Universidade de Brasília - UnB
Faculdade UnB Gama - FGA
Engenharia de Software

Grata - Gerenciamento de Reuniões e ATAs

Autor: Victor Hugo Lopes Mota
Orientador: Dr. Wander Cleber Maria Pereira da Silva

Brasília, DF
2019

Victor Hugo Lopes Mota

Grata - Gerenciamento de Reuniões e ATAs

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software .

Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Orientador: Dr. Wander Cleber Maria Pereira da Silva

Brasília, DF

2019

Victor Hugo Lopes Mota
Grata - Gerenciamento de Reuniões e ATAs/ Victor Hugo Lopes Mota. – Brasília, DF, 2019-
27 p. : il. (algumas color.) ; 30 cm.

Orientador: Dr. Wander Cleber Maria Pereira da Silva

Trabalho de Conclusão de Curso – Universidade de Brasília - UnB
Faculdade UnB Gama - FGA , 2019.

1. Gerenciamento de Reuniões. 2. Otimização. I. Dr. Wander Cleber Maria Pereira da Silva. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. Grata - Gerenciamento de Reuniões e ATAs

CDU 02:141:005.6

Victor Hugo Lopes Mota

Grata - Gerenciamento de Reuniões e ATAs

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software .

Trabalho aprovado. Brasília, DF, 14 de abril de 1912:

Dr. Wander Cleber Maria Pereira da Silva
Orientador

Membro convidado 1
Convidado 1

Membro convidado 2
Convidado 2

Brasília, DF
2019

Dedicatória ficara aqui

Agradecimentos

AQUI VÃO FICAR OS AGRADECIMENTOS

Resumo

Resumo do projeto

Palavras-chaves: Gerenciamento de Reuniões, Otimização.

Abstract

Here go to abstract

Key-words: Meeting management,Optimization.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

Grata, *Gerenciamento de Reuniões e ATAs*

MAS, *Síndrome de Aceitação Sem Sentido*

NMIL, *Núcleo de Modernização da Informação Legislativa*

TCC, *Trabalho de Conclusão de Curso*

MVC, *Model-View-Controller*

RUP, *Rational Unified Process*

Sumário

1 Introdução

1.1 Apresentação do Tema

Um dos instrumentos mais fundamentais e que são cada vez mais crescentes na vida organizacional de uma empresa são as reuniões. Segundo (ALLEN, 2016), já se gastam até 15% do tempo coletivo da organização com reuniões.

Encontros institucionais que não são produtivas e sem sentido, é o que (DAVID, 2013) chama de "Síndrome de Aceitação Sem Sentido"(MAS). David define o MAS como "um reflexo involuntário em que uma pessoa aceita um convite de reunião sem sequer saber o porquê. Uma doença comum entre o escritório e trabalhadores em todo mundo". Atividades que deveriam ser simples e rápidas se tornam complicadas com várias reuniões para a execução completa delas. Reuniões não são nenhum ponto de prazer entre dentro de uma instituição, contudo se os próprios funcionários não conseguem ver o sentido da reunião e os tópicos abordados, mostra que a empresa como um todo está fadada ao fracasso.

Nesse viés, reuniões é um dos meios usados para programar uma atividade, reunir pessoas em busca de uma solução relacionada ao problema X. Composta por vários encontros sem tópicos e objetivos específicos, sem *feedbacks* aos gerentes e sem atas sobre o que foi discutido, as reuniões infelizmente se tornam um problema para gerentes e seus funcionários para as instituições. Problemas como estes podem ser pela falta de investimento disponível para aplicação na área tecnológica ou até mesmo pela priorização de outras necessidades.

Este trabalho visa propor uma solução de *software* para auxiliar a condução de reuniões, utilizando dos conhecimentos adquiridos no curso de Engenharia de Software para o desenvolvimento de uma solução que se adeque às reais necessidades empregadas aos gerentes dos projetos organizacionais.

1.2 Justificativa

Tendo como premissas os problemas em reuniões apresentados no tópico anterior (1.1), uma das soluções para aumentar a produtividade em reuniões é através de um sistema *web* que auxilie os gerentes e líderes de reuniões a gerenciar seus encontros de forma rápida, intuitiva e gratuita para que qualquer organização possa usar os recursos do *software* com o objetivo de melhorar a organização de sua empresa.

O Sistema GRATA, vem oferecer a solução prática para a melhoria do controle das

informações e qualidade dos serviços. Tendo como a principal funcionalidade o registro das Atas de reuniões de forma simples e intuitiva, tanto para quem gerencia como para quem participa. Além da automação dos processos essenciais da organização, o sistema fornece relatórios gerenciais e analíticos, que podem ser usados para identificação de pontos de melhoria ou até mesmo para dar visibilidade a questões específicas.

1.3 Problema de Pesquisa

O crescente problema com reuniões mal gerenciadas seja por gerentes não capacitados, ou por falta da especificação prévia dos tópicos a serem abordados, levam diretamente a reuniões mal sucessididas e com isso desperdício de tempo e dinheiro. Estima-se que empresas gastam em média US $ 37 bilhões anualmente em reuniões (DRAKE, 2014). O custo real desses encontros impulsionou a (HARVARD, 2016) a criar uma calculadora que ajuda gerentes a calcularem o verdadeiro custo de um encontro.

Nessa viés e utilizando a justificativa desenvolvida no tópico 1.2, é possível se ter o problema de pesquisa. A problemática a ser resolvida neste projeto é: *Como desenvolver um sistema para auxiliar a condução de reuniões em organizações?*

1.3.1 Metodologia

A metodologia abordada neste trabalho teve sua escolha baseada após a realização de pesquisas comparativas entre metodologias tradicionais como o (PMBOK, 2012) e a metodologia ágil de *software*.

Por conta de um conhecimentos maior sobre a metodologia e com entregas frequentes em menos tempo, a metodologia escolhida para este projeto no desenvolvimento do sistema foi a metodologia ágil, juntamente com algumas práticas do *Scrum* e baseado nos valores do *Lean Software Development*.

1.3.2 Requisitos

Requisito não é um termo usado apenas pela Engenharia de Software . Há casos em que requisitos são apenas uma declaração abstrata em alto nível de um serviço ou restrição que um sistema deve oferecer.

(SOMMERVILLE, 2011) os define como: "Os requisitos de um sistema são as descrições do que o sistema deve fazer, os serviços que oferece e as restrições a seu funcionamento. Esses requisitos refletem as necessidades dos clientes para um sistema que serve a uma finalidade determinada, como controlar um dispositivo, colocar um pedido ou encontrar informações."

Requisitos podem ser definidos em duas categorias: Requisitos Funcionais e Requisitos Não-Funcionais, ambos serão definidos a seguir.

1.3.2.1 Requisitos Funcionais

Os requisitos funcionais descreve o que o sistema deve de fato ser. Requisitos funcionais podem ser tão específicos quanto necessário,por exemplo, podem ter sistemas com requisitos funcionais gerais e outros que além de refletir os sistemas, também abrangem as formas de trabalho de uma organização. Requisitos funcionais de um sistema deve ser completo, isso quer dizer que todos os serviços requisitados pelo usuário devem ser definidos.

1.3.2.2 Requisitos Não-Funcionais

Requisitos não-funcionais são requisitos que são relacionados as propriedades do sistema como confiabilidade, tempo de espera, desempenho, segurança e até restrições do sistema. Requisitos não-funcionais podem possui tanta relevância quanto os requisitos funcionais, pois em uma reunião de levantamento de requisitos, o cliente sonha o mundo e não está atento se os recursos os próprios recursos e os recursos da emprega conseguem atender ao requisito. Um requisito não-funcional não atendido pode inclusive inutilizar um projeto. Exemplo disso é caso um sistema de uma aeronave não consiga atingir a confiabilidade necessária, não será dado o certificado de segurança para operar, sendo assim a aeronave não poderá voar.

1.3.3 Processo de Desenvolvimento de Software

Segundo (SOMMERVILLE, 2011), esse processo pode ser definido como "Um processo de *software* é um conjunto de atividades relacionadas que levam à produção de um produto de *software*."

Neste trabalho, foram definidas as principais atividades a serem realizadas para alcançar o objetivo final de ter um sistema gratuito que auxilie os gerentes a otimizar suas reuniões por meio computacional:

- Especificação do *software*: funcionalidades e restrições do *software*;
- Projeto e implementação do *software*: as especificações que o *software* deve atender;
- Validação de *software*: para que atenda as expectativas do cliente, o *software* deve ser validado pelo mesmo;
- Evolução do *software*: o *software* deve ser capaz de ser extensível a mudanças, tendo assim seu código aberto.

É nessa fase que são definidas a arquitetura e a linguagem de *software*.

1.3.4 Arquitetura de Software

A arquitetura de *software* é como o sistema deve ser organizado com a estrutura geral do projeto. A arquitetura possui um valor alto dentro da construção de um *software*, pois nela se tem o elo entre o projeto e a engenharia de requisitos. Possui o dever identificar os principais componentes estruturais no sistema e o relacionamento entre eles. Neste projeto a arquitetura é o MVC *(model-view-controller)*.

1.3.4.1 Model-View-Controller

O padrão arquitetural MVC é responsável de responsabilidades em camadas. A primeira é *Model*(modelo), que é responsável pela manipulação de dados, ou seja, leitura, escrita de dados e também suas validações é de responsabilidade da Model. A segunda camada é a *View*(visão), que possui a responsabilidade de interação com o usuário. Por último se tem a *Controller*(controladora), responsável por receber as aquisições do usuário. A controller também tem o dever de disponibilizar os dados para a *view* e assim ocorrer a interação com o usuário.

1.3.5 Linguagem de Software

Linguagem de programação são instruções passadas de maneira que o computador entenda e apresente um retorno. Existem diversas linguagens de programação, desde a mais baixo a alto nível.

Linguagens de *software*, como também podem ser chamadas, são divididas em duas frentes: *front-end* e *back-end*. Ambas serão explicadas nos tópicos a seguir.

1.3.5.1 Front-end

A programação de um *software* pelo ponto de vista do *front-end* é a visão final do usuário com o sistema. *Front-end* é a *View* do MVC, como explicado no tópico 1.3.4.1. Existem diversos tipos de *frameworks* que auxiliam os desenvolvedores a trabalhar com essa frente, como:

- *Bootstrap*
- *Materialize*
- *Material UI*
- *Angular 4*

A linguagem *front-end* escolhida para este projeto, foi a *Angular 4*, pois além de facilitar o desenvolvimento e interação com usuário final, é uma das mais utilizadas ao redor do mundo, então é facilita uma manutenção futura do *software*.

1.3.5.2 Back-end

A programação *back-end* possui as responsabilidades da *model* e *controller* no padrão arquitetural MVC. Em conjunto o *Angular 4*, que é a linguagem *front-end* deste projeto, a *back-end* possui o dever de tratar os dados, valida-los e fomenta-los a visão do usuário.

Existem diversas linguagens *back-end* que auxiliam os desenvolvedores a trabalhar em uma linguagem que o computador entende, como:

- *Python Django-Rest*
- *Java*
- *Ruby on Rails*
- *PHP*

A linguagem *back-end* escolhida para este projeto, foi a *Python Django-Rest*, pois tem uma ótima conexão com a linguagem *front-end*, e por ser muito utilizada, possibilita assim uma manutenção futura.

1.4 Objetivos

1.4.1 Objetivos Gerais

O objetivo do projeto é propor novos processos de reuniões e gerenciamento dos documentos gerados no ciclo de desenvolvimento dos projetos, contando com o suporte de um software gratuito e de código aberto para automatizar alguns dos trabalhos manuais, tornando os processos mais ágeis e com armazenamentos seguros.

1.4.2 Objetivos Específicos

- Criar um *software* que auxilie gerentes e líderes a terem reuniões mais objetivas;
- implementar mecanismos que permita o controle gerencial das reuniões;
- permitir o controle de todas as informações provindas das reuniões.

2 Fundamentação Teórica

2.1 Gerenciamento de Projetos

2.1.1 Modelo Tradicional

Uma metodologia de gerenciamento de projetos no modelo tradicional, de acordo com (KERZNER, 2009) é o alcance da excelência no gerenciamento de projetos se torna impossível sem um processo repetitivo que possa ser utilizado em cada projeto.

No modelo tradicional, um dos mais modelos mais utilizados é o RUP *(Rational Unified Process)*. O RUP oferece uma metodologia responsável por responder questões como boas práticas para o gerenciamento de projetos, com o objetivo de estruturar e formatar os processos associados às atividades que envolvem a tecnologia de informação.

As 4 fases principais do (RUP, 1980) pode ser vista na figura 1:

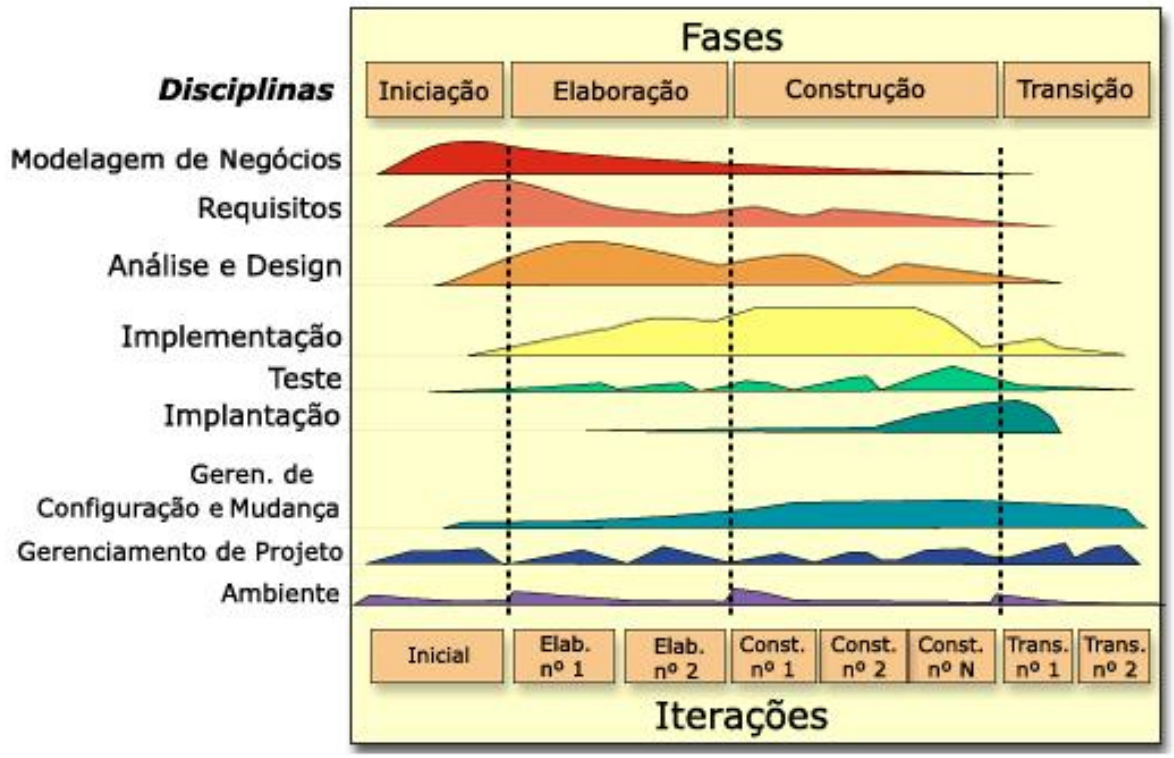

Figura 1 – Fases do RUP. Fonte: (RUP, 1980).

O desenvolvimento do plano de gerenciamento do projeto é uma atividade iterativa ao longo do ciclo de vida do projeto, sempre pronto para melhoria contínua e permitindo

à equipes do projeto definir e trabalhar com maior nível de detalhes. De acordo com o (PMBOK, 2012), as fases do (RUP, 1980) são sobrepostas, ou seja, o início de uma fase é ao término de uma outra, isso leva a algumas atividades ocorrerem de forma paralela. A maneira como este tipo de projeto aumenta os riscos, retrabalhos, e exigir recursos adicionais para permitir as atividades em paralelo, como mostrado na figura 1.

Nesta perspectiva, se tem o papel do gerente de projeto como um líder responsável por liderar a equipe para alcançar os objetivos previstos no planejamento do projeto. Entre as funções destes líderes se tem:

- Conhecimento acerca do gerenciamento de projetos;
- Desempenho para aplicar seus conhecimentos na prática;
- Comportamento pessoal de liderança, atingindo objetivos e equilibrando restrições.

Este tipo de gerenciamento de projeto é mais utilizado em empresas já consolidadas, de ramo mais formal, que possui mais burocracia em seus projetos e por tanto maior rigor de documentação e de liderança dos gerentes de projeto. Essa hierarquia pode ser vista na figura 2.

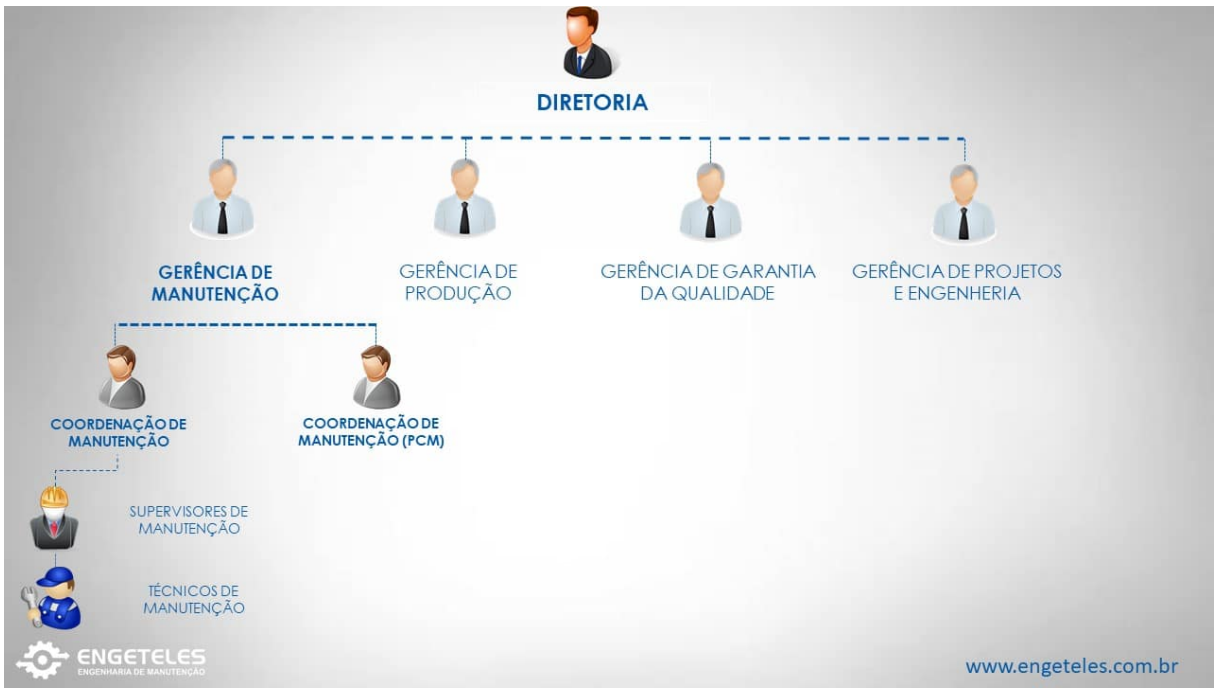

Figura 2 – Hierarquia em projetos tradicionais. Fonte: (JHONATAS, 2018).

2.1.2 Modelo Ágil

Em contraposição ao modelo tradicional, surte o manifesto ágil como uma reação contra o processo burocrático presente no modelo tradicional, que possuem por característica atividades sequências em modelo cascata. Segundo a (STANDISH, 2014) apenas 16,2% dos projetos entregues por companhias americanas foram entregues respeitando prazos, custos previamente acordados e objetivos determinados. Segundo a própria (STANDISH, 2014), as principais causas destes problemas estavam relacionadas com o modelo sequencial tradicional.

O modelo ágil, segundo (SOARES, 2009), ela deve primeiro aceitar as mudanças em vez de tentar prevê-las, agir de maneira rápida sabendo receber, avaliar e responder como elas devem ser respondidas. As principais características da metodologia ágil são:

- Desenvolvimento iterativo e incremental;
- Comunicação;
- Documentação extensiva;

Em 2001, membros da comunidade de *software* se reuniram e criaram o (AGILE, 2001). O objetivo deste manifesto é utilizar as melhores práticas observadas em projetos anteriores que obtiveram sucessos.

Os principais conceitos do manifesto ágil são:

- Indivíduos e interações ao invés de processos e ferramentas;
- *Software* executável ao invés de documentação;
- Colaboração do cliente ao invés de negociação de contratos;
- Resposta rápida a mudanças ao invés de seguir planos pré-estabelecidos.

Uma das boas práticas adotadas ao modelo ágil é o SCRUM. O SCRUM, se refere ao jogo *Rugby*, que é a ação dos jogadores se organizarem em círculo para planejar a próxima jogada. Um dos principais pontos de vista do SCRUM é mostrar um projeto com pequenos ciclos, aumentando as iterações entre os participantes, mas com visão a longo prazo.

O ciclo de vida pode ser visto na figura 3:

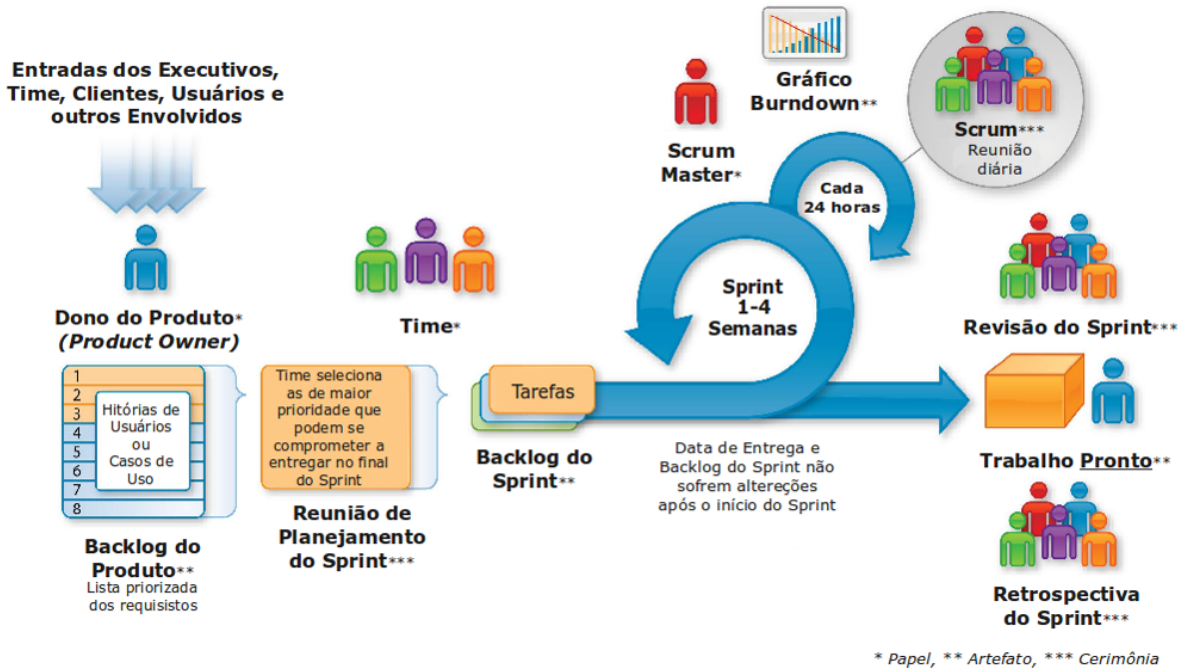

Figura 3 – Ciclo de Vida SCRUM. Fonte: (FABIANE, 2016).

Como visto na figura 3, o SCRUM é um ciclo progressivo de várias iterações bem definidas, denominadas *Sprints*. As *Sprints* podem ter duração de uma a quatro semanas. Antes de cada *Sprint*, deve ser realizada a reunião de planejamento da *Sprint*, chamada de *Sprint Planning Meeting*, na qual os desenvolvedores tem contato com o *Product Owner*, que possui o dever de priorizar as atividades, seleciona-las e estimar as tarefas que a equipe pode desenvolver na próxima *Sprint*.

Com o objetivo de saber o progresso de cada equipe dentro da *Sprint*, ocorrem as reuniões diárias, denominadas *Daily Meetings*, que tem duração de no máximo 15 minutos e ocorrem com todos os participantes em pé, respondendo perguntas como: "O que você fez ontem?", "O que você fez hoje?"e "O que você vai fazer amanhã?".

Ao final de uma *Sprint* é feita uma análise gráfico do progresso do projeto atráves do *Sprint Backlog* durante a *Sprint Review*. Após a *Sprint Review* ocorre a *Sprint Restrospective* que é a análise de experiências que ocorreram durante a *Sprint* sejam boas ou não a fim de melhora-las.

Segundo (FOWLER, 2005), as equipes devem possuir um quadro para registro das atividades, denominado *Kanban*. O *Kanban* possui o objetivo de auxiliar as equipes em relação ao progresso da *Sprint*, esse quadro pode ser dividido em 4 fases:

- Para fazer;
- Em andamento (com o nome do responsável pela atividade);

- Em revisão;
- Feito.

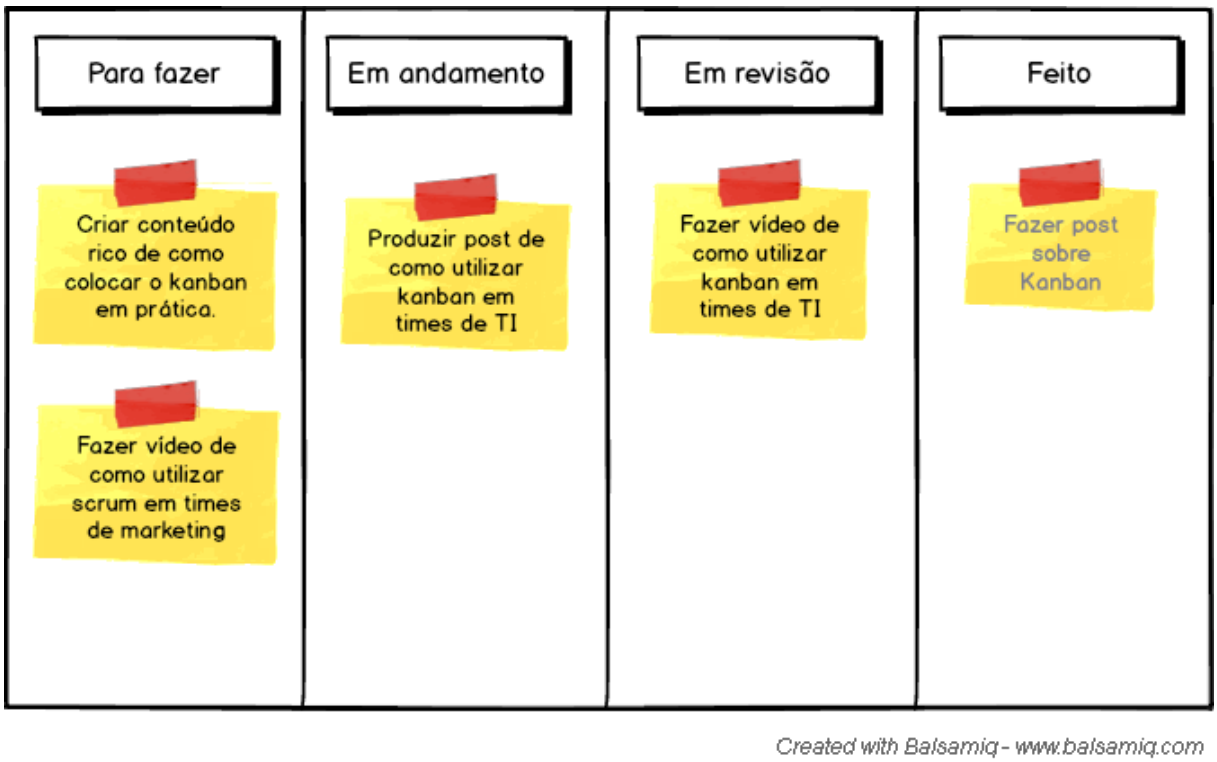

Figura 4 – Quadro Kanban. Fonte: (LAMECK, 2016).

No modelo ágil os requisitos dos clientes podem ser mudados a qualquer momento, e o time de gerência e desenvolvimento devem estar preparados para conversar com o cliente a fim de resolver as alterações de requisitos da melhor maneira possível. Este tipo de pensamento no modelo tradicional é mais difícil de acontecer, pois ao observar a figura 1, é possível notar ao iniciar uma fase, essa mesma fase não é retornada mais tarde, ou seja, no modelo tradicional uma troca de requisitos pode levar ao reinicio do projeto.

Este modelo é mais focado para empresas emergentes, que não são muito rigorosas em seus processos e aceita que mudanças nos requisitos ou na visão do produto são sempre bem vindas, desde que melhore o projeto final.

3 Metodologia

4 Proposta de Solução

4.1 Definição dos Requisitos

4.1.1 Requisitos Funcionais

4.1.2 Requisitos Não-Funcionais

5 Planejamento de Desenvolvimento

5.1 Cronograma TCC

6 Conclusão

Referências

AGILE, M. *Manifesto for Agile Software Development*. 2001. Disponível em: <http://www.agilemanifesto.org>. Acesso em: 05.05.2019. Citado na página 19.

ALLEN, L.-W. Meetings as a positive boost? how and when meeting satisfaction impacts employee empowerment. *Journal of Business Research*, 2016. Acesso em: 25.04.2019. Citado na página 12.

DAVID, G. *How to save the world (or at least yourself) from bad meetings*. 2013. Disponível em: <https://www.ted.com/talks/david_grady_how_to_save_the_world_or_at_least_yourself_from_bad_meetings>. Acesso em: 22.04.2019. Citado na página 12.

DRAKE, B. 37 billion is lost every year on these 12 meeting mistakes. *Business Insider*, 2014. Disponível em: <https://www.businessinsider.com/37-billion-is-lost-every-year-on-these-meeting-mistakes-2014-4>. Acesso em: 29.04.2019. Citado na página 13.

FABIANE, S. *Ciclo de Vida do Scrum*. 2016. Disponível em: <https://br.pinterest.com/pin/417357090459098830/>. Acesso em: 05.05.2019. Citado 2 vezes nas páginas 8 e 20.

FOWLER, M. The new methodology. 2005. Disponível em: <https://moodle2016-17.ua.es/moodle/pluginfile.php/69142/mod_resource/content/1/martin-fowler-the-new-methodology.pdf>. Acesso em: 05.04.2019. Citado na página 20.

HARVARD, B. *Estimate the Cost of a Meeting with This Calculator*. 2016. Disponível em: <https://hbr.org/2016/01/estimate-the-cost-of-a-meeting-with-this-calculator>. Acesso em: 29.04.2019. Citado na página 13.

JHONATAS, T. *PCM Descomplicado – Planejamento e Controle de Manutenção*. 2018. Disponível em: <https://engeteles.com.br/pcm-descomplicado/>. Acesso em: 04.05.2019. Citado 2 vezes nas páginas 8 e 18.

KERZNER, H. Gestão de projeto 2ª edição. *Bookman Editora*, 2009. Acesso em: 03.05.2019. Citado na página 17.

LAMECK, O. *Uma Breve Introdução ao Kanban*. 2016. Disponível em: <https://blog.diferencialti.com.br/uma-breve-introducao-ao-kanban/>. Acesso em: 04.05.2019. Citado 2 vezes nas páginas 8 e 21.

PMBOK, P. *Um Guia do Conhecimento em Gerenciamento de Projetos 5ª edição*. [S.l.]: Saraiva, 2012. Acesso em: 30.04.2019. Citado 2 vezes nas páginas 13 e 18.

RUP, I. *RUP - Rational Unified Process*. 1980. Disponível em: <http://www.anisio.eti.br/index.php/sistemas-de-informacao-menuvertical/conceito-de-sistema/item/47-rup-rational-unified-process>. Acesso em: 04.05.2019. Citado 3 vezes nas páginas 8, 17 e 18.

SOARES, S. Metodologias Ágeis extreme programming e scrum para o desenvolvimento de software. 2009. Acesso em: 05.04.2019. Citado na página 19.

SOMMERVILLE, I. *Engenharia de Software 9ª edição*. [S.l.]: Pearson Education do Brasil, 2011. Acesso em: 01.04.2019. Citado 2 vezes nas páginas 13 e 14.

STANDISH, G. Failure record. *Standish Group Report Chaos*, 2014. Disponível em: <https://www.projectsmart.co.uk/white-papers/chaos-report.pdf>. Acesso em: 05.04.2019. Citado na página 19.