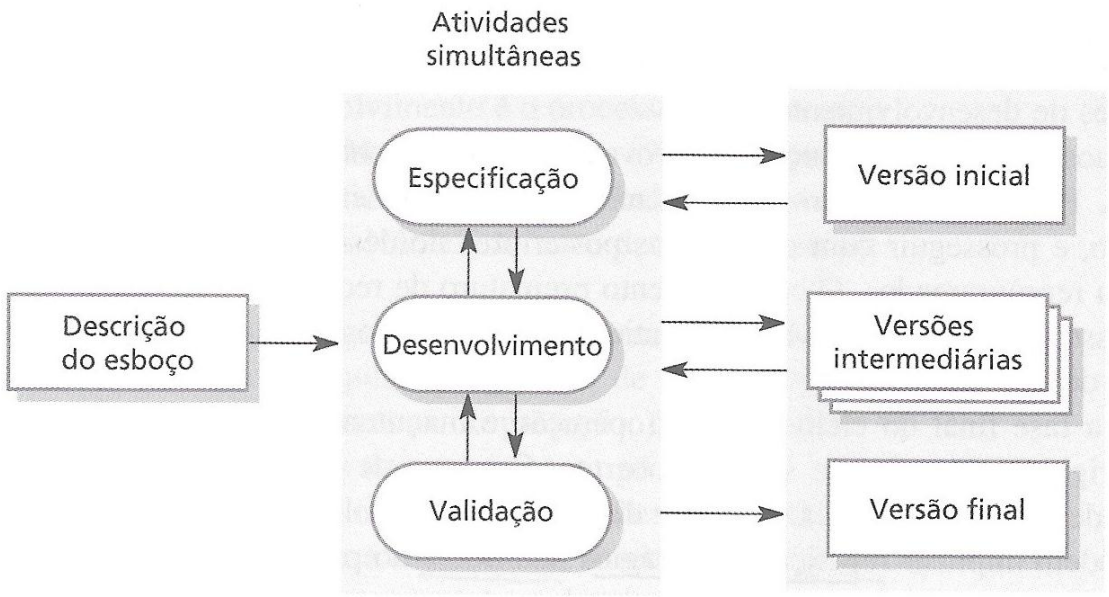
- O processo não é visível: Não tem como medir o progresso e o sistema e desenvolvido rápido, não é viável de maneira econômica produzir documentos que reflitam cada versão do sistema alterado.

- Os sistemas são frequentemente mal estruturados: Pois temos muita mudança corromper a estrutura do software. Com muita alteração se torna difícil e cara.

A abordagem evolucionaria e indicada para sistema simples com (até 500 linhas de códigos). Há problemas em aconselhar para sistemas complexos de grande porte e de longo ciclo de vida. É difícil entender toda distribuição para uma equipe integrar o desenvolvimento nesse modelo. (SOMMERVILLE, 2010)

Para o desenvolvimento para grande porte, pode fazer um processo misto com evolucionário e o cascata envolvendo o protótipo, usando o evolucionário para as incertezas na especificação do sistema, e na alteração do sistema com uma abordagem estruturada, já as partes bem entendidas do sistema o modelo em cascata e na interface um modelo exploratório que são difíceis de ser especificados com o usuário.

Figura 7 – Desenvolvimento evolucionário

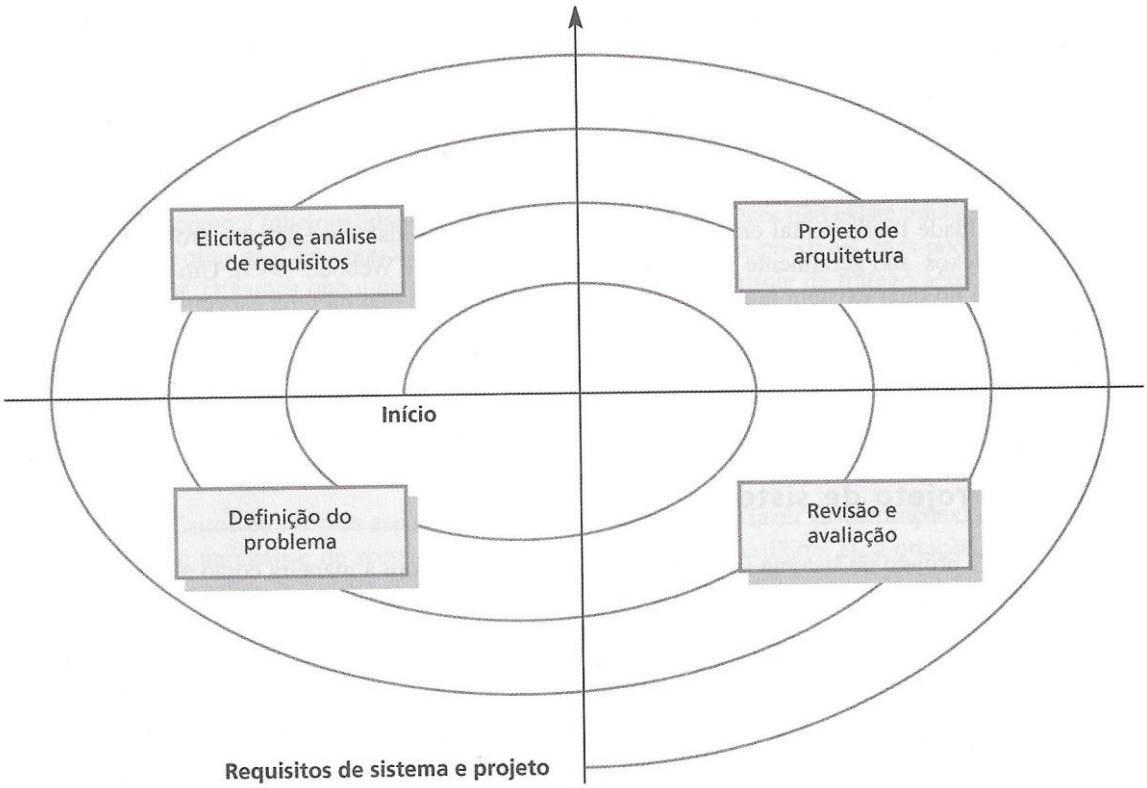


Fonte: Sommerville, 2010, p. 46

3.1.3 ESPIRAL

O modelo espiral tem sua restrição como qualquer outra metodologia, à medida que o desenvolvimento avança, você pode encontrar problemas com os requisitos existentes e novos requisitos podem aparecer. Portanto, um modo de pensar nesses procedimentos ligados é como uma espiral, conforme mostrado na Figura 8. (SOMMERVILLE, 2010)

Figura 8 - Modelo espiral de requisitos e projeto



Fonte: Sommerville, 2010, p. 20

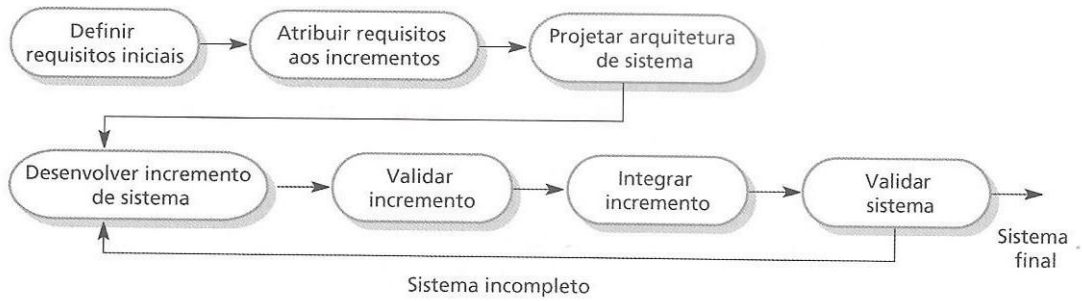
O processo espiral reflete o fato de que os requisitos aparentam as decisões de projetos e ao contrário, portanto, faz sentido interligar esses procedimentos. Dar início pelo centro, cada volta da espiral pode adicionar detalhes aos requisitos e ao projeto. Determinadas voltas podem focar nos requisitos e outras a nova informação adquirida durante os processos de requisito e de projeto faz com que a própria afirmação do problema deva ser alterada. (SOMMERVILLE, 2010)

Com cada interação ao redor do espiral (dá início ao centro e avançando para fora), são levantadas de forma progressiva. Durante o primeiro giro ao redor do espiral, os objetivos, alternativas e restrições são determinados e os riscos são identificados e analisados. E em cada análise examinar se tem erros para verificar os requisitos. (PRESSMAN, 2007)

3.1.4 INCREMENTAL

A entrega incremental (Figura 9) é uma abordagem intermediaria que combina as vantagens desses modelos. No processo de desenvolvimento o cliente identifica, os serviços a serem fornecidos primeiramente pelo sistema. Eles são identificados quais serviços são mais importantes e quais são menos importantes. Assim, são definidas as entregas do incremento para um conjunto de funcionalidades do sistema. As alocações das entregas são feitas pelas suas prioridades com as entregas com maior prioridade são feitas primeiro. (SOMMERVILLE, 2010)

Figura 9 – Entrega incremental



Fonte: Sommerville, 2010, p. 47

Após definidos os incrementos do sistema, são definidos os requisitos detalhadamente para ser desenvolvido. No desenvolvimento, pode começar a pensar nos requisitos para o próximo incremento, mas não podemos fazer alteração no atual. (SOMMERVILLE, 2010)

Após a finalização de um incremento é entregue para o cliente para ser colocado em operação. Isso significa que parte do sistema já está disponível, eles podem experimentar o sistema e isso ajuda a conhecer os requisitos dos incrementos posteriores e das próximas versões e da atual. Quando novos incrementos são concluídos, eles são integrados aos já existentes, que acaba incrementando o sistema e aprimorando com novas funcionalidades. Os serviços comuns podem ser implementados no início do processo ou podem ser implementados de forma incremental, conforme a funcionalidade for exigida por um incremento. (SOMMERVILLE, 2010)

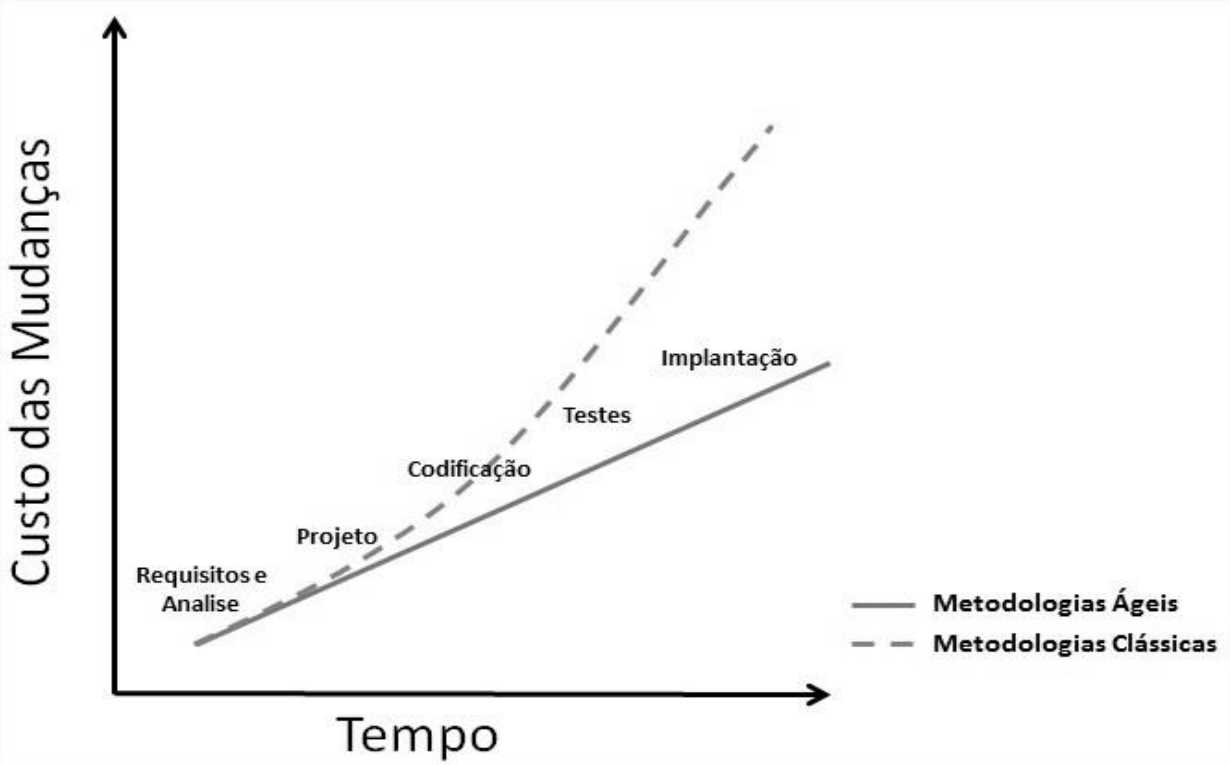
Segundo Sommerville (2010), existem problemas com a entrega incremental, como devem ser relativamente pequenos (não mais que 20 mil linhas de código) e a cada entrega tem alguma funcionalidade de sistema. E difícil mapear os requisitos em entrega de tamanho adequado, a maior parte dos sistemas requer um conjunto de recursos básicos usados por diferentes partes do sistema, pode ser difícil identificar os recursos comuns exigidos por todos os incrementos pois temos que esperar cada implementação do incremento.

4 METODOLOGIA ÁGIL

A metodologia ágil é ajustada para a situação em que a alteração de requisitos é frequente, por isso realmente considerada ágil, a metodologia deve aceitar mudança em vez de tentar antecipar o futuro. (KOSCIANSKI; SOARES, 2007)

Temos uma comparação entre a metodologia clássica e ágil na Figura 10, podemos analisar a relação entre os custos em cada etapa da mudança que está ocorrendo. Na linha pontilhada vamos representar a metodologia clássica, enquanto a linha contínua apresenta como se espera que a relação melhore nas metodologias ágeis.

Figura 10 – Comparação do custo de mudança entre metodologia clássicas e ágeis



Fonte: https://audir.wordpress.com/2012/11/21/reduzindo-custo-de-mudanças-em-projetos-com-desenvolvimento-agil/

Segundo Koscianski e Soares (2007) o termo metodologia ágil nasceu em 2001, quando 17 especialistas que realizava um processo de desenvolvimento de *software*, que representavam os métodos *eXtreme Programming* (XP), *Scrum*, DSDM. Crystal, entre outros, definiram alguns princípios que representavam esse método. Foi criado então a Aliança Ágil. Os principais atributos da metodologia ágil são desenvolvimento interativo e incremental, comunicação e redução de produtos intermediários, como documentação extensiva.

Os fundamentos nos conceitos do Manifesto Ágil são:

- Indivíduos e interações em vez de processos e ferramentas;
- *Software* executável em vez de documentação;
- Colaboração do cliente ao invés de negociação de contratos;
- Respostas rápidas a mudanças em vez de seguir planos.

O Manifesto Ágil não rejeita processos e ferramentas, documentação, negociação de contratos nem planejamento, mas coloca esses requisitos em segundo plano quando fala em indivíduos, o software executável, portanto tem uma colaboração dos clientes, e respostas rápidas às mudanças.

O projeto foca as pessoas e alterações, especialmente quando pensamos em entregas constantes. Desta forma as metodologias ágeis trabalham com equipes altamente motivadas e suporte a modificações durante o processo de desenvolvimento.

4.1 eXtreme Programming (XP)

Segundo Koscianski e Soares (2007), a metodologia eXtreme Programming é indicada para equipes pequenas e médias em que os requisitos sofram modificações rapidamente. As diferenças do método XP em relação às demais são:

- Feedback constantes;
- Abordagem incremental;
- Comunicação entre as pessoas é encorajada.

Em princípio o XP não parece uma metodologia que faz um sentido se aplicada isoladamente. Mas é o conjunto que faz o sucesso de XP, segundo seus criadores, uma verdadeira revolução no desenvolvimento de software.

De acordo com Magela (2006), os valores do XP são:

Comunicação: diálogo constante da equipe para melhor desenvolvimento, comunicação entre par na programação, planejamento, testes e presença do cliente. A falta de diálogo causa mais risco, como a falta de comunicação na mudança de um módulo pode causar prejuízos a várias equipes que dependem do mesmo. Portanto, somente as práticas do XP podem garantir que realmente teremos comunicação entre a equipe.

Simplicidade: Sempre manter as soluções simples, programar somente o essencial, sem componente que não é útil inicialmente.

Feedback: a necessidade de retorno no feedback das tarefas em andamento, sempre deve acompanhar o desenvolvimento, não se deve esperar apenas o final. Realização de testes no término de cada tarefa para correções, caso necessárias, para fazer alterações nos prazos levados em conta, a partir da nova interação. Nosso maior problema é o risco e é alvo principal de nosso feedback,

temos que fazer mudanças para realinhar a direção do software, portanto, diminuamos os riscos.

Coragem: práticas necessárias para desempenhar melhoria no processo de desenvolvimento. Testes, integração contínua, programação em dupla e outras estratégias da metodologia XP.

Práticas do XP segundo Koscianski e Soares (2007) como Magela (2006)

- Planejamento: Definir as necessidades a ser desenvolvidas no projeto, sempre verificar os requisitos atuais para o software, não pensando em requisitos futuros.

- Entregas frequentes: O desenvolvimento simples com os requisitos necessários no momento. Se houver necessidade, atualizar a versão. As versões com o menor tamanho possível, contendo os requisitos de maior valor para o negócio. Na metodologia as entregas são recomendadas a uma frequência mensal, ou no máximo a cada dois meses, para possibilitar feedback rápido do cliente.

- Metáforas: São detalhes do sistema, sem a utilização de termos técnicos com o objetivo de guiar o desenvolvimento do software, visando a maior transparência possível para o cliente.

- Projeto simples: O sistema desenvolvido pela metodologia XP deve ser o mais simples e aceitável e agregar os requisitos atuais. Caso precise de requisitos futuros, esses devem ser adicionados no momento necessário.

- Testes: deve ser priorizada a validação do projeto no desenvolvimento do mesmo. Os desenvolvedores implementam o software criado primeiramente nos testes.

- Programação em pares: A implementação do código é feita em dupla, dois desenvolvedores trabalham em um único computador, Para encontrar

erros sintáticos e semânticos, para aperfeiçoar o código que está sendo implementado. Devem ser modificados sempre que possível.

- Refatoração: Focaliza no desenvolvimento de todo o projeto, está presente, sempre pensando em simplificar uma parte do software, sem perder as funcionalidades necessárias.

- Propriedade coletiva: O código pertence a toda equipe, todos podem fazer alterações, desde que realize os testes necessários, pois todos os integrantes são responsáveis pelo software inteiro. A vantagem caso um colaborador saia da equipe, é que todos tem conhecimento do projeto todo, mesmo que não seja detalhado e não tem dependência de um programador específico.

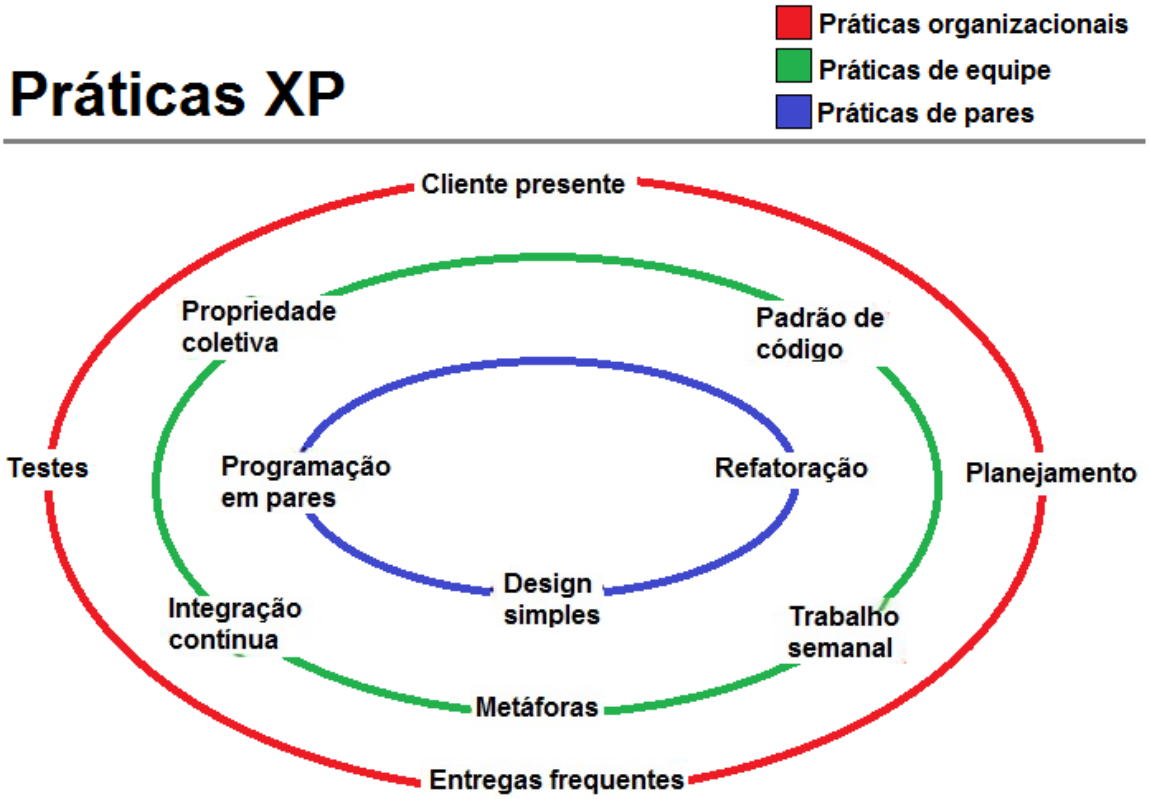
- Integração contínua: Na liberação de novas versões, são realizados testes e validação para integração ao sistema. Por sua vez também deve ser testado, pois só termina quando os erros foram todos corrigidos. Este estágio é promovido com o uso de apenas uma máquina de integração, que deve ter livre acesso a todos os membros da equipe.

- Trabalho semanal de 40 horas: Na metodologia XP não recomendar fazer horas extras, caso tenha necessidade, somente uma semana é recomendada, pois o que ultrapassar esse período, e sinal de que há um problema no projeto, que deve ser resolvido não com o aumento de horas trabalhadas. A prática tem foco nas pessoas e não no processo e planejamento. Caso necessário, os planos devem ser modificados, em vez de sobrecarregar a equipe.

- Cliente presente: É necessário que o cliente sempre esteja presente para sanar dúvidas sobre requisitos, pois evita atraso e erros. O interessante e manter o cliente junto a equipe do projeto.

- Padrão de código: E recomendado a adoção de regras para o desenvolvimento com um padrão a ser seguido pela equipe, pois na XP o código é a documentação.

Figura 11 - Práticas do XP (Imagem e textos adaptados)



Fonte: Adaptado de Schimiguel Apud Devmedia, 2017

Na Figura 11, temos as boas práticas que equipe de desenvolvimento XP utilizam enquanto produzem um software com é grupo que atingir as prática.

4.2 SCRUM

De acordo com Koscianski e Soares (2006, p. 200), “o foco da metodologia é encontrar uma forma de trabalho dos membros da equipe para produzir o software de forma flexível e em um ambiente em constante mudança”. Portanto, essa metodologia foca em uma forma de trabalho flexível e adaptável a ambientes dinâmicos, garantindo tratar mudanças frequentes de requisitos entre outros problemas que poderão surgir no desenvolvimento do sistema.

A metodologia *Scrum* tem princípios semelhantes aos da XP (*eXtreme Programming*): equipes pequenas, com uma troca frequente de requisitos para mudanças e com entregas a um curto prazo para as interações que gerar uma visibilidade para o desenvolvimento. Dentre as diferenças, o *Scrum* já divide o desenvolvimento em ciclos interativos (*sprints*) de até trinta dias, equipes pequenas com até dez integrantes, são formadas por analistas, programadores, engenheiros e gerentes de qualidade, e trabalha em cada funcionalidade definida no início de cada *Sprint*. (MAGELA, 2006)

No *Scrum*, há reuniões diárias de acompanhamento, que são preferencialmente de curto tempo (aproximadamente 15 minutos), com foco no que foi feito desde a última reunião e o que tem que ser feito até a próxima, as dificuldades encontradas e as causas, que são identificadas e resolvidas. Por isso, problemas pontuais no projeto são debatidos e resolvidos diariamente, para que não sejam adiados e apontados tardiamente. (MAGELA, 2006)

Segundo Magela (2006), O ciclo de vida do *Scrum* é baseado em três fases principais, divididas em sub-fases:

- Pré-planejamento: Os requisitos são especificados em um documento chamado *backlog*. Na sequência são organizados por prioridade cada fase é destinada aos seus integrantes para organizar o desenvolvimento. É planejado a equipe de desenvolvimento, as ferramentas a serem usadas, a identificação de possíveis erros no projeto. É verificado a necessidade de Treinamento. Esta fase é concluída com a definição do desenvolvimento, mas

pode ocorrer alterações nos requisitos descritos no *backlog*, como possíveis erros.

- Desenvolvimento: Os riscos já são identificados no começo para ser controlados durante o desenvolvimento, o *software* é desenvolvido em ciclos (*sprints*), e novas funcionalidades são adicionadas. O desenvolvimento é feito de uma forma tradicional, inicialmente se faz análise e, em seguida, o projeto é implementado e testado. Os ciclos são planejados para permanecer de uma semana a um mês.

- Pós-planejamento: Fazem a integração do software, os testes finais e a documentação do usuário. A equipe reúne-se para analisar o avanço do projeto e demonstrar o software atual para os clientes.

4.2.1 Visão geral

Nas equipes *Scrum* temos três papéis principais no projeto: *Prodoct Owner*, o *Scrum Master* e o time de desenvolvimento: (SCHWABER, 2004)

- *Prodoct Owner* (Dono do Produto): representa os interesses do cliente no projeto e, em alguns casos, é o próprio cliente;

- *Scrum Master* (Mestre Scrum): responsável pelo cumprimento de todo o projeto;

- *Team* (Time): responsável por desenvolver o projeto.

4.2.2 Pilares do *Scrum*

O *Scrum* é fundamentado nas teorias sem caráter científico de controle de processo, o que afirmam que o conhecimento vem da experiência e da tomada de decisão do que é conhecido. Assim temos três pilares: a transparência, a inspeção e a adaptação. (SCHWABER; SUTHERLAND, 2013)

4.2.2.1 Transparência

É ter uma visão clara e objetiva de todo o processo. O importante nesta parte é a comunicação, solicitando aspectos definidos por um padrão comum para que todos tenham um mesmo entendimento do que está sendo apresentado. Por exemplo, a definição de “pronto” para todos os responsáveis pela criação e validação do produto. (SCHWABER; SUTHERLAND, 2013)

4.2.2.2 Inspeção

A inspeção determina que, frequentemente, deve-se inspecionar os artefatos gerados, bem como o progresso do projeto em direção ao objetivo, para detectar indesejáveis variações, porém, sem ter uma frequência que atrapalhe a própria execução das tarefas. As inspeções são mais benéficas quando realizadas de forma diligente por inspetores especializados no trabalho a se examinar. (SCHWABER; SUTHERLAND, 2013)

4.2.2.3 Adaptação

Quando necessita de uma adaptação determinada por uma inspeção que os aspectos de um processo estão fora do padrão aceitável, gerando um produto onde o resultado não será o esperado, o processo deve ser ajustado o mais breve possível para minimizar mais riscos e melhores resultados. (SCHWABER; SUTHERLAND, 2013)

A inspeção e adaptação podem ser feitas de quatro formas:

- Reunião de planejamento da *Sprint*
- Reunião diária (*Daily Scrum Meeting*)
- Reunião de revisão da *Sprint*
- Retrospectiva da *Sprint*

4.2.3 Eventos do *Scrum*

A definição de eventos são usados no *Scrum* para criar uma rotina, com o intuito de diminuir reuniões desnecessárias, e para que todo evento tenha uma duração máxima. Para garantir que não vai ocorrer perda no processo de planejamento, permite-se transparência e uma inspeção criteriosa. A não inclusão de qualquer um dos eventos resultará na redução da transparência e da perda de oportunidade para inspecionar e adaptar. (SCHWABER; SUTHERLAND, 2013)

4.2.3.1 Sprint

O *Sprint* é o principal evento do *Scrum*, é o período onde incrementos "prontos" são criados, e normalmente possuem a duração de um mês. Uma nova *Sprint* se inicia após a conclusão da *Sprint* anterior. (SCHWABER; SUTHERLAND, 2013)

Cada *Sprint* tem a definição do que deve ser construído. Durante o escopo é detalhado e revisado entre a equipe e o *product owner*, podendo ser renegociado, desde que não comprometa os objetivos iniciais descritos. Não é aconselhado alterar a composição do *time* ou a sua duração.

4.2.3.2 Reunião de Planejamento da Sprint

É realizada a reunião para planejar a *Sprint*, e este plano e criado com toda a equipe *Scrum*. Essa reunião possui um *time-box* com no máximo 8 horas para cada Sprint com um mês de duração, E para *Sprint* menores pode alterar esse tempo. O *Scrum Master* organiza e garante que todos saibam e entenda seu propósito e orienta o *Time Scrum* a se manter no limite do time-box. (SCHWABER; SUTHERLAND, 2013)

A reunião de planejamento da *Sprint* responde as seguintes questões:

- O que pode ser entregue como resultado do incremento da próxima *Sprint*?

- Como o trabalho necessário para entregar o incremento será realizado?

4.2.3.3 Reunião Diária

A reunião diária é de 15 minutos, para o *Time* sincronizar as atividades e criar plano para o dia. O foco da reunião e verificar o trabalho desenvolvido desde o último encontro, organizar o trabalho para antes da próxima reunião diária. (SCHWABER; SUTHERLAND, 2013)

É sempre mantido o mesmo horário e local para o encontro, a fim de reduzir a complexidade. E verificada as seguintes perguntas:

- O que eu fiz ontem que ajudou o *Time* de Desenvolvimento a atender a meta da *Sprint*?

- O que eu farei hoje para ajudar o *Time* de Desenvolvimento atender a meta da *Sprint*?

- Eu vejo algum obstáculo que impeça a mim ou o *Time* de Desenvolvimento no atendimento da meta da *Sprint*?

4.2.3.4 Revisão da *Sprint*

A revisão e feita no final da *Sprint* com o intuito de ver o incremento pronto e adaptar o *Backlog* do Produto caso seja necessário. O *Time* e a parte interessada informa sobre o desenvolvimento da *Sprint* e mostra o projeto para saber se e necessário alteração e se está de acordo com o esperado. O foco da reunião e mostrar o incremento e receber feedback. (SCHWABER; SUTHERLAND, 2013)

Com uma duração de 4 horas para uma *Sprint* de um mês, e para eventos menores é habitualmente mais breve. O *Scrum Master* garante a participação de todos os colaboradores, para que entendam o seu objetivo e limita para executar da melhor forma possível.

4.2.3.5 Retrospectiva da *Sprint*

A retrospectiva é para analisar o desenvolvimento da *Sprint* já finalizada, os seus pontos positivos e negativos, com uma duração de 3 horas para eventos de um mês e caso menor é usualmente mais breve. O *Scrum Master* garante a participação do *Time Scrum*, o entendimento de seu propósito, mantendo o tempo e participação como um membro auxiliar do time. (SCHWABER; SUTHERLAND, 2013)

4.2.4 Artefatos do Scrum

Os artefatos são definidos para maximizar as especificações para o projeto com a transparência da informação, chave para o *Scrum* para todo entendimento dos artefatos. (SCHWABER; SUTHERLAND, 2013)

4.2.4.1 *Backlog* do Produto

O *Backlog Product* é uma lista priorizada, contendo breves descrições de todas as funcionalidades desejadas para o produto. É necessário para o projeto iniciar com um esforço para qualquer mudança a ser feita em outras versões do produto, coletando e documentando todos os requisitos de uma vez. Normalmente, a equipe e o *Product Owner* (dono do produto) escrevem e priorizam os itens iniciais do *Backlog Product*, sendo esses itens suficientes para que a equipe inicie a primeira interação. O *Backlog Product* irá crescer e mudar à medida em que se aprende mais sobre o produto e sobre o cliente. (SCHWABER; SUTHERLAND, 2013)

4.2.4.2 *Backlog* da *Sprint*

O *Backlog* da *Sprint* são itens do *Backlog Product* selecionados para a Sprint, com o plano de entregar o incremento do produto e atingir o objetivo da Sprint. São feitos previsões para o Time de Desenvolvimento sobre a funcionalidade para esta no próximo incremento, e sobre o trabalho necessário para entregar essa funcionalidade em um incremento “Pronto”. (SCHWABER; SUTHERLAND, 2013)

O *Backlog* da *Sprint* torna visível todo o desenvolvimento do Time para atingir o objetivo da Sprint. O *Scrum Master* mantém o *Backlog Sprint* atualizado para refletir que tarefas são completadas e quanto tempo a equipe acredita que será necessário para completar aquelas que ainda não estão prontas.

4.2.4.3 Incremento

O incremento e a soma de todos os itens do *Backlog* do Produto completados durante a *Sprint*. No final de uma *Sprint* um incremento está “Pronto”, atendendo a definição de utilizável, independente do *Product Owner* decidir por liberá-lo realmente ou não. (SCHWABER; SUTHERLAND, 2013)

4.2.5 Transparência do Artefato

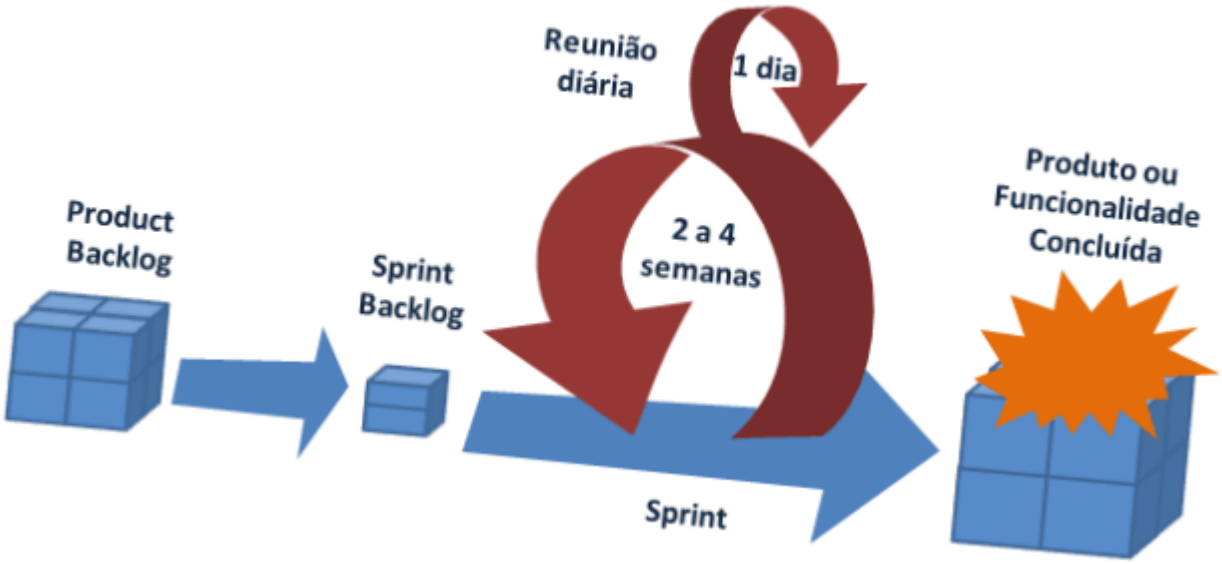
O *Srum Master* trabalha com o *Product* Owner, Time de Desenvolvimento, e outras partes envolvidas, para saber se os artefatos estão reaumente transparentes para o melhor entendimento do projeto, caso contrário pode se aumentar os riscos e o valor pode diminuir. (SCHWABER; SUTHERLAND, 2013)

4.2.5.1 Definição de “Pronto”

Quando o item do *Backlog Product* ou um incremento é descrito como “Pronto”, todos os integrantes tem que entender o significado e caso o trabalho esteja completo para assegurar a transparência. E a “Definição de Pronto” para o Time *Scrum* quando o trabalho está completo no incremento do produto para ser potencialmente utilizado, que aderem à definição atual de “Pronto” para toda a equipe. (SCHWABER; SUTHERLAND, 2013)

A Figura 12 apresenta uma ilustração do ciclo do *Scrum*:

Figura 12 – ciclo *Scrum*



Fonte: http://www.mindmaster.com.br/scrum/

5 ESTUDO DE CASO

O Sistema de Eventos da FATEC Americana é uma ferramenta de auxílio no controle de proposição, aprovação e inscrição para eventos – palestras, cursos de extensão e afins - realizados na FATEC Americana.

Os usuários desse sistema podem ser divididos em quatro grupos: professores, alunos, membros da comissão de extensão e coordenadores de curso. Os professores apresentam a proposta do evento. Os membros da comissão de extensão decidem, no primeiro nível da alçada de aprovação, se o evento se encaixa como extensão ou não. Os coordenadores constituem o segundo nível da alçada de aprovação, analisando o conteúdo do evento proposto e se este está alinhado com os cursos oferecidos pela Faculdade. Por fim, os alunos se inscrevem nos eventos.

A ferramenta permitirá um acompanhamento das propostas cadastradas, o status da aprovação, os recursos exigidos pelo evento e as inscrições realizadas, além de gerar a documentação necessária e garantir a formalidade do processo.

5.1 EQUIPE

A equipe é formada por quatro integrantes, sendo elas: Ana Lucia, Camila, Joline e Thaynara. Que iram dividir entre si as tarefas, relacionadas a documentação e programação do software, pré-definidas e descritas no cronograma do projeto.

5.2 SISTEMA

O sistema foi desenvolvido para o cumprimento da matéria de Laboratório de Engenharia de Software. O mesmo é um emissor de certificados, que atenderá a uma necessidade identificada na Fatec Americana.

5.3 LEVANTAMENTO DE REQUISITOS

A primeira etapa é o levantamento de requisitos, a reunião de planejamento, o que normalmente é feito através do *Prodoct Owner*, o *Scrum Master*, e o *time Scrum*, à equipe elaborou um escopo dos requisitos funcionais e não funcionais, incluindo diagramas, tabelas e interfaces do sistema.

Com o escopo realizado pela equipe, junto com os detalhes solicitados o *time* começa a criação e modelagem do projeto.

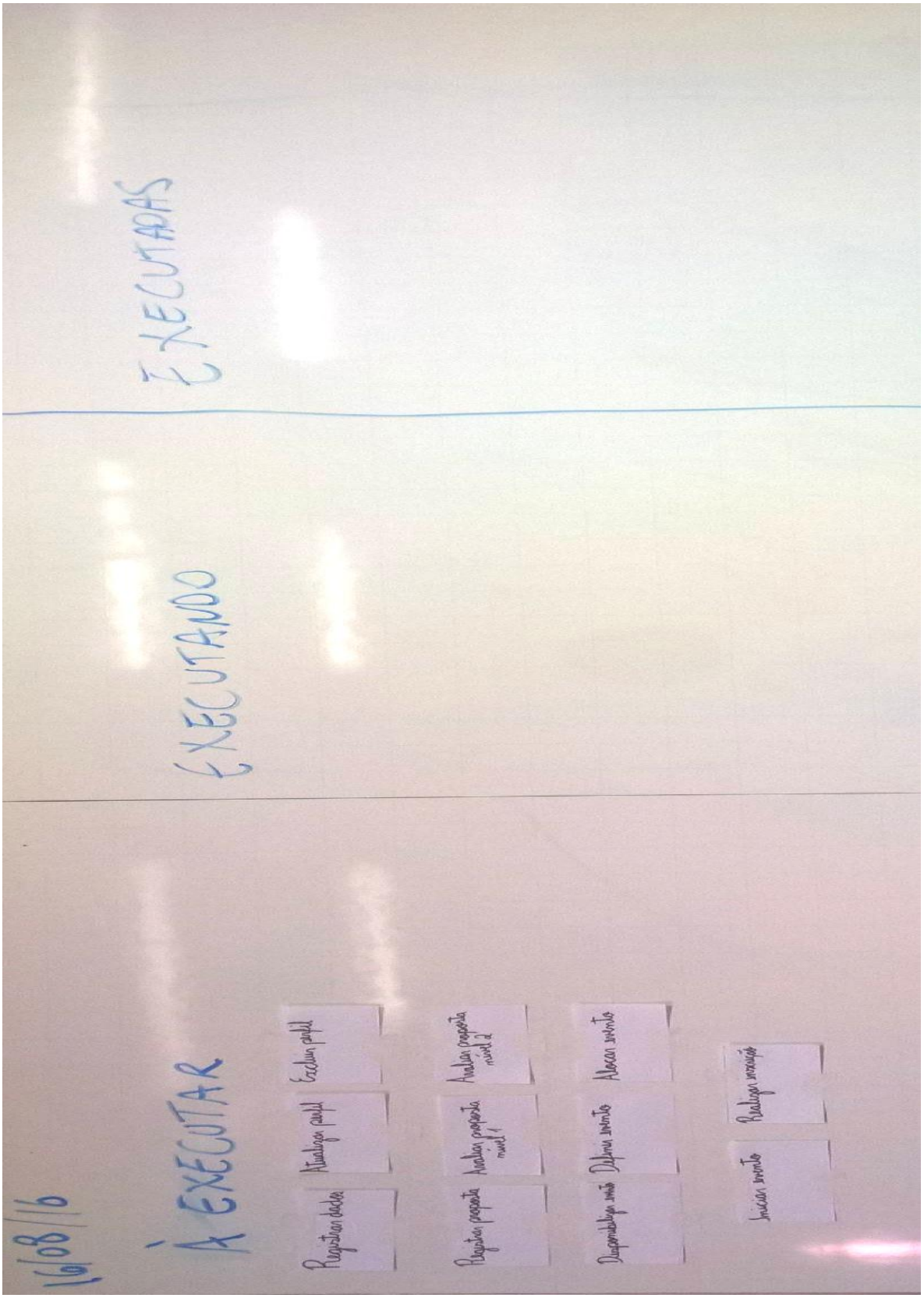
5.4 BACKLOG

Todos os requisitos necessários para o desenvolvimento do projeto devem ser listados e descritos no *Product Backlog*. Geralmente são separados em atividades, de acordo com a ordem de sua prioridade. A classificação de prioridade é exibida e exposta no topo, ou colocado na ordem. (SCHWABER; SUTHERLAND, 2013)

O quadro deve ser fixado para todos os envolvidos no projeto. Todos ficam cientes dos requisitos e das tarefas mais importantes, bem como as que estão em desenvolvimento e as que ainda não iniciaram. A responsabilidade de criar e colocar o quadro no lugar correto é do *Scrum Master.*

A Figura 13 mostra as atividades que foram fixadas no quadro que será exposta para a equipe. Com as tarefas para o desenvolvimento, temos três tabelas, onde a primeira lista as tarefas à executar, na segunda as tarefas em execução e na última as tarefas executadas.

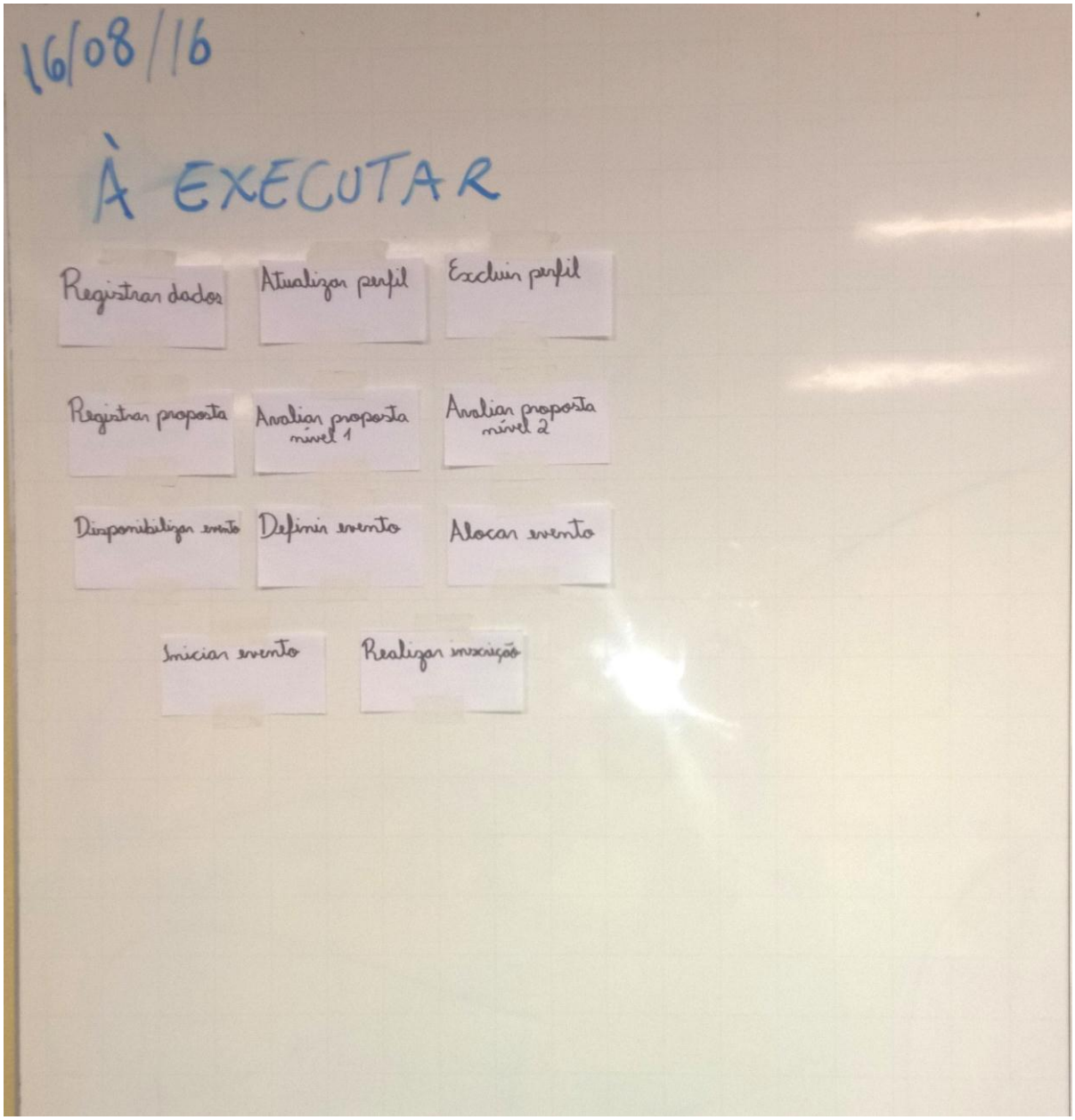
Figura 13 – Sprint 1



Fonte: Autor

A Figura 14 mostra todas as atividades para executar em todo projeto como Registrar dados, Atualizar e Excluir perfil, Registrar proposta, Avaliar proposta nível 1 e 2, Disponibilizar evento, Definir evento, Alocar evento, Iniciar evento e Realizar inscrição, organizado na ordem das prioridades.

Figura 14 – Sprint 1 à executar

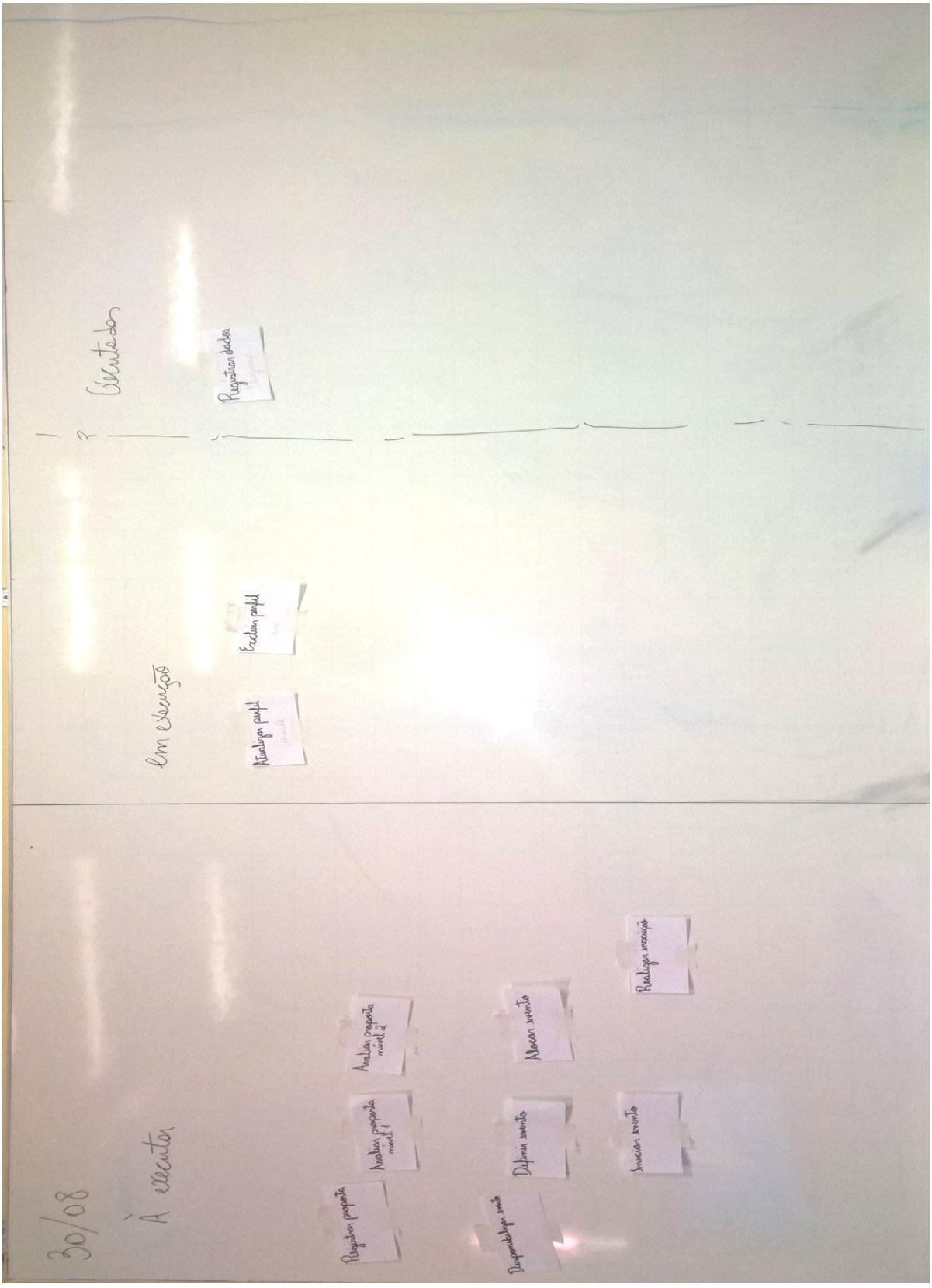


Fonte: Autor

5.5 Sprint 2

Na Figura 15 pode ser claramente visualizado no quadro as tarefas para executar, em execução com prioridade para o *Sprint,* e a já executada. Todos os dias temos a *Scrum Meeting* que são as reuniões diárias, porém antes da equipe iniciar o desenvolvimento. Essa reunião é para acompanhar o andamento do desenvolvimento com o *time*. (SCHWABER; SUTHERLAND, 2013)

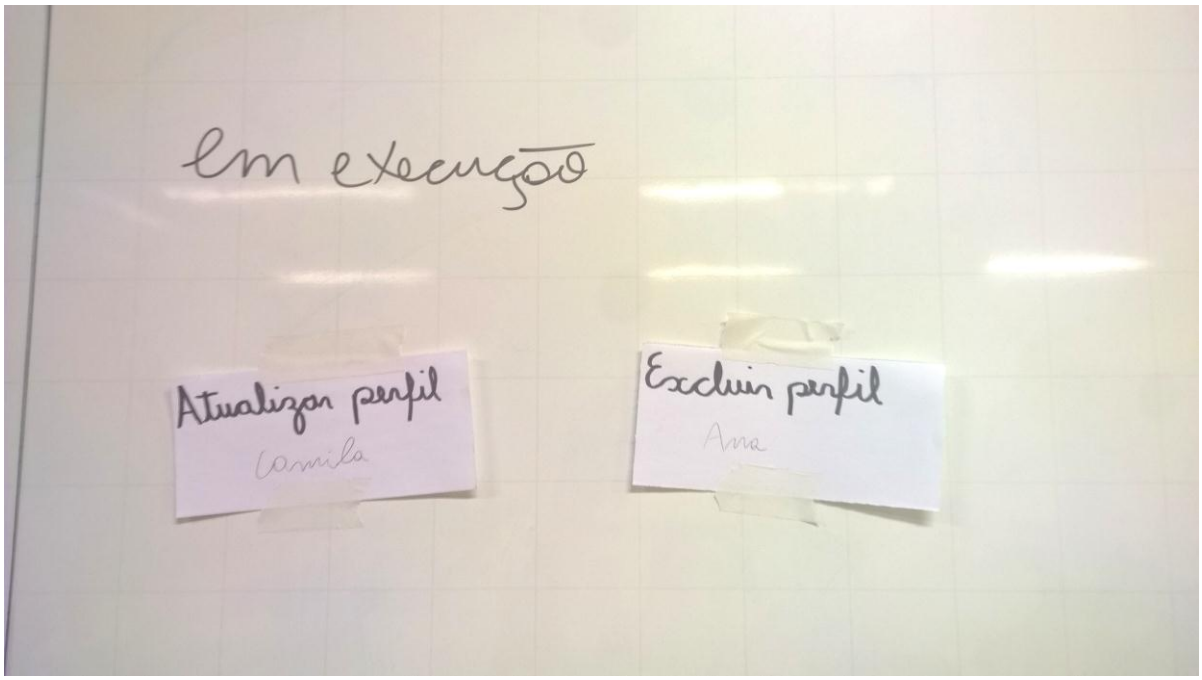
Figura 15 – Sprint 2



Fonte: Autor

Na Figura 16 podemos visualizar duas tarefas em execução para atualizar e excluir perfil em desenvolvimento pelos membros da equipe.

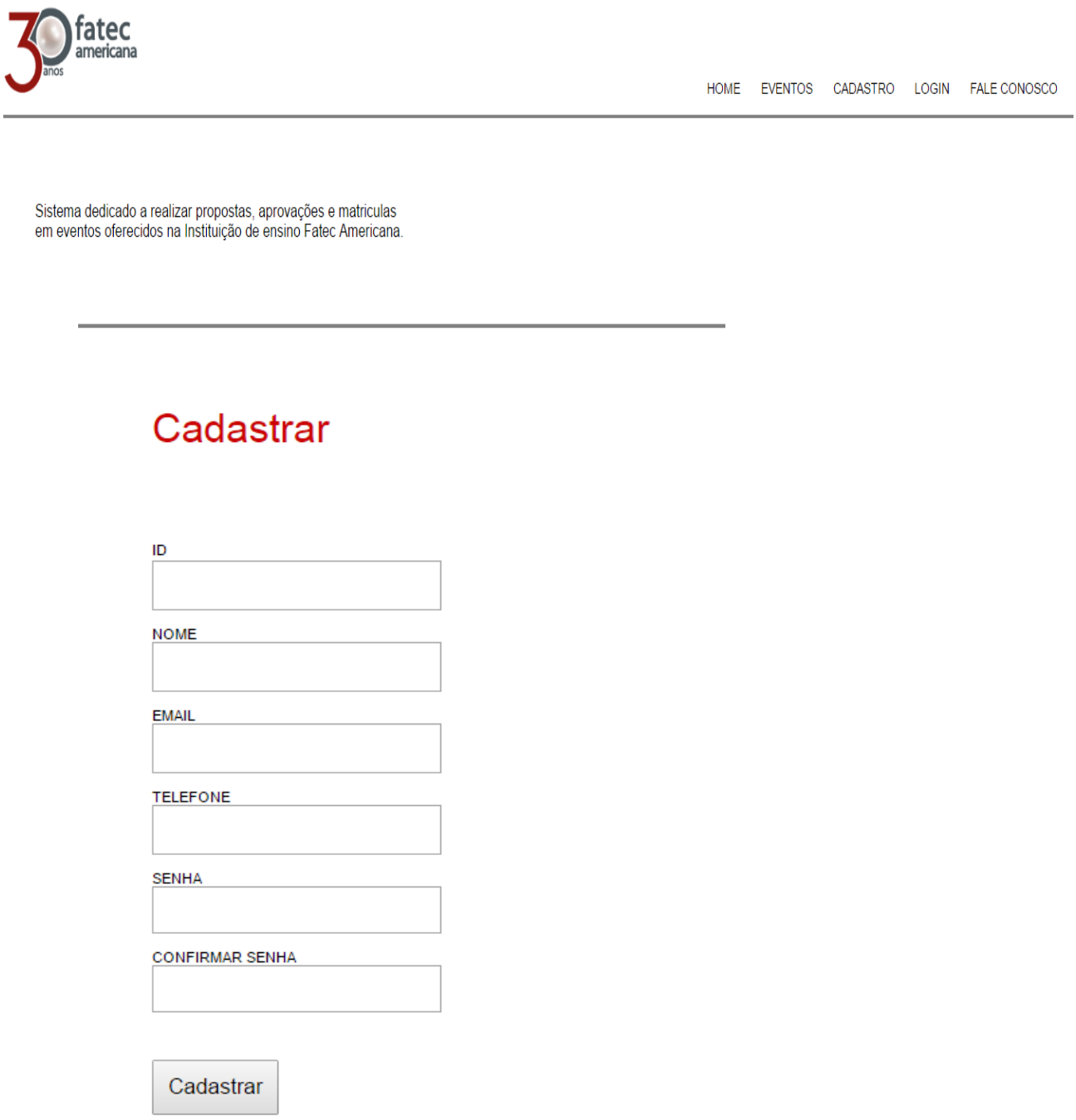
Figura 16 – Sprint 2 à executar



Fonte: Autor

Na Figura 17 podemos visualizar a funcionalidade do Registro de dados para cadastro do usuário, e se deparar com uma interface simples e objetiva, onde alunos, coordenadores, professores e funcionários poderão se cadastrar, preenchendo seus dados pessoais e criando uma senha de acesso, para poderem visualizar, aprovar e matricular-se em cursos e palestras oferecidos na faculdade.

Figura 17 – Tela Cadastro de usuário



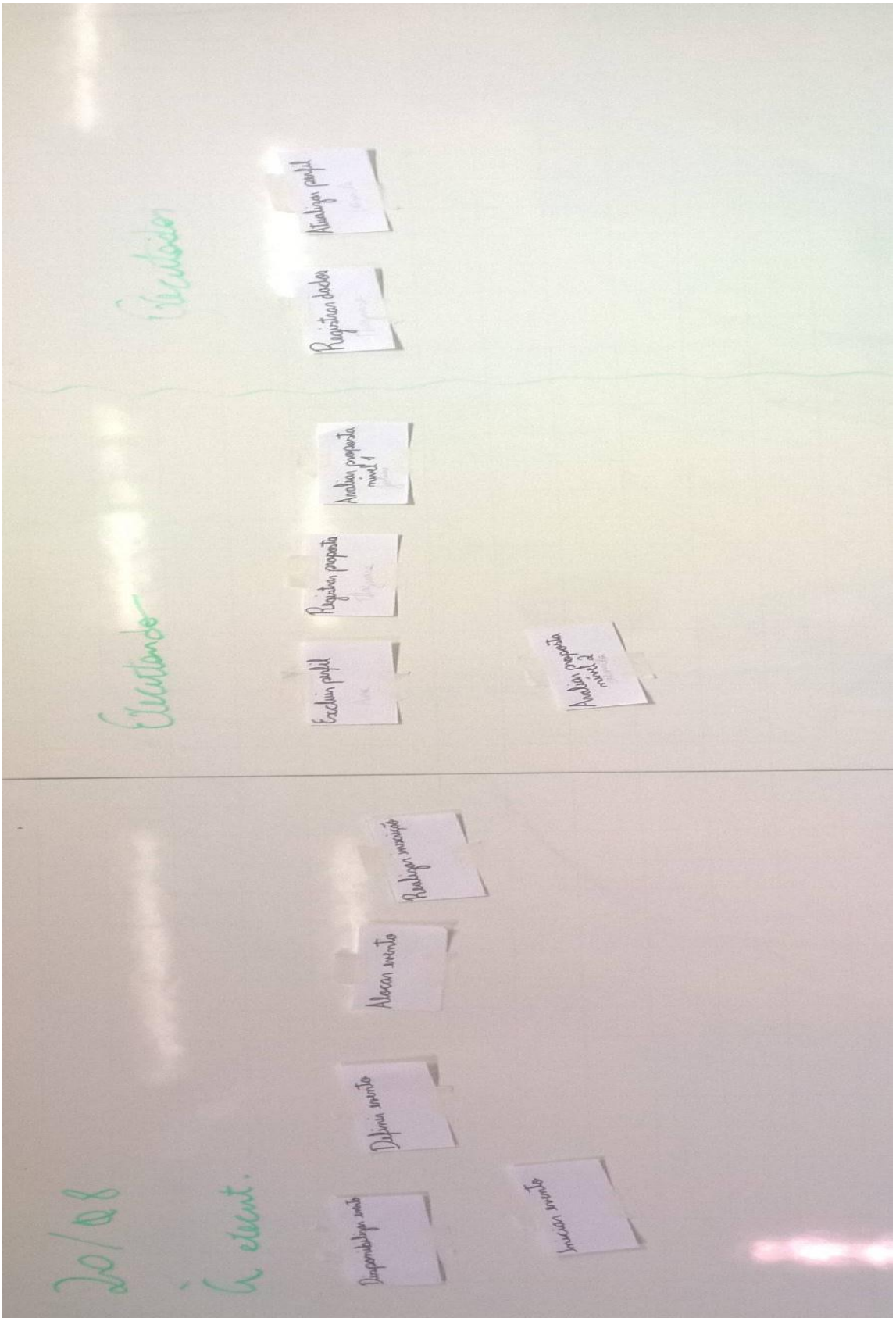
Fonte: Autor

No final da *Sprint* foi feito *Sprint Review* que é uma reunião informal com a finalidade de apresentar a funcionalidade desenvolvida e promover a colaboração do *time* para o que será feito em seguida. (SCHWABER; SUTHERLAND, 2013)

5.6 Sprint 3

Na Figura 18 que se encontra abaixo temos um erro na data que é 20/09 mostramos mais um quadro com suas *Sprint* em execução com o *time* e parte do projeto já finalizado. Temos a *Sprint Retrospective* que é retrospectiva do *Sprint* com a Equipe de Desenvolvimento, o *Product Owner* e o *Scrum Master*. Nessa reunião todos devem expor suas ideias para melhorar o próximo.

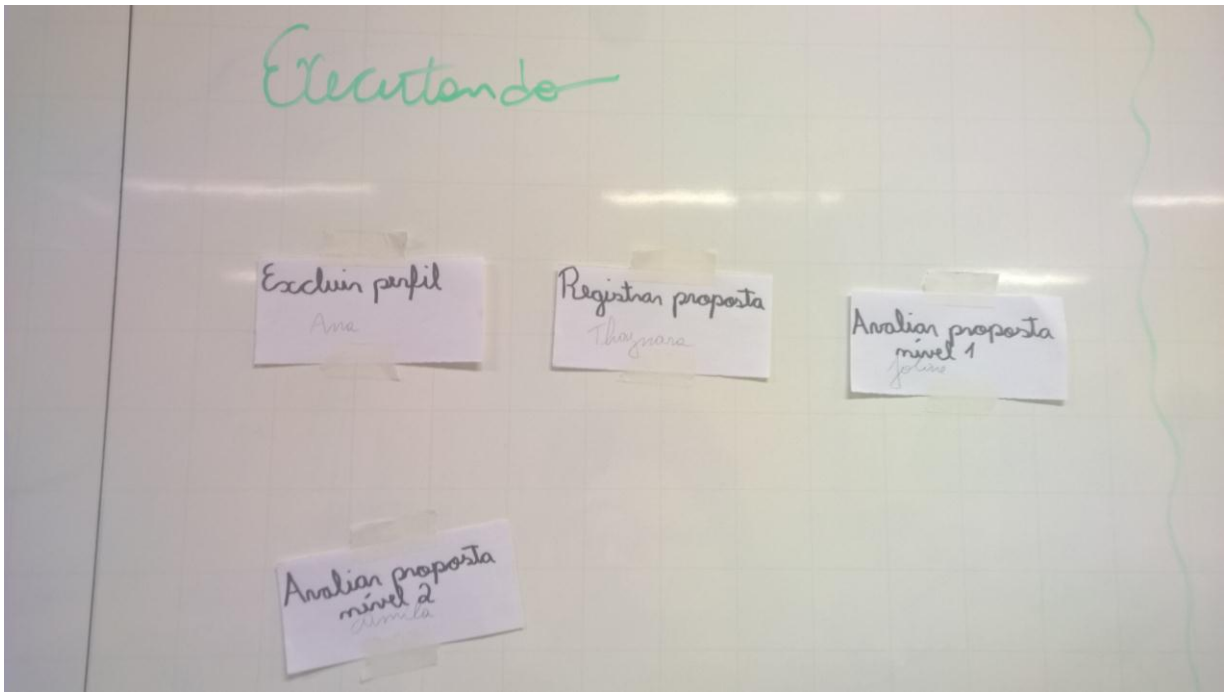
Figura 18 – Sprint 3



Fonte: Autor

Na Figura 19 temos em execução as tarefas de Excluir perfil, Registrar proposta, Avaliar proposta nível 1 e 2 com a equipe.

Figura 19 – Sprint 3 à executar

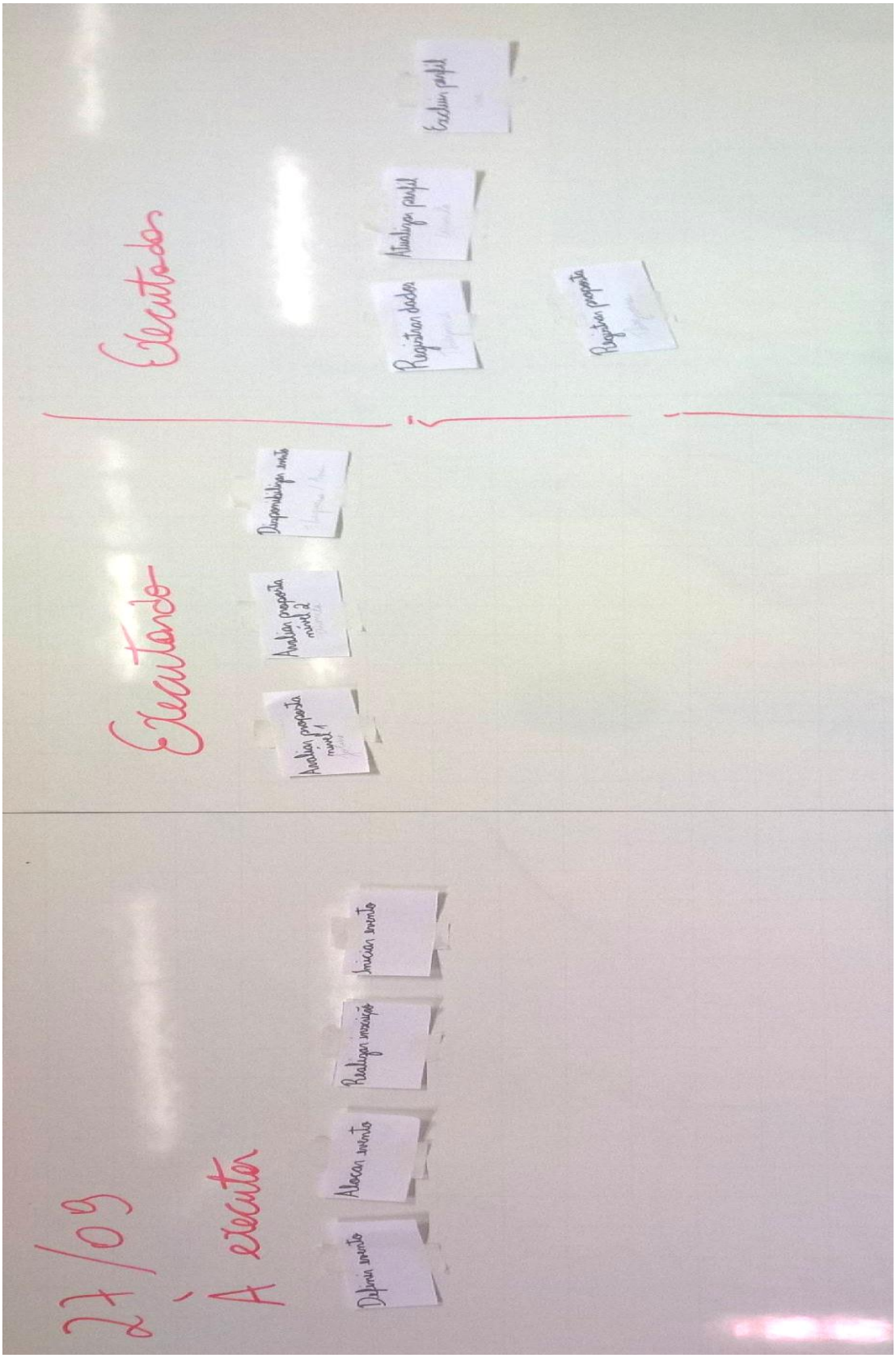


Fonte: Autor

5.7 Sprint 4

A Figura 20 mostra o andamento das tarefas no dia 27/09. Antes de começar temos a *Sprint Planning Meeting* (Reunião de Planejamento da Sprint) que é a reunião da equipe, Srum Master que é o cliente que acontece antes de iniciar um *Sprint,* precisamos receber a confirmação. Caso exista alteração é nessa etapa que o cliente deve descrevê-las. (SCHWABER; SUTHERLAND, 2013)

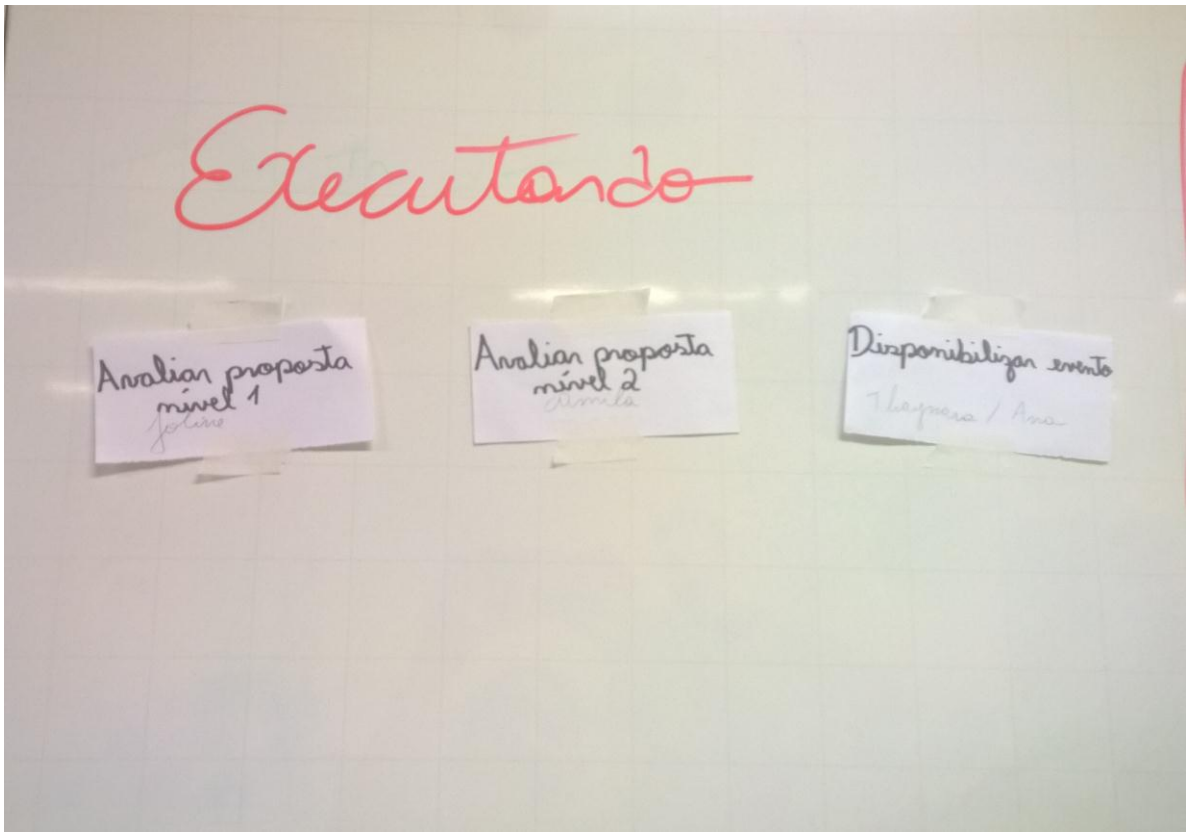
Figura 20 – Sprint 4



Fonte: Autor

Na Figura 21 pode ser visualizado as tarefas em execução, como Avaliar proposta nível 1 e 2 e, Disponibilizar evento para o desenvolvimento do projeto.

Figura 21 – Sprint 4 à executar



Fonte: Autor

Na Figura 22 podemos ver a funcionalidade Registrar proposta, que está finalizada com mais uma parte já aprovada e revisada pelo cliente.

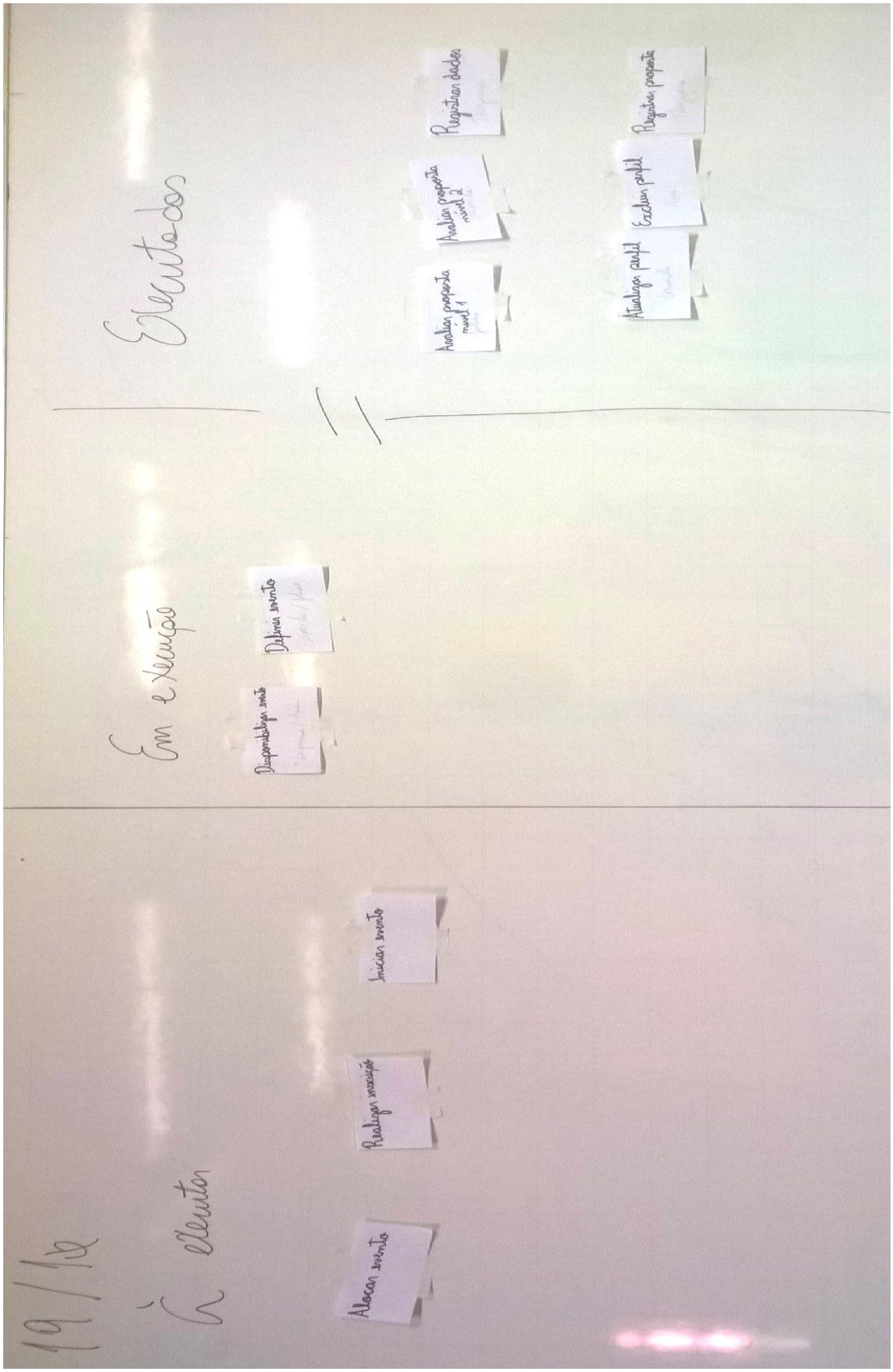
Figura 22 – Tela de Cadastrar Proposta

Fonte: Autor

5.8 Sprint 5

Na Figura 23 temos mais uma Sprint em execução, com tarefas para executar, em execução e executadas.

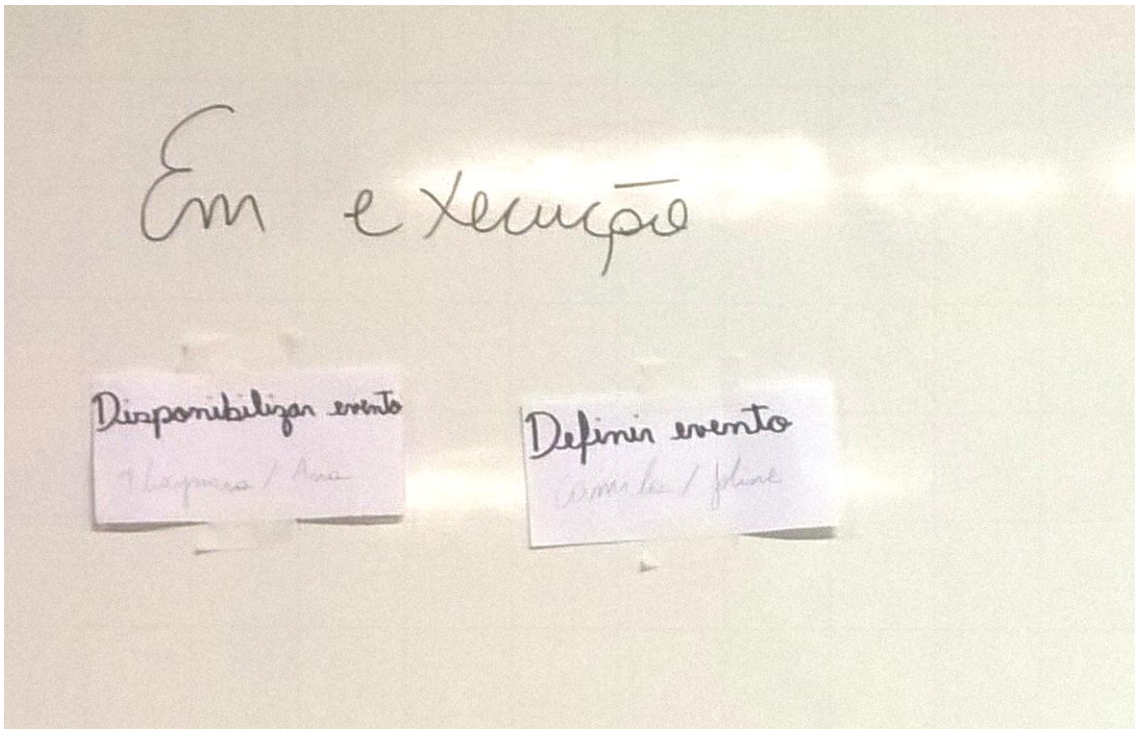
Figura 23 – Sprint 5



Fonte: Autor

Na Figura 24 temos as tarefas em execução, que tem nesse *Sprint* como Disponibilizar e Definir evento que a equipe está executando.

Figura 24 – Sprint 5 à executar

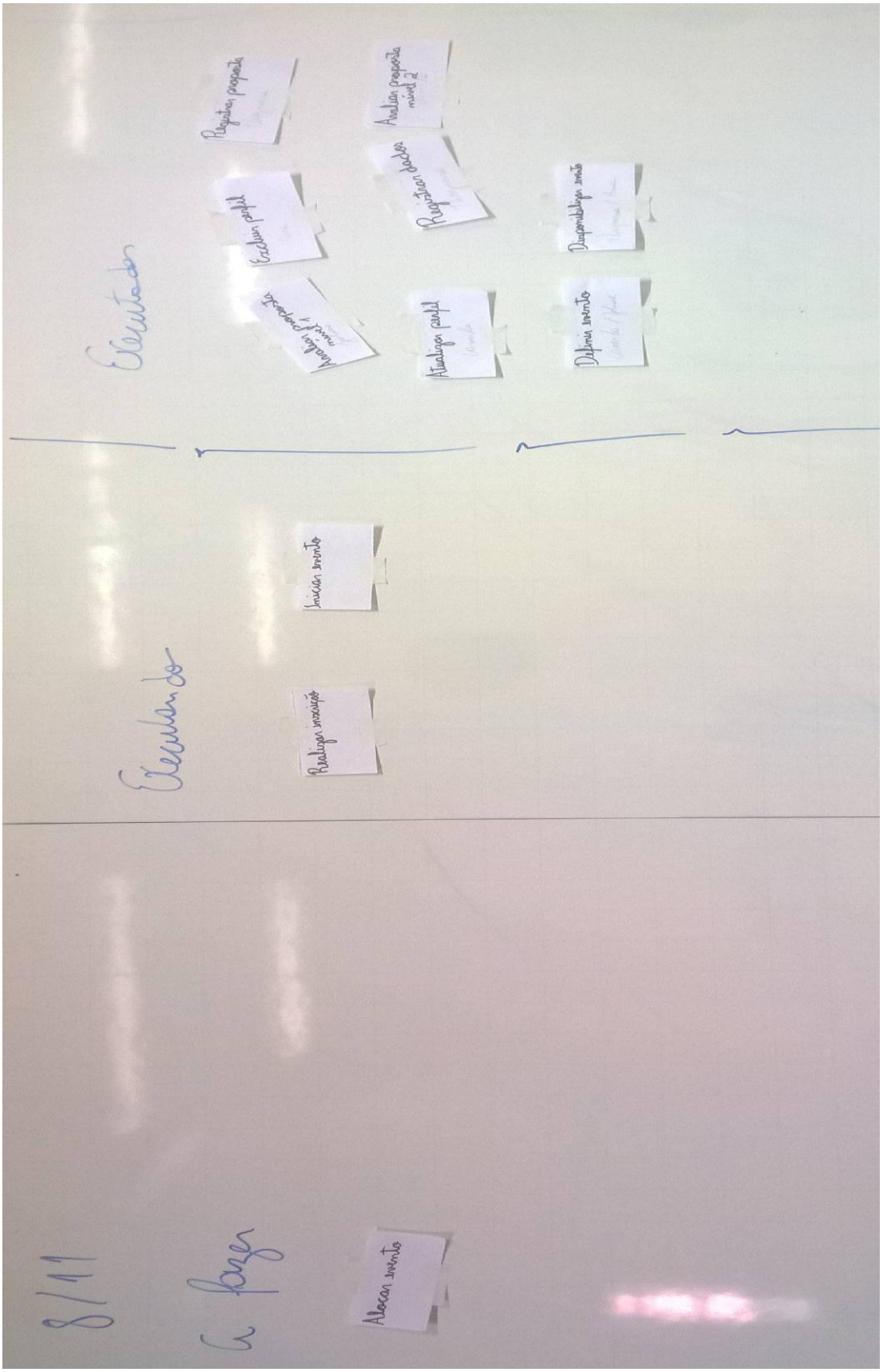


Fonte: Autor

5.9 Sprint 6

Na Figura 25 temos uma nova Sprint começando com as tarefas a fazer, executar e executadas no quadro.

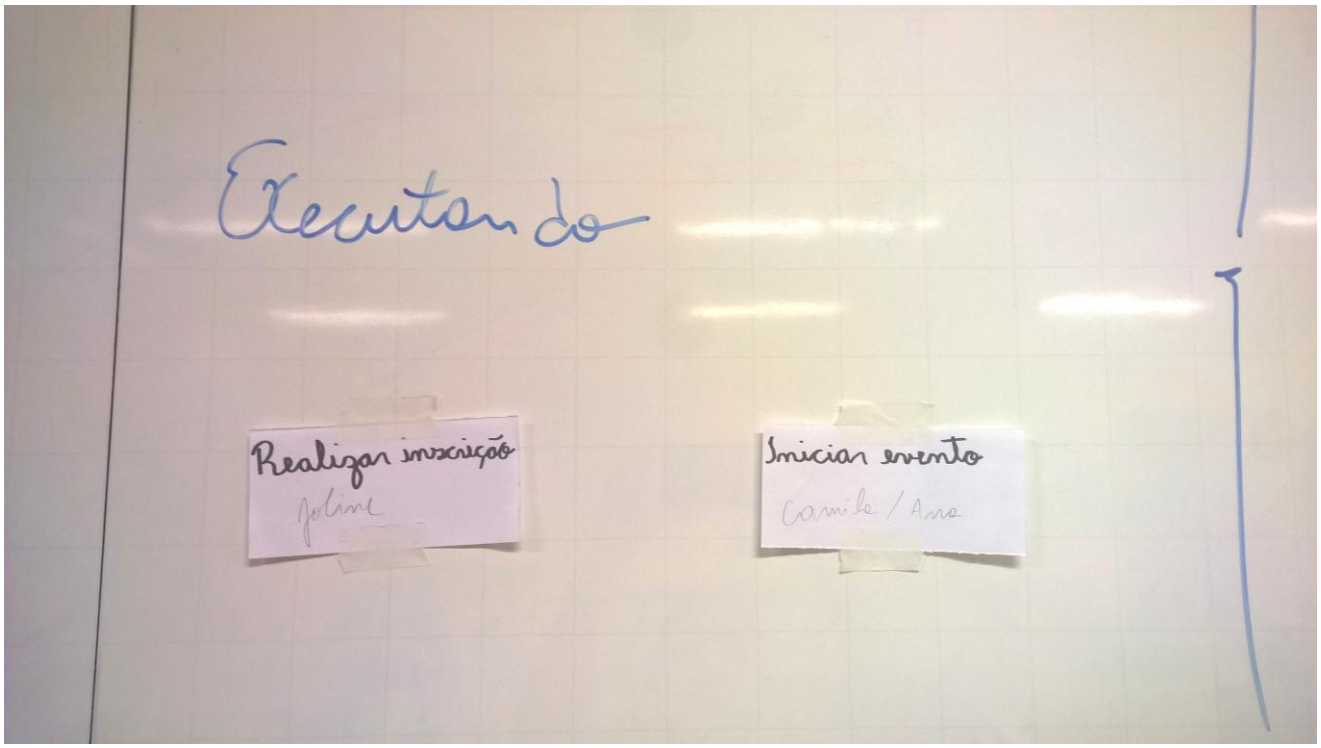
Figura 25 – Sprint 6



Fonte: Autor

Na Figura 26 se visualiza as tarefas em execução no ciclo, temos de Realizar inscrição e Iniciar evento.

Figura 26 – Sprint 6 à executar

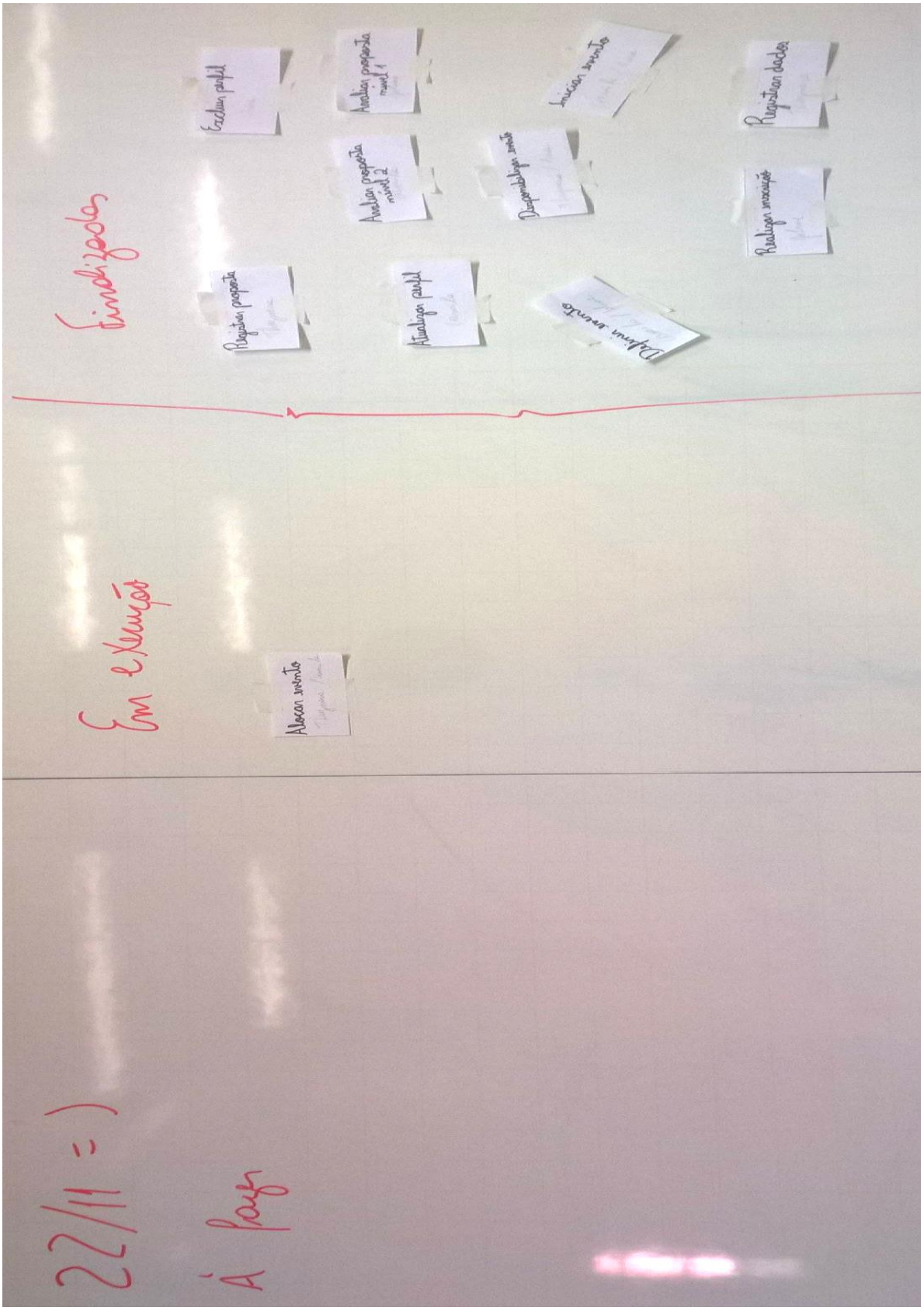


Fonte: Autor

5.10 Sprint 7

A Figura 27 no quadro mostra as tarefas em execução e finalizadas. Já não se encontra tarefas à fazer.

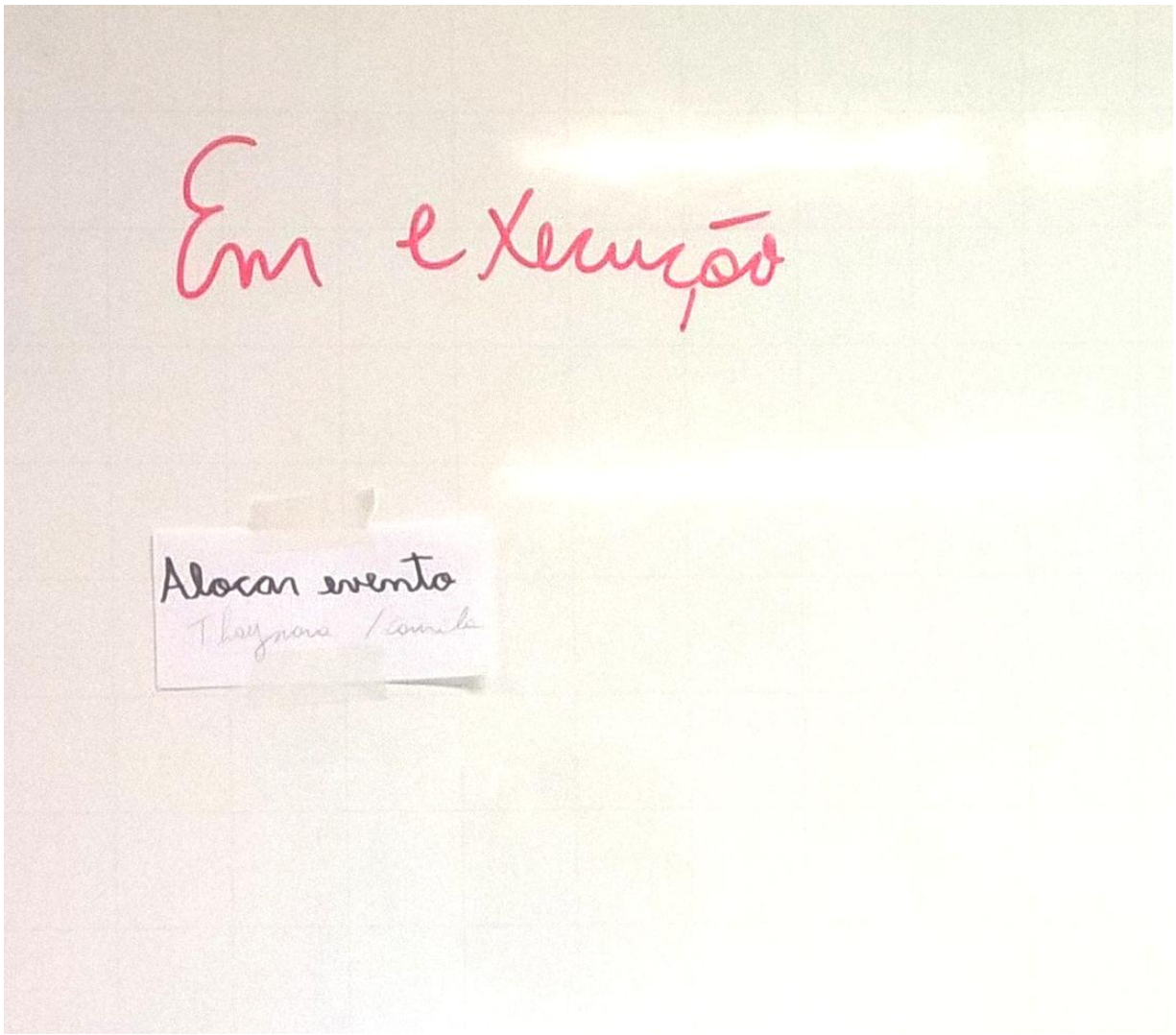
Figura 27 – Sprint 7



Fonte: Autor

A Figura 28 mostra só mais uma tarefa em execução, a Alocar evento para finalizar o desenvolvimento.

Figura 28 – Sprint 7 à executar

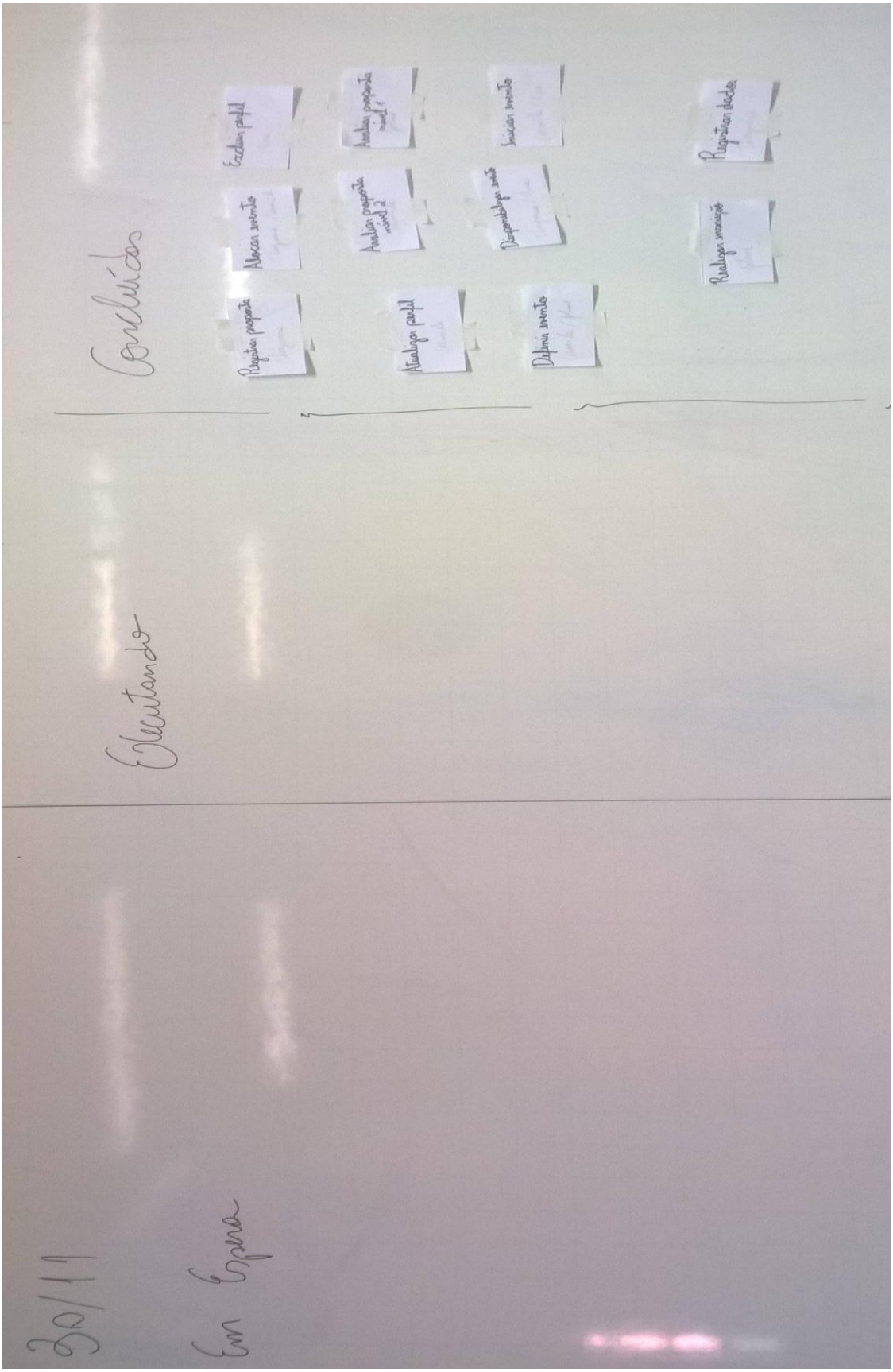


Fonte: Autor

5.11 Sprint 8

Na Figura 29 temos o último quadro com todas as tarefas já finalizadas só fazendo revisão final.

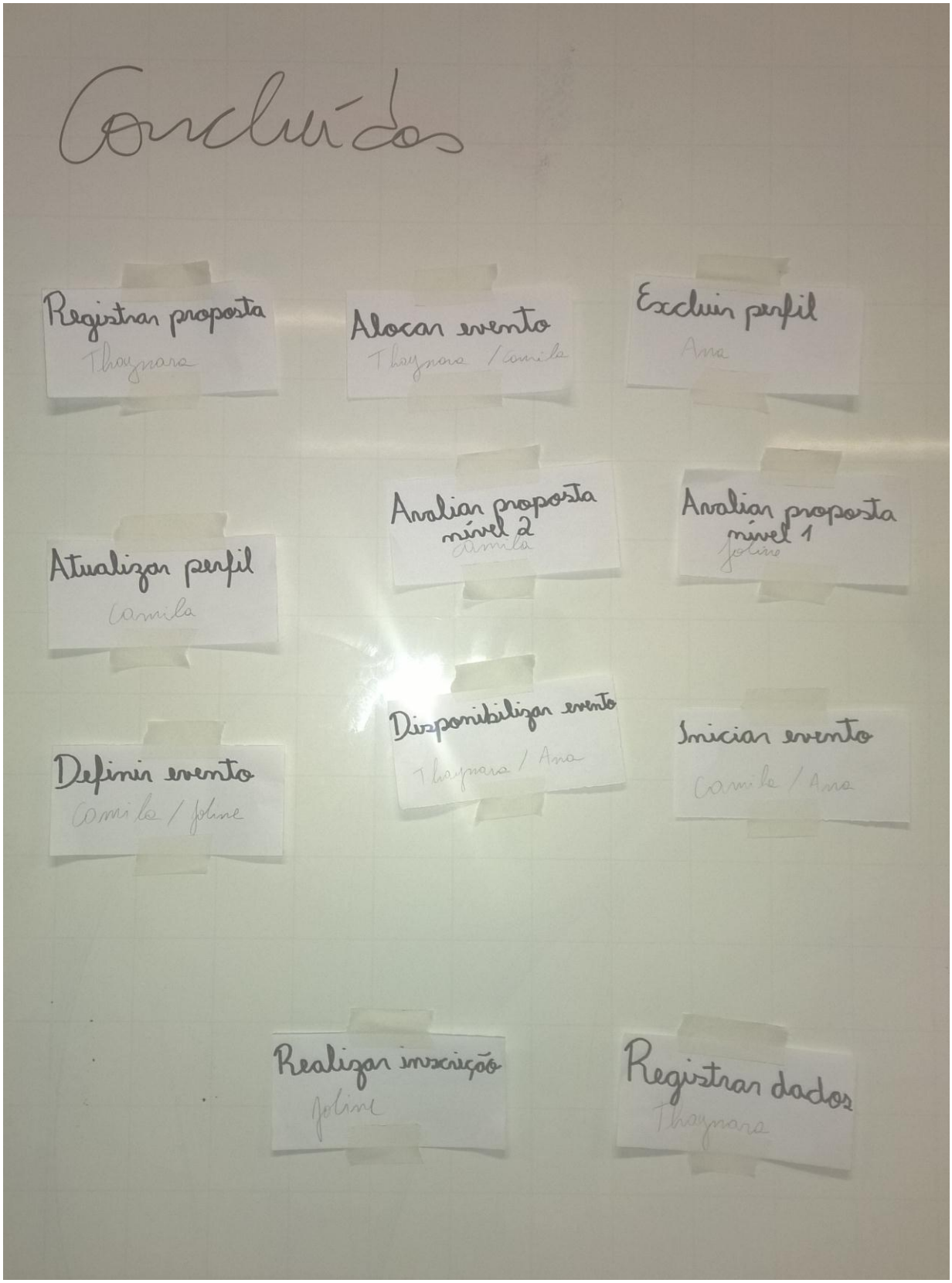
Figura 29 – Sprint 8



Fonte: Autor

Na Figura 30 abaixo podemos visualizar todas as tarefas finalizadas e revisadas pelo *time*.

Figura 30 – Sprint 8 concluída



Fonte: Autor

5.12 PROBLEMAS IDENTIFICADOS

Durante todo o processo de desenvolvimento do software descrito no estudo de caso acima, foram encontrados diversos problemas. Alguns desses problemas foram facilmente resolvidos, outros mesmo com alguma dificuldade também foram solucionados, porém alguns problemas identificados no decorrer dos processos, obtiveram uma dificuldade maior para o *time*.

Alguns do problemas iniciais foi a utilização da metodologia, para uma adaptação ao desenvolvimento, pela falta de conhecimento e pratica, outro seria definição de requisitos pôr no começo parecia um sistema simples mais quanto chegamos para todas as definição para desenvolvimento com uma complexidade maior do que aparentava.

6 RESULTADOS

A aplicação dos conceitos do *Scrum*, para mostrar em quais pontos são importantes no desenvolvimento do *software*, mais especificamente no estudo de caso acima, mostrando todas as etapas e uma boa relação da equipe.

Todos os problemas foram minimizados e procurado maior conhecimento nas áreas que a equipe não tinha o mesmo, focamos em aplicar os conceitos da metodologia *Scrum*. Metodologia ágeis vieram para suprir a dificuldade de tornar o desenvolvimento mais prático, com maior controle de riscos e maior adaptação diante das mudanças necessárias no decorrer do desenvolvimento, com foco em evitar os transtornos enfrentados pela equipe e pelo cliente, fazendo com que o software atendesse melhor as necessidades da faculdade, e o mesmo poder ser implantado para funcionamento. (MARCHI; PAINKA, 2013)

Posteriormente aplicou-se, um questionário com o intuito de verificar a opinião da equipe sobre toda a utilização no processo de desenvolvimento.

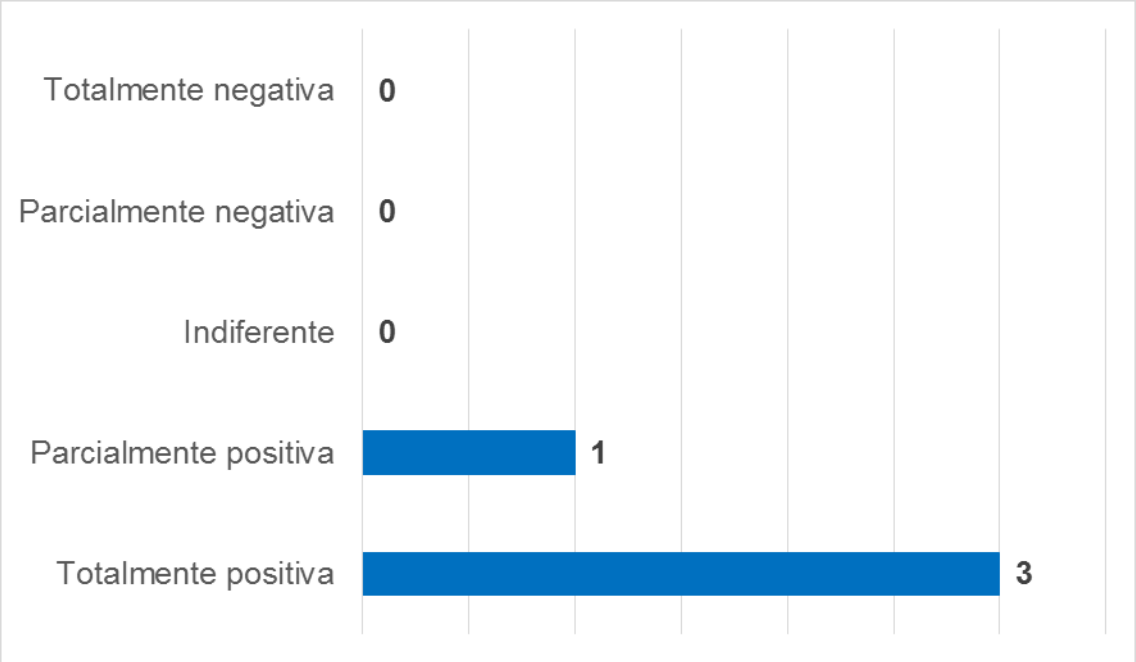
Para a avaliação do *Scrum* não é necessário encontrar resultados e respostas perfeitas, mas sim encontrar resultados e respostas que nos ajudem a responder algumas questões, como por exemplo: “A adoção do *Scrum* valeu a pena? O que devemos melhorar? Fizemos um produto melhor? Nosso produto tem menos defeitos? Somos mais rápidos com a adoção do *Scrum*?”

Com o intuito de conhecer melhor a opinião dos envolvidos no processo de adoção do *Scrum* e responder as perguntas citadas no parágrafo anterior, foi realizado um questionário com os quatro participantes da equipe. Este foi dividido em três partes: a primeira parte traz perguntas em um âmbito mais geral da utilização no processo de desenvolvimento; já a segunda parte traz perguntas referentes ao *Scrum*, e pôr fim a terceira parte com perguntas a respeito da visão do processo de desenvolvimento com a adoção do *Scrum*.

Desde a primeira reunião da equipe para escolher a adoção de uma metodologia de desenvolvimento, foi enfatizada a importância da presença e da

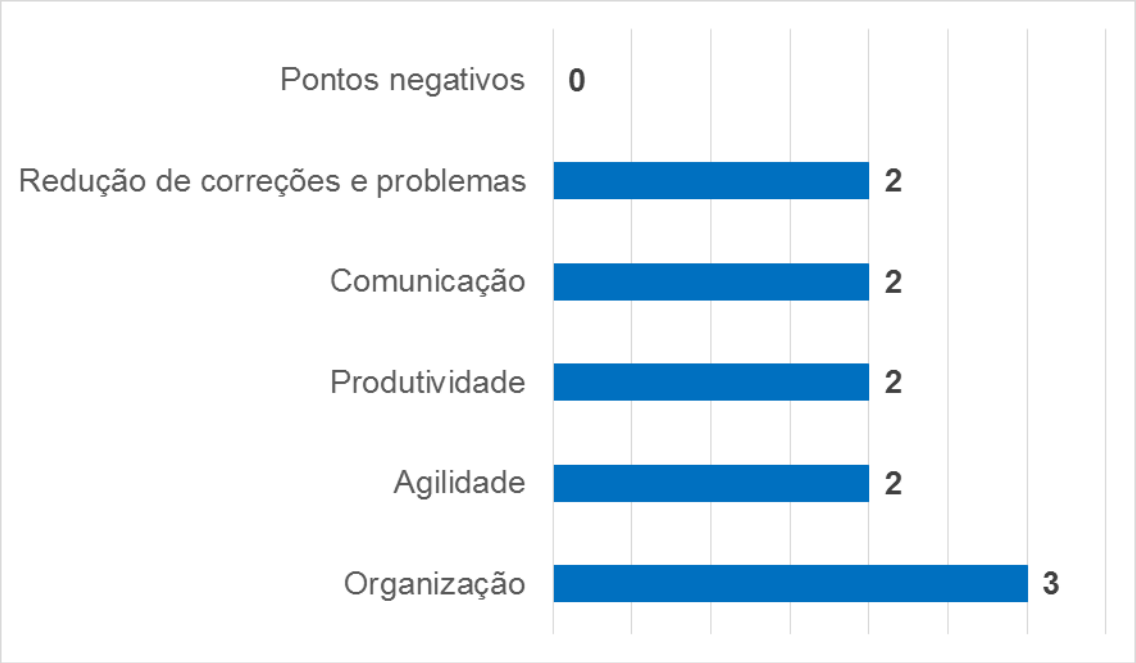
opinião de todos, com o objetivo de obter maior aceitação e interesse coletivo. Assim como também as decisões foram tomadas em conjunto, respeitando a área de conhecimento de cada um, mas nivelando todos em um mesmo grau de importância para o sucesso da equipe.

GRÁFICO 1 – Você considera que a utilização do processo de desenvolvimento no geral foi?



Fonte: Autor

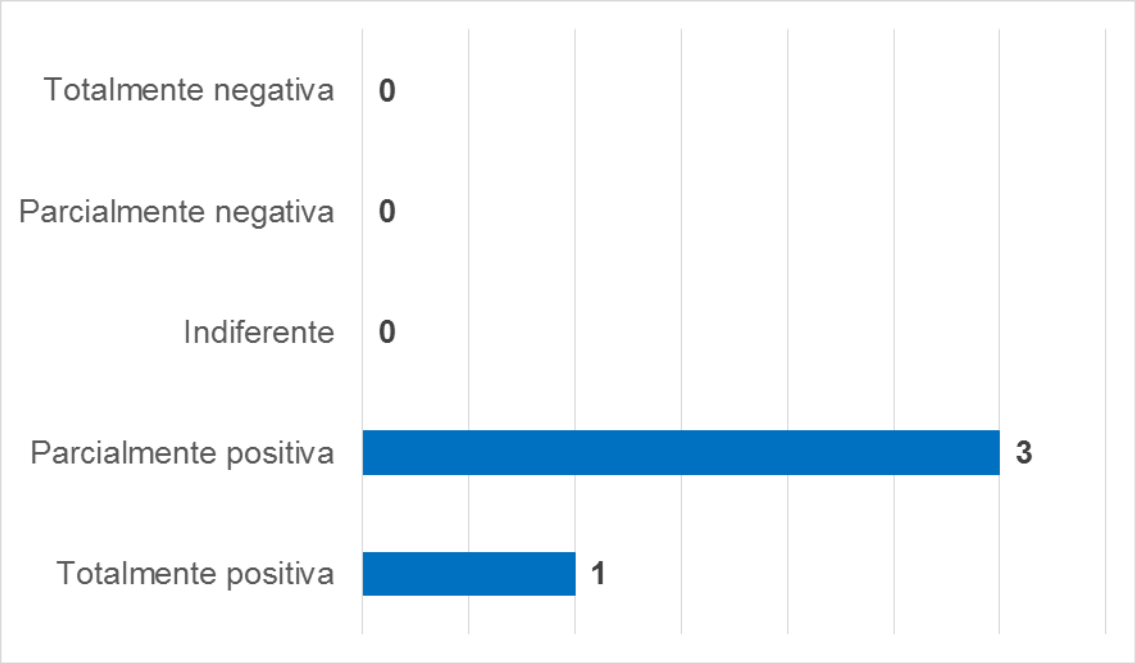
GRÁFICO 2 - Quais seriam os principais pontos a se destacar, conforme seu posicionamento na questão anterior?



Fonte: Autor

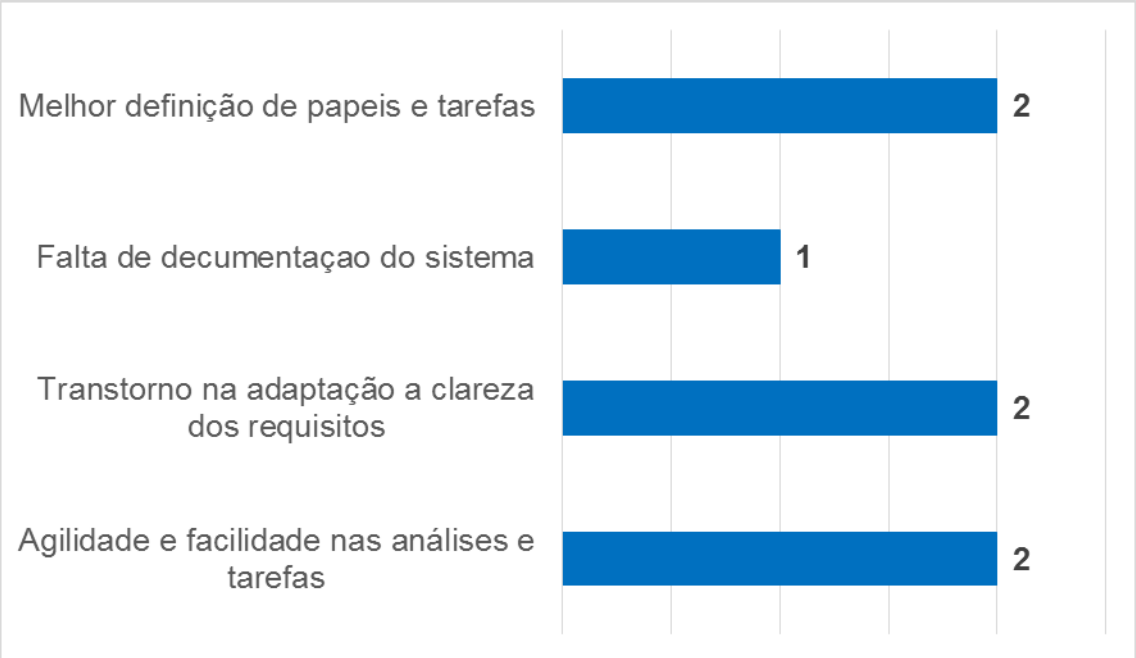
Os gráficos 1 e 2 mostram os resultados obtidos nas perguntas de conhecimento geral, em que a maioria dos integrantes definiram como a utilização são extremamente positivas, destacando-se alguns pontos positivos, como a organização, agilidade, produtividade e comunicação, diminuindo um dos maiores problemas encontrado no desenvolvimento como o retrabalho.

GRÁFICO 3 - Como você avalia a utilização do processo no contexto do gerenciamento de tarefas (análise das novas requisições, divisão em tarefas, clareza nos requisitos, documentação)?



Fonte: Autor

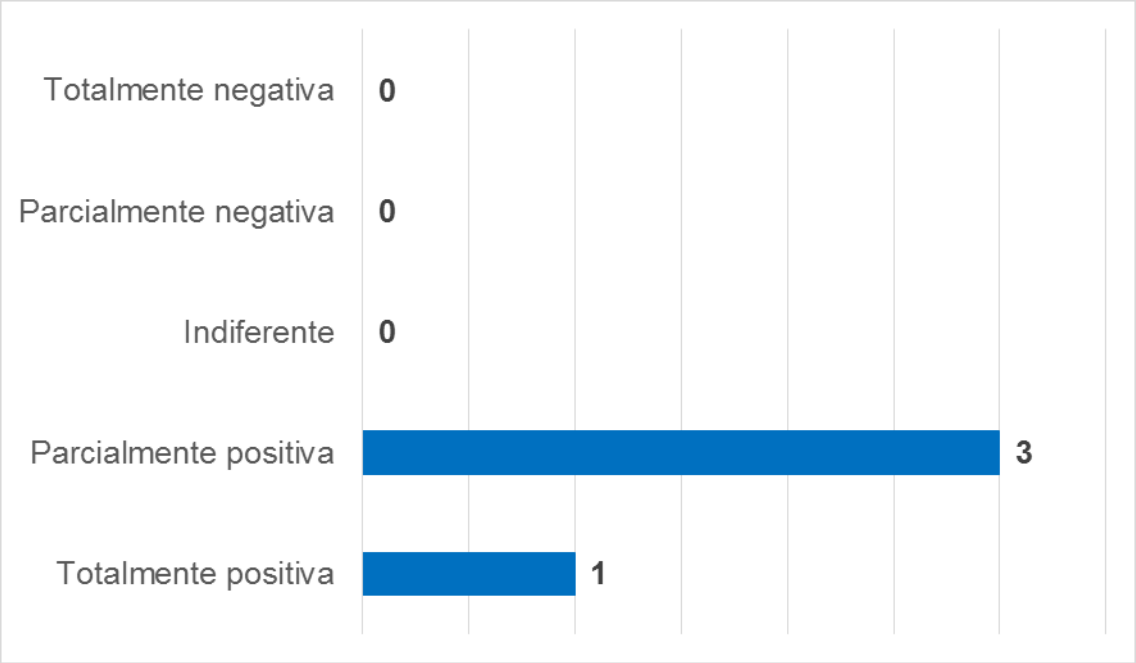
GRÁFICO 4 - Quais seriam os principais pontos a se destacar, conforme seu posicionamento na questão anterior?



Fonte: Autor

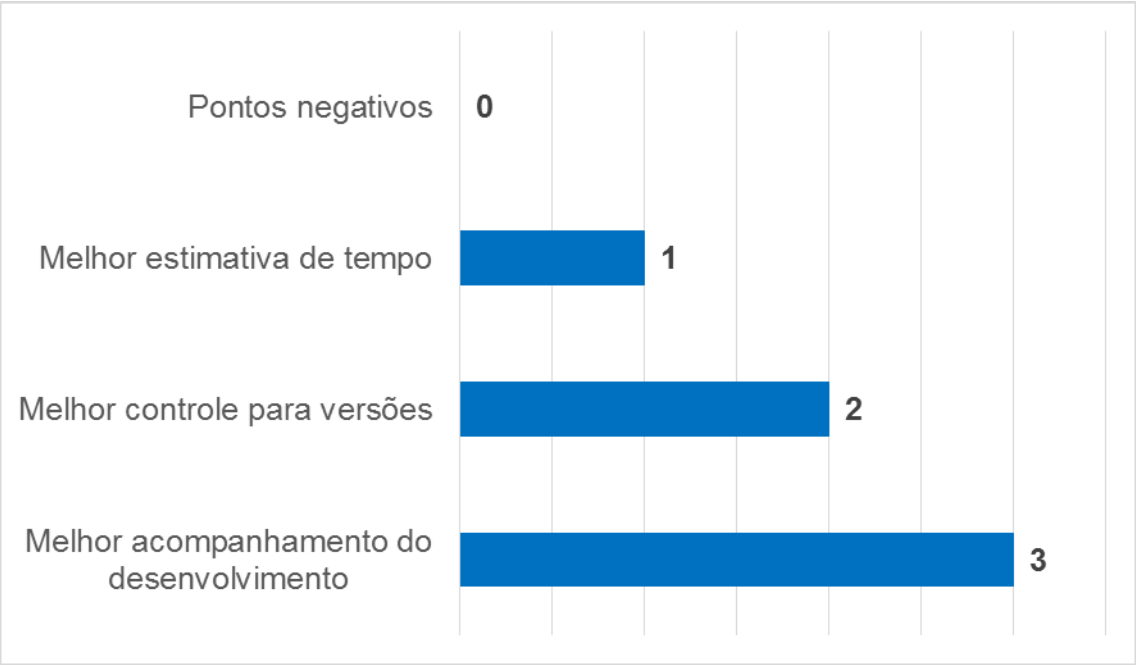
Analisando os gráficos 3 e 4 percebeu-se que problemas foram encontrados na documentação do produto e na clareza dos requisitos. Em contrapartida, a definição dos papéis e tarefas a serem realizadas foram muito bem empregadas. O ganho com a organização do *backlog* do produto facilitou muito a análise das solicitações prioritárias. Neste quesito destaca-se a falta de uma documentação mais sólida, sendo que a equipe ainda não a possui e permanece uma oportunidade de melhoria.

GRÁFICO 5 - Como você avalia a utilização do processo no contexto do gerenciamento de tempo (delegação de tarefas, estimativas de tempo, acompanhamento do desenvolvimento)?



Fonte: Autor

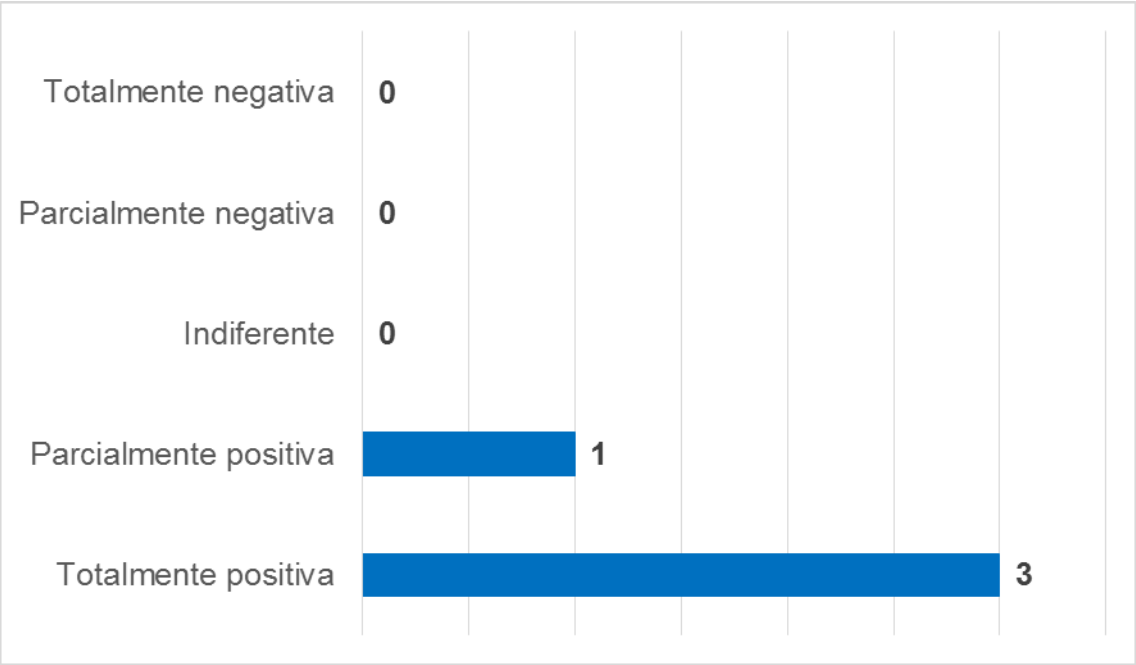
GRÁFICO 6 - Quais seriam os principais pontos a se destacar, conforme seu posicionamento na questão anterior?



Fonte: Autor

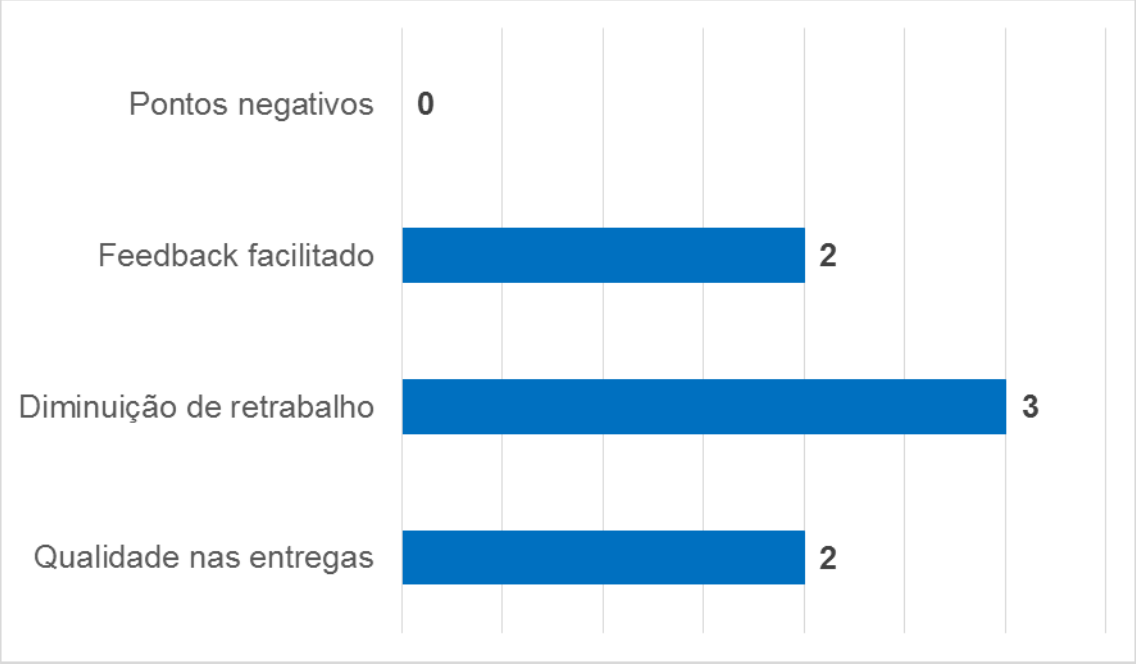
Os gráficos 5 e 6 mostram o reconhecimento parcialmente positivo na delegação de tarefas, estimativa de tempo e acompanhamento do desenvolvimento. A transparência das tarefas que estão sendo realizadas foi um ponto de destaque, permitindo um melhor acompanhamento, como também uma definição das versões desenvolvidas, apresentando o que será entregue e quando.

GRÁFICO 7 - Como você avalia a utilizar do processo no contexto do gerenciamento de qualidade (validação das implementações, gerenciamento das versões)?



Fonte: Autor

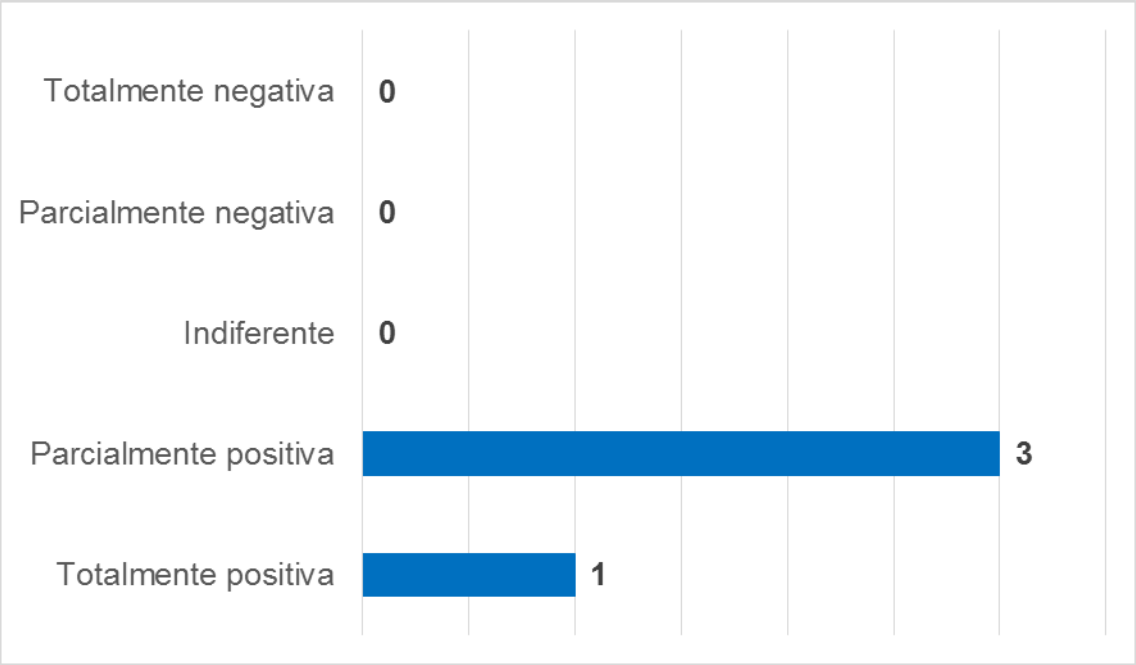
GRÁFICO 8 - Quais seriam os principais pontos a se destacar, conforme seu posicionamento na questão anterior?



Fonte: Autor

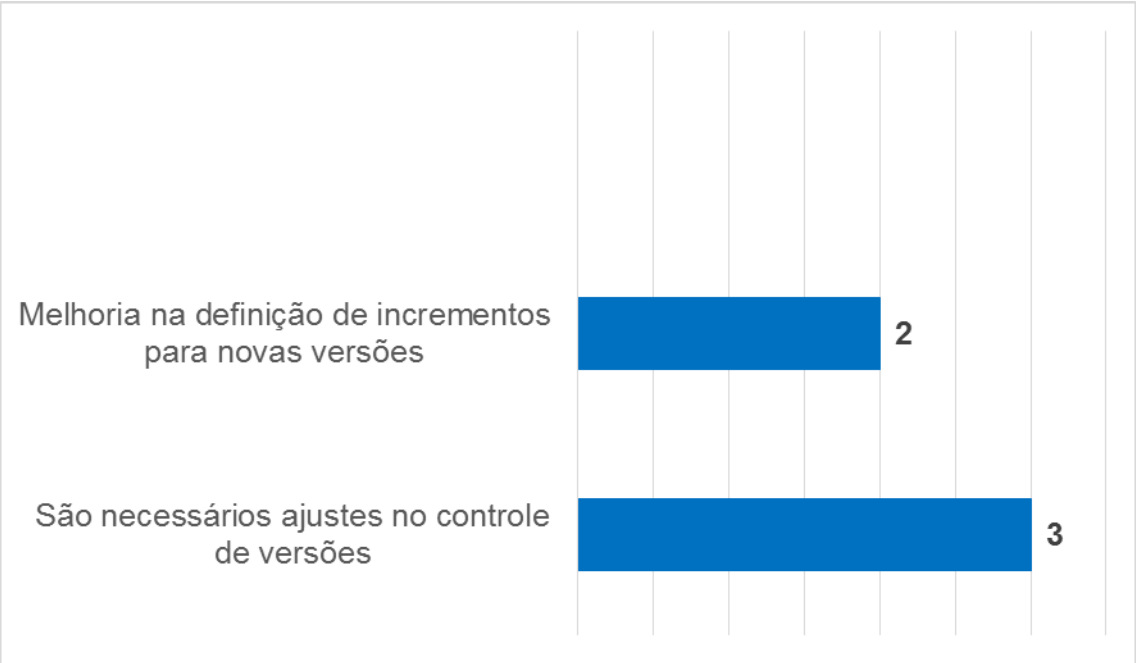
Os gráficos 7 e 8 mostram a satisfação da equipe quanto à qualidade nas entregas e no gerenciamento destas, tendo em vista que o feedback esperado pelos programadores tem chegado com qualidade e um retorno interessante, pois os responsáveis pelos testes estão mais próximos e tem uma visão do que realmente é esperado, refletindo em entregas mais consistentes para os clientes.

GRÁFICO 9 - Como você avalia a utilização do processo no contexto do gerenciamento de configuração (controle de versões, registro das mudanças, integração contínua)?



Fonte: Autor

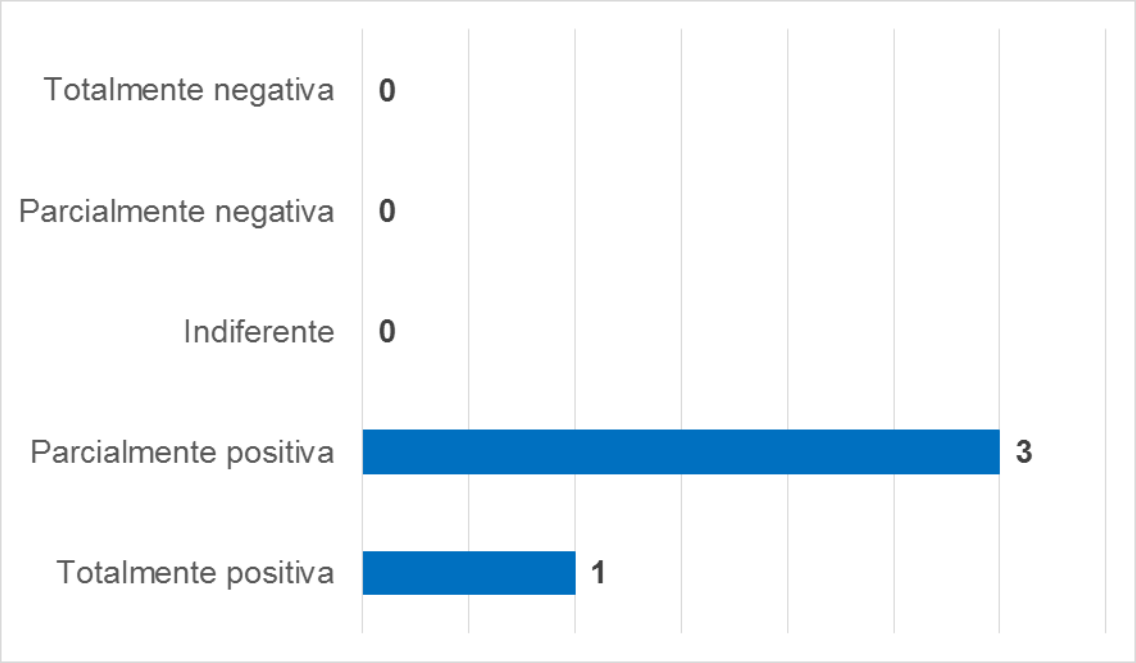
GRÁFICO 10 - Quais seriam os principais pontos a se destacar, conforme seu posicionamento na questão anterior?



Fonte: Autor

Analisando os gráficos 9 e 10 mostram que a técnica de controle de versão definida pela empresa não tem atendido as necessidades de acompanhamento e controle de versões. Por outro lado, cada nova versão possui incrementos bem definidos. É necessária uma melhor avaliação dos pontos a serem melhorados que foram identificados pela pesquisa.

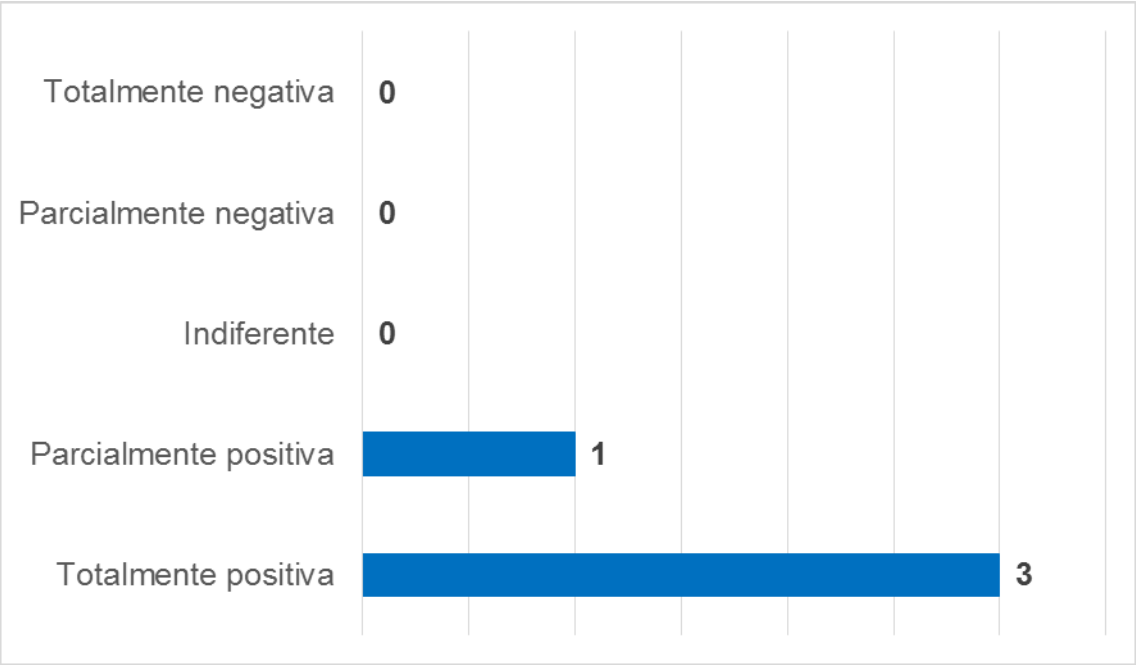
GRÁFICO 11 - Como você avalia o processo de Implantação do *Scrum* no projeto?



Fonte: Autor

O gráfico 11 apresenta a satisfação parcialmente positiva se tratando do processo de implantação. Alguns fatores podem ter sido provenientes de problemas enfrentados pela equipe, tais como, falta de uma descrição mais abrangente das solicitações e o controle das mesmas. Reunir dados antes da implantação do *Scrum* foi uma tarefa difícil, pois a equipe não tinha trabalhado antes com essa metodologia no desenvolvimento de projeto.

GRÁFICO 12 - Como você avalia a comunicação entre os membros da equipe após a implantação do *Scrum*?



Fonte: Autor

O gráfico 12 mostra a satisfação positiva após a adoção do *Scrum* em relação à comunicação entre a equipe, pois o desenvolvimento dependia de uma interação do *time* e conhecimento do projeto.

CONSIDERAÇÕES FINAIS

Este trabalho apresentou um breve estudo sobre as metodologias tradicionais e ágeis, com maior detalhamento sobre o *Scrum*, uma das metodologias mais utilizadas atualmente para gerenciamento e desenvolvimento de projeto em geral, porém mais focada em desenvolvimento de software. O trabalho pode ser utilizado para auxiliar os gerentes de projeto ou até mesmo para as organizações entenderem um pouco melhor sobre essa metodologia e verificar a possibilidade de aplicá-la em projetos ou dentro de empresas, seja ela de pequeno, médio ou grande porte.

A grande competitividade na área de desenvolvimento faz com que as organizações busquem sempre melhorar os seus serviços para se destacar dos concorrentes. Alguns fatores que beneficiam nesse diferencial são prazo de entrega e qualidade, além de uma adaptação a mudanças constantes. A utilização do *Scrum* pode facilitar as empresas a atingir esses objetivos mais facilmente, basta ser bem aplicado, independente do ambiente.

A metodologia não é a solução para todos os problemas, porém mostra uma maneira de trabalhar bastante organizada e interativa, contribuindo também com um ambiente de trabalho mais amigável, facilitando a comunicação, gerando um resultado final muito mais satisfatório.

Primeiramente este trabalho apresentou alguns conceitos básicos da engenharia de software, que apesar de antigos ainda são utilizados com muita frequência, abordou também conceitos das metodologias tradicionais, as quais foram bases para a criação de muitas metodologias ágeis e até hoje empregam um pouco dos seus valores.

Em seguida concentrou-se em estudar fundamentos teóricos, valores e práticas sobre o *Scrum*, principalmente no que diz respeito a desenvolvimento de software, detalhando cada processo da metodologia, equipe, conceitos e aplicabilidade.

Posteriormente foi visto um estudo de caso, no qual houve um desenvolvimento de software, utilizando a metodologia ágil, usando os conceitos e prática do *Scrum*. Esse mesmo estudo de caso foi utilizado para demonstrar a utilização na prática para alcançar o resultado final do projeto com suas dificuldades e falta de experiência.

Para finalizar, em trabalhos futuros é sugerido a utilização da metodologia *Scrum,* em novos projetos de ambientes corporativos, é uma ótima opção para uma melhora contínua nos projetos da empresa, e com os resultados obtidos a empresa poderá também divulgar a metodologia para outras empresas.

7 REFERÊNCIAS

PAULA FILHO, Wilson de Pádua. **Engenharia de Software:** Fundamentos, Métodos e Patrões. 3ª. ed. Rio de Janeiro: LTC – Livros Técnicos e Científicos Editora S.a, 2009

KOSCIANSKI, Andre; SOARES, Michael Dos Santos. **Qualidade de software**. 2º. ed. São Paulo: Novatec, 2007.

MAGELA, Rogério. **Engenharia de software aplicada**. Rio de Janeiro: Alta Books, 2006.

MARCHI, Késsia Rita da Costa; PAINKA, Marcelo Augusto Lima, **Utilização das Metodologias ágeis XP e Scrum para o desenvolvimento rápido de aplicações**. Universidade Paranaense, Paranavai—R, 2013. Disponível em: < http://ftp.unipar.br/~seinpar/2013/artigos/Marcelo%20Augusto%20Lima%20Painka.pdf>. Acesso em: 06 mai. 2017.

DEVMEDIA, **Agile Development:** XP e Scrum em uma Abordagem Comparativa, 2017. Disponível em: < http://www.devmedia.com.br/agile-development-xp-e-scrum-em-uma-abordagem-comparativa/30808> Acesso em: 06 mai. 2017.

PRESSMAN, Roger S. **Engenharia de software**. São Paulo: Makron Books, 2007.

SABBAGH, Rafael. **Scum**: gestão ágil para projetos de sucesso. São Paulo: Casa do Código, 2013.

SCHWABER, Ken; SUTHERLAND, Jeff. **Guia do Scrum:** Um guia definitivo para o Scum: as regras do jogo. 2011, 18p. Disponível em: < http://www.scrumguides.org/docs/scrumguide/v1/Scrum-Guide-Portuguese-BR.pdf>. Acesso em: 17 fev. 2017.

SOARES, Michel dos Santos, **Metodologia Ágeis Extreme Programming e Scrum para o Desenvolvimento de Software**. Conselheiro Lafaiete: Universidade Presidente Antônio Carlos. Disponível em: < file:///C:/Users/thay__000/Documents/Faculdade/FATEC/6%C2%BA%20Semestre/Monografia%20Revisada/Artigo%20Scrum.pdf>. Acesso em: 05 mai. 2017.

SOMMERVILLE, Lan, **Engenharia de Software**. 5° reimpressão. São Paulo : Pearson, 2010.