FACULDADE DE ADMINISTRAÇÃO E NEGÓCIOS DE SERGIPE
ESPECIALIZAÇÃO EM GESTÃO DE PROJETO DE SOFTWARE 2

DIEGO RABELO DE FRANÇA

APLICABILIDADE DOS CONCEITOS DO SCRUM NO GERENCIAMENTO DAS PRÓPRIAS DEMANDAS DE MANUTENÇÃO DE SISTEMAS

ARACAJU
2017

DIEGO RABELO DE FRANÇA

APLICABILIDADE DOS CONCEITOS DO SCRUM NO GERENCIAMENTO DAS PRÓPRIAS DEMANDAS DE MANUTENÇÃO DE SISTEMAS

JUSTIFICATIVA

O estudo a ser aplicado se justifica pela relevância do tema diante da importância na área de TI, no desenvolvimento de software, na capacidade adaptativa das metodologias ágeis e no gerenciamento das demandas de manutenção de sistema.

ORIENTADOR: Alércio Bressano
COORDENADOR: Luciano Cerqueira Passos

ARACAJU
2017

DIEGO RABELO DE FRANÇA

APLICABILIDADE DOS CONCEITOS DO SCRUM NO GERENCIAMENTO DAS PRÓPRIAS DEMANDAS DE MANUTENÇÃO DE SISTEMAS

Trabalho de Conclusão de Curso apresentado ao Núcleo de Pós-graduação e Extensão – NPGE, da Faculdade de Administração e Negócios de Sergipe – FANESE, como requisito para obtenção do título de Especialista em Gestão de Projeto de Software2.

_____

Nome completo do Avaliador

_____

Nome completo do Coordenador do Curso

_____

Nome completo do Aluno

_____

Aprovado (a) com média: _____

Aracaju (SE), ___ de _____ de 2017

RESUMO

Os sistemas computacionais (ou softwares como conhecido na língua inglesa) cresceram, ficaram mais complexos e importantes para as empresas. Com intuito de acompanhar esse crescimento, a engenharia de software precisou criar metodologias de desenvolvimento de softwares para assim garantir a qualidade e entregar sistemas. As metodologias ágeis surgiram para tentar atender essas necessidades. O *scrum* é a mais conhecida das metodologias ágeis, ele é um framework de gerenciamento e propõe um modelo de desenvolvimento de software rápido, apto a modificações, adaptações e bastante eficaz, garantindo a qualidade e a sua entrega. Este artigo apresenta uma introdução sobre a evolução da engenharia de software até os métodos ágeis, desenvolve-se sobre o manifesto ágil e *scrum* e contem um estudo de caso a respeito de uma aplicação de seus conceitos no gerenciamento das próprias demandas de manutenção de sistemas e por fim as considerações finais.

Palavras chaves: Engenharia de Software. Framework SCRUM. Metodologia Ágil.

LISTAS DE GRAFICOS

LISTAS DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1. INTRODUÇÃO

No início da era tecnológica, havia-se uma grande preocupação com os hardwares em relação aos softwares, isso porque, muitas máquinas eram responsáveis apenas pela elaboração de relatórios. Segundo Pressman (1995), nesta época existia um grande desafio: desenvolver um hardware que reduzisse o custo de processamento e melhorasse a armazenagem de dados.

Na década de 80, com os avanços da microeletrônica, os poderes computacionais aumentaram e os custos dos computadores começaram a reduzir, ficando cada vez mais acessível à população. "O poder de um computador mainframe da década de 80 agora está à disposição sobre uma escrivaninha" (Pressman, 1995). Assim surgiu o *Personal Computer* (PC) ou PC-Desktop, em uma tradução para língua portuguesa: computador pessoal ou computador pessoal sobre a mesa. Estes eram pequenos e com grande potencial computacional se comparado com outros computadores da época.

Na década de 90, o grande desafio era melhorar a qualidade e reduzir ainda mais o custo de soluções baseadas em computador, tais soluções seriam implementadas com softwares. O aumento do poder computacional das maquinas fez crescer a procura por softwares como mecanismo de aproveitamento e utilização desse potencial computacional. A preocupação com os hardwares agora estava equiparada com a busca de softwares cada vez mais eficientes. Essa mudança levou os estudiosos a chamarem a década de 90 como "revolução do computador" (Pressman, 1995) e "nova revolução industrial" Osborne *apud* (Pressman, 1995).

"O software agora ultrapassou o hardware como chave para o sucesso de muitos sistemas baseados em computador" (Pressman, 1995). Eles começaram a fazer parte do negócio da empresa: dirigindo, controlando e capacitando. Em outras palavras, sistemas começaram a ser "fator diferencial" de uma empresa no mercado capitalista mundial. Entretanto, a construção de um software novo e com qualidade, tornava-se uma tarefa terrível e complexa. Para minimizar aversões e complexidade ao desenvolvimento de um software novo, muitos profissionais procuraram a Engenharia como base para desenvolvê-los.

Pressman foi o primeiro que usou o termo *Software Engineer* (traduzido Engenharia de Software) que até então era pouco difundida pela falta de popularização dos softwares. Ele utilizou sólidos princípios de engenharia para que pudesse obter economicamente a construção de um software confiável e eficaz para máquinas reais. Em suma, Engenharia de

Software é uma área de conhecimentos que presume conjunto de práticas da engenharia civil para a construção de um software confiável. Assim, muitos profissionais vieram a utilizar estas práticas no desenvolvimento de seus softwares como requisito de credibilidade para seus clientes.

Ao decorrer do tempo, a engenharia de software evoluiu, os softwares ficaram ainda mais complexos e importantes, aumentaram a exigência de tempo, custo e qualidade no seu desenvolvimento. Por consequência, destacou-se o aumento do índice de projetos com insucesso *(Fracasso/Falha/Atraso/Prejuízo)*, ou seja, elevaram o nível de insatisfação do cliente final com a empresa desenvolvedora ou a equipe mantedora de seu software. Como é possível observar no gráfico abaixo.

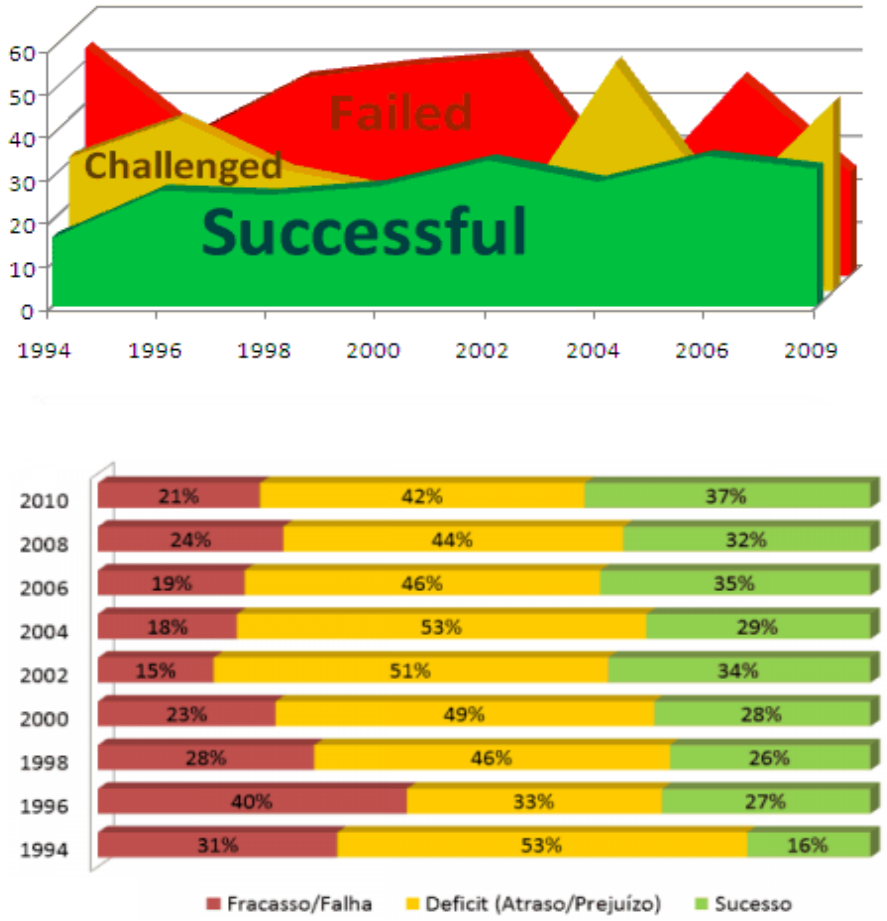

GRAFICO 1 :Índice de sucesso ou insucesso de projetos no mundo.
FONTE: (COSTA, 2008) e (D'ÁVILA, 2011).

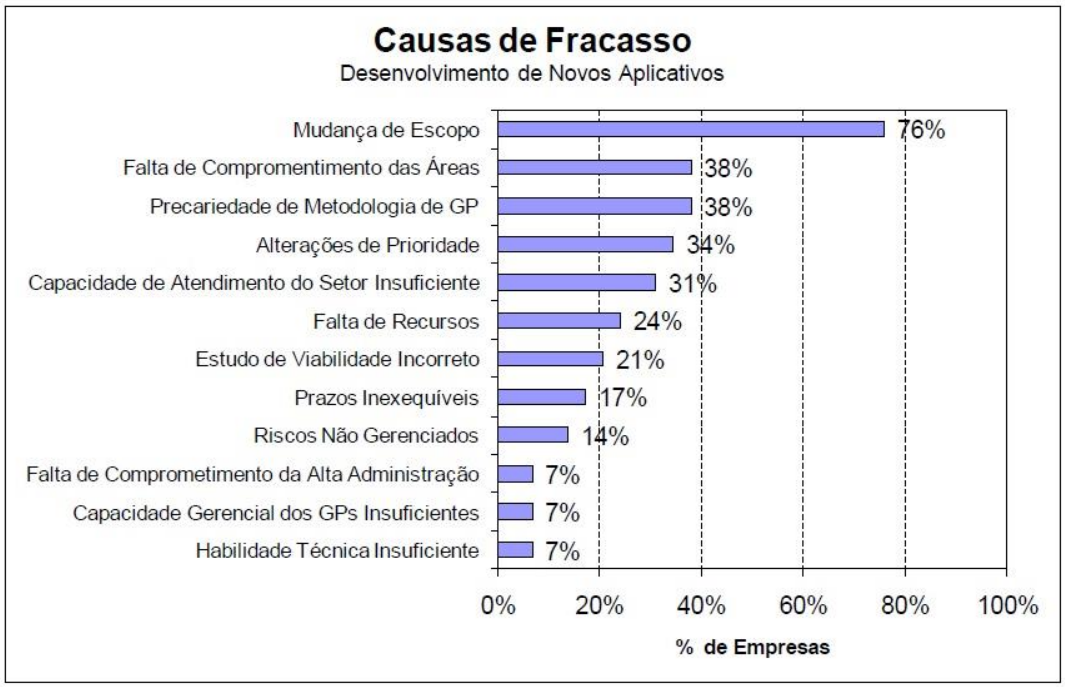

GRAFICO 2 : Causas de Fracasso Desenvolvimento de Novos Aplicativos.
FONTE: (FIRPO, 2011)

Diante desses insucessos e casos de fracassos no desenvolvimento de novos aplicativos, a quantidade de produtos equivocados cresceram. Alguns defeitos nos produtos de software são descobertos pelos usuários mesmo aqueles desenvolvidos com os melhores processos (FILHO, 2013). Assim, a área de manutenção de sistemas tornou-se de grande relevância, pois ela viria a ser a responsável por amenizar esses impactos negativos gerados pelos defeitos. Porém, isso só ocorre se ela funcionar de forma adequada, gerenciando suas demandas de acordo com a expectativa do cliente.

O manifesto ágil e as metodologias ágeis de desenvolvimento surgiram para diminuir os índices de projetos com insucesso e o nível de insatisfação dos clientes, além de quebrar alguns paradigmas da engenharia clássica de software como a valorização dos artefatos, ferramentas, procedimentos e rigidez dos requisitos em relação flexibilidade, colaboração, motivação e satisfação diante o processo de produção de um software. Os métodos ágeis, como são conhecidas essas metodologias, auxiliam todo o processo de criação de um produto de *software*, desde a sua concepção, interação da equipe até a entrega do produto final, enfatizando a preocupação com a qualidade até o bem estar do cliente.

Dentre os métodos mais conhecidos, destaca-se o framework *scrum*. Segundo FERNANDES, MEIRELLES E PICININI (2015) *apud* ABRAHAMSSON, esse framework fornece um conjunto de práticas de gerenciamento de projeto. Devido a sua capacidade

adaptativa e facilidade de entendimento, suas praticas e conceitos podem ser utilizados de forma independente e em diversas situações/contexto como, por exemplo, no gerenciamento das próprias demandas de manutenção de sistemas já implantados ou entregues.

2. REFERÊNCIAL TEÓRICO

2.1 MANIFESTO ÁGIL

Um grupo de engenheiros de software, formado na época por futuros criadores de metodologias ágeis como Kent Beck e Robert C Martin (XP – Extreme Programming), Ken Schwaber e Jeff Sutherland (*SCRUM*), se uniram, analisaram as causas e fracassos de um projeto de desenvolvimento de software e formularam 12 pensamentos que vieram a ser os princípios por trás do manifesto ágil. Esses princípios são:

(1) Nossa maior prioridade é satisfazer o cliente através de entrega continua e adiantada de software com valor agregado.

(2) Mudanças nos requisitos são bem-vindas mesmo tardiamente no desenvolvimento, ou seja, processos ágeis tiram vantagem da mudança visando vantagem competitiva para o cliente.

(3) Entregar frequentemente software funcionando, de tempos em tempos, com preferência à menor escala de tempo.

(4) Pessoas de trabalho e de negócio devem trabalhar diariamente em conjunto por todo o projeto.

(5) Construa projetos em torno de indivíduos motivados, ou seja, dê a eles o ambiente e o suporte necessário e confie neles para fazer o trabalho.

(6) O método mais eficiente e eficaz de transmitir informações para uma equipe de desenvolvimento é através de conversa face a face.

(7) Software funcionando é a medida primária para o progresso.

(8) Os processos ágeis promovem desenvolvimento sustentável. Em outras palavras, os patrocinadores, desenvolvedores e usuários devem ser capazes de manter um ritmo constante indefinidamente.

(9) Contínua atenção a excelente técnica e bom design aumentam a agilidade.

(10) A arte de maximizar a quantidade de trabalho não realizado é essencial.

(11) As melhores arquiteturas, requisitos e designs surgem de equipes auto-organizadas.

(12) Em intervalos regulares, a equipe precisa se reunir para que reflita sobre como se tornar mais eficaz e então possa se ajustar seu comportamento de acordo.

A partir desses princípios citados, alguns valores tiveram suas importâncias alteradas, por exemplo:

- Indivíduos e interações mais que processos e ferramentas – Não adianta ter processos e ferramentas bem definidos e não cumprir com prazo e com a qualidade, rompendo assim a confiança do cliente.
- Software em funcionamento mais que documentação abrangente – A documentação é importante, no entanto, devem-se evitar os excessos, documentar apenas o necessário afinal o objetivo final do desenvolvimento de software é o produto funcionando corretamente da maneira que foi solicitado.
- Colaboração com o cliente mais que negociação de contratos – Trazer o cliente para dentro do processo de desenvolvimento fará enxergar coisas que antes não enxergavam.
- Responder a mudanças mais que seguir um plano – É importante ter a capacidade de se adaptar, afinal o desenvolvimento de um software não representa uma linha horizontal constante, pois requisitos mudam a toda instante.

Vale ressaltar que mesmo havendo valor no item da direita, o manifesto valoriza mais o item da esquerda.

2.2 SCRUM

A palavra *scrum* é oriunda de uma jogada de *rugby*, no qual o *team* se une para formar uma muralha. Nessa jogada, o trabalho em equipe é importantíssimo, pois se um dos jogadores da formação falhar, toda a jogada é comprometida. A figura abaixo reflete a jogada e como é a formação.

FIGURA 1: O *scrum* do *rugby*.
FONTE: (WIKIPÉDIA, 2012).

Em 1986, Nonaka e Takeuchi, já utilizavam princípios que futuramente fariam parte do *scrum*. Nonaka e Takeuchi (1997) destacavam a importância de crenças, compromissos, situações e interações apropriadas para que as informações fossem convertidas em conhecimento e estas pudessem circular pela organização, influenciando positivamente e diretamente os comportamentos, julgamentos e atitudes das pessoas.

Em 1995, Ken Schwaber formalizou um conjunto de definições chamando de *scrum* e ajudou a implantá-lo no desenvolvimento de software em todo o mundo. Seu foco era a entrega do produto final no prazo determinado com qualidade, o comprometimento dos envolvidos e o compartilhamento do conhecimento entre eles, a fim de conquistar a confiança do cliente.

Para Beedle e Schwabber (2002), *scrum* é um processo de desenvolvimento iterativo e incremental para gerenciamento de projetos e desenvolvimento de software. Ele utiliza a ideia do ciclo P.D.C.A. (*Plan, Do, Check, Act* - Planejamento, Execução, Checagem e Ação) e o

seu processo não é avaliado enquanto se está rodando. Ou seja, um produto finalizado é um produto que passou por todas as fases.

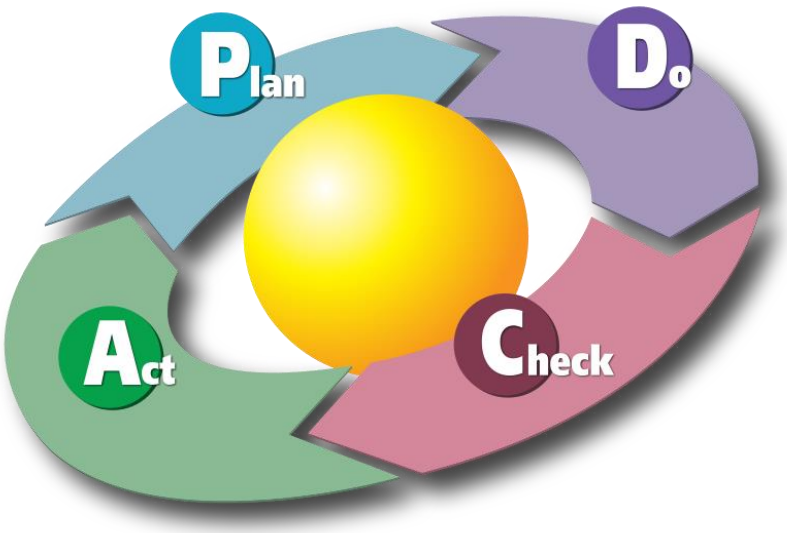

FIGURA 2: Ciclo PDCA.
FONTE: (WIKIPÉDIA, 2009).

Teoricamente, o framework *scrum* pode ser aplicado em qualquer contexto no qual um grupo necessite trabalhar unido para atingir um objetivo comum, valorizando assim o trabalho em equipe.

2.3 TERMINOLOGIAS

2.3.1 PAPEIS

São todas as funções atribuídas às pessoas envolvidas no framework. Elas terão funções importantes e serão objetos chaves para o sucesso do seu funcionamento. Os papeis são:

a) Scrum Master (SM)

Responsável pela visão de negócio do projeto, o *scrum master* atua como facilitador e mediador garantindo que o *team* seja sempre produtivo e nada interfira no processo de desenvolvimento. Ele pode ser um membro do *team* ou alguém externo.

b) Product Owner (PO)

É o dono do produto ou cliente final. Ele possui a visão do retorno que o projeto trará para a empresa e para os envolvidos, logo sua missão é cuidar do *product backlog*, planejar *releases* (liberações), priorizar requisitos e passar ao time uma visão clara sobre os objetivos do projeto.

c) Team

Conjunto de pessoas que implementará o projeto e executará o *sprint*. Responsável por entregar a soluções de forma rápida e com qualidade. Geralmente o *team* é formado por um grupo pequeno (entre 5 e 9 pessoas) e que tenta trabalhar de forma auto gerenciada.

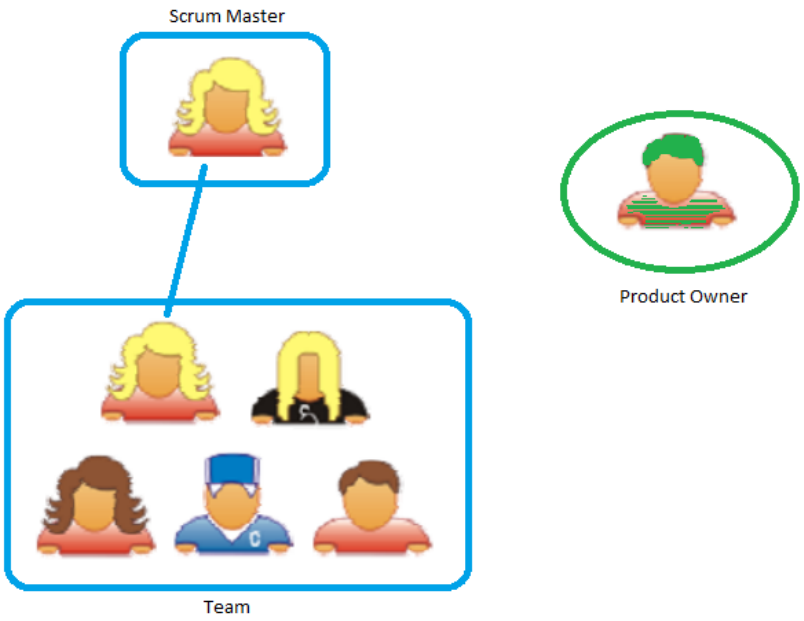

FIGURA 3: Papéis do *scrum*.
FONTE: (BARBOSA; LIBARDI, 2010).

2.3.2 ARTEFATOS

São todos os documentos gerados com a utilização do framework.

a) Product Backlog (PB)

O *product backlog* é uma lista de funcionalidades que compõem a construção de um produto. Para SCHWABER (2004), as funcionalidades de um sistema ou produto que está sendo desenvolvido são listadas no *product backlog*. "O *product backlog* é dinâmico" (SCHWABER, 2004), ou seja, ele pode ser alterado de acordo com a necessidade do produto e do cliente. Uma funcionalidade só pode ser alterada se, somente se, ela não estiver sendo rodada no *sprint*. Essa lista é definida, priorizada e avaliada pelo *product owner*. A tabela abaixo exemplifica um *product backlog.*

Product Backlog (Loja Virtual - Modulo de Compra)	R$ -	R$ -
Estória	**Business Value(BV - R$)**	**Store Point (SP - Serie Fibonnaci - 1,2,3,5,8,13,21,34,55,89,144,233,377,...)**
Logar usuário	R$ 10.000,00	1
Gerar boleto	R$ 15.000,00	3
Exibir forma de pagamento	R$ 9.000,00	3
Confirmar dados endereço de entrega	R$ 8.000,00	3
Consultar status do pedido (Empacotando ... Entregue ..)	R$ 10.000,00	5
Consultar status da comprar(Aprovada-Reprovada-Expirada-Não Realizado pagamento...)	R$ 12.000,00	8
Finalizar compra	R$ 10.000,00	8
Solicitar compra operadora de cartão	R$ 6.000,00	5
Detalhar produto	R$ 5.000,00	5
Resgatar Cupom	R$ 3.000,00	3
Detalhar fornecedor do produto	R$ 2.000,00	3
Detalhar categoria do produto	R$ 2.000,00	3
Manter produto no carrinho	R$ 5.000,00	8
Listagem de produtos com filtro (Categoria/Descrição/Fornecedor/Etc)	R$ 3.000,00	8
	Total	66

TABELA 1: *Product backlog*.
FONTE: Elaborado pelo autor.

b) Sprint Backlog (SB)

"O *sprint backlog* define o trabalho ou as tarefas do *team*" (SCHWABER, 2004). Em outras palavras, é uma lista de funcionalidade que o *team* desenvolverá ao longo de um *sprint*. Essa lista é composta pelas funcionalidades que têm maior importância para o *product owner* e potencialmente rentáveis para o *team*. Ela é definida pelo *team* no *sprint planning meeting*. A figura abaixo mostra um exemplo do *sprint backlog.*

FIGURA 4: *Sprint backlog*.
FONTE: (LARSSON, 2009).

c) Burndown Chart

É um gráfico de monitoramento que serve para avaliar o andamento do projeto. No *scrum* existem dois tipos de gráficos:

- *Product ou Release burndown* avalia a quantidade de pontos trabalhados e tempo utilizado (*sprint)* em relação à quantidade de pontos e tempo total para construção do produto.
- *Sprint burndown* avalia a quantidade de pontos entregues e tempo utilizado (*dias*) em relação à quantidade de pontos e tempo total de um *sprint*. Pode-se então concluir que cada *sprint* possuirá seu próprio *burndown*.

De Barbosa e Libardi (2010), estes gráficos mostram ao longo do tempo a quantidade de trabalho que ainda resta ser feito, sendo um excelente mecanismo para visualizar a correlação entre a quantidade de trabalho e o progresso do time. É importante ressaltar que o *team* é responsável pela atualização dos gráficos de forma explicita, ou seja, os gráficos ficam expostos para que todos possam ver suas atualizações, acompanhar o processo do *sprint* e do projeto. A figura abaixo demonstra um exemplo de *burndown chart* do *scrum*.

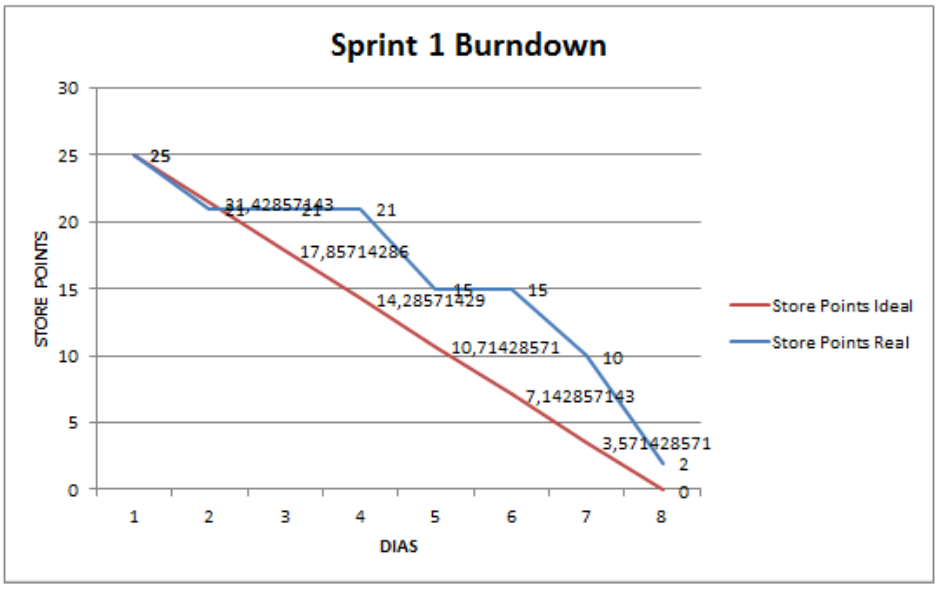

GRAFICO 3 :*Sprint burndown*.
FONTE: Elaborado pelo autor.

2.3.3 CERIMÔNIAS

São encontros ou reuniões feitas para promover a comunicação a fim de compartilhar informação. Elas são:

a) Release Planning Meeting ou Product Planning Meeting

É a reunião inicial na qual estão presentes o *product owner* e o *scrum master* para descrever, planejar e avaliar de forma geral e macro as necessidades para a criação do produto. Neste encontro eles geram o artefato chamado de *product backlog*.

b) Sprint Planning Meeting

Segundo Schawber (2004), é uma reunião com duração de 8 horas que inicia cada *sprint*.

Este encontro é dividido em dois segmentos de 4 horas cada:

1 No primeiro segmento, "o *product owner* apresentará os itens de maior prioridade do *product backlog* para o *team*" (SCHWABER, 2004). Isto é, o *product owner* irá dizer qual dos itens listados no *product backlog* é mais importante para ele. O *team* tira suas dúvidas a respeito das funcionalidades especificadas.

2 No segundo segmento, o *team* planeja quais itens serão colocados no *sprint backlog* de acordo com o custo x benefício x dificuldade. "O *team* planeja o *sprint backlog*" (SCHWABER, 2004).

c) Daily Scrum Meeting

Para Schwaber (2004) é um encontro para o *team* sincronizar as tarefas com os outros membros, compartilhar informações sobre o andamento do *sprint* e reportar os impedimentos para que o *scrum master* possa resolvê-los.

Dentre as características desse encontro, devem-se destacar as básicas:

- Ter no máximo 15 minutos.
- Ser feita com os membros do *team* "em pé" para que não haja prolongamento e cada membro deve responder as seguintes perguntas:
  - O que você fez ontem?
  - O que você fará hoje?
  - Há algum impedimento no seu caminho, seja ele físico (maquinas, redes, etc), intelectual (conhecimento, capacidade, etc) ou emocional (*bad days*).
- Ser realizada no mesmo lugar.
- Discussões sobre o *sprint* são bem-vindas.
- Ser facilitada pelo *scrum master* para que o assunto da reunião não seja desviado.
- Questões levantadas devem ser levadas para fora da reunião e normalmente tratadas por um grupo menor de pessoas que tenham a ver diretamente com o problema ou possam contribuir para solucioná-lo.

É importante ressaltar que o *daily scrum* não é uma reunião de status na qual um chefe fica coletando informações sobre quem está atrasado. Ao invés disso, é uma reunião na qual os membros do *team* assumem compromissos perante os demais.

d) Sprint Review Meeting

É um encontro entre o *product owner* e o *scrum master*. Essa reunião acontece sempre no final de cada *sprint*. Seu objetivo principal é apresentar para o *product owner* o que

foi desenvolvido durante o *sprint*. Segundo Schwaber (2004), essa reunião é planejada para ser realizada em no máximo 4 horas.

e) Sprint Retrospective Meeting

"É uma reunião de 3 horas" (SCHWABER, 2004). Essa reunião ocorre também no final de cada *sprint*. Tem objetivo de identificar:

- O que funcionou bem.
- O que pode ser melhorado e quais ações serão tomados para melhorar.

2.3.4 OUTRAS

a) Sprint

É a unidade básica de tempo de desenvolvimento em *scrum*. Um *sprint* pode durar de 1 a 4 semanas. "O período de tempo de um *sprint* é de no máximo 30 dias e ele se repete a cada *sprint backlog*" (SCHWABER, 2004), ou seja, é um ciclo de desenvolvimento e nele são desenvolvidas as tarefas listadas no *sprint backlog*.

b) Velocity

Para Schwaber (2004), *velocity* é a capacidade máxima de produção de um *team* em um determinado intervalo de tempo (*sprint*). Ela é definida pelo *team*, pois é necessário o comprometimento de todos.

c) Dashboard

É o quadro de trabalho que conterá o *sprint backlog* e o *burndown chart* e estará a amostra do *team*.

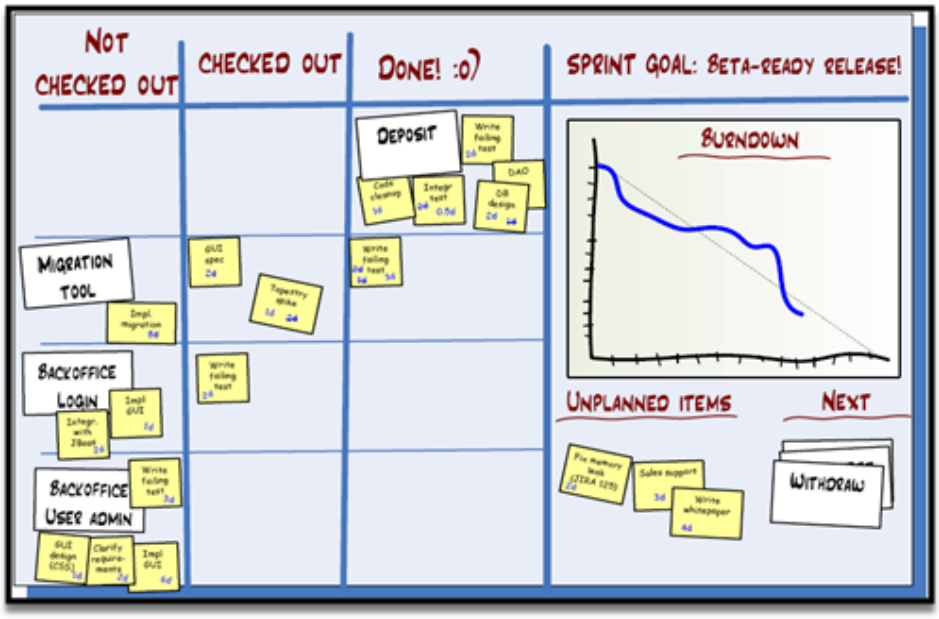

FIGURA 5: Dashboard.
FONTE: (MALMQUIST, 2011)

2.4 ALGORITMO DE FUNCIONAMENTO DO SCRUM

Abaixo segue a sequência do funcionamento do *scrum*:

Primeiro passo: o *scrum master* faz o *product* ou *release planning meeting* com o *product owner*.

Segundo passo: existe tarefa no *product backlog*? Se sim, o *team* faz-se *sprint planning meeting* para priorizar, avaliar e definir o *sprint backlog*. Senão, vai para **oitavo passo**.

Terceiro passo: o *team* obtém o *sprint backlog* e inicia o *sprint*.

Quarto passo: o *team* executam as tarefas do *sprint backlog* e cada ciclo de 24 horas é feito o *daily planning meeting* para eliminar as dificuldades.

Quinto passo: terminou o tempo do *sprint*? Se sim, o *team* entrega o *sprint backlog* finalizado e vai para o **sétimo passo**. Senão, segue o fluxo.

Sexto passo: ainda existe tarefa no *sprint backlog*? Se sim, o *team* se ajudam para conseguirem concluir todas as tarefas do *sprint backlog* corrente e vai para o **quinto passo**. Senão, adianta alguma tarefa do *product backlog* e vai para o **quinto passo**.

Sétimo passo: o *scrum master* faz o *sprint review meeting* com o *product owner* e o *sprint retrospective meeting* com o *team*. Vai para o **segundo passo.**

Oitavo passo: **FIM!**

FIGURA 6: Os processos do SCRUM.
FONTE: (PIGOT, 2010).

3. ESTUDO DE CASO

3.1 OBJETIVO

Prover um processo de gerenciamento de atendimento de demandas próprias da área de manutenção de sistema baseado nos conceitos do *scrum*. O estudo de caso foi elaborado e aplicado por um recém-analista de sistemas, ou seja, uma pessoa que há pouco tempo tinha o papel de programador e passou a ser analista de sistemas. Ele tem o perfil de analista desenvolvedor. Em outras palavras, é um analista de sistemas que também faz o papel de programador. Esse analista deverá conseguir atender, executar, argumentar, comunicar e também prover demandas para com seus programadores de modo que evolua ou corrija os sistemas em que atua e altere a visão pessimista-negativa contida na área usuária.

3.2 CENÁRIO

A empresa produz e mantêm sistemas de gestão tributária, arrecadação e fiscalização para um órgão do estado. Essa empresa é composta por programadores, analistas de sistemas, administradores de banco de dados (DBA) e gerentes de projetos com papeis bem definidos segundo a semântica clássica. Os papeis são definidos da seguinte maneira:

- Programador desenvolve soluções sistemáticas especificadas pelo analista;
- Analista de sistemas analisa as demandas e provê soluções sistemáticas;
- DBA administra o banco de dados; e
- Gerente de projetos controla os projetos.

A empresa possui uma área de projeto responsável pela concepção do projeto até a entrega do produto final e uma área de manutenção, cuja responsabilidade é cuidar das manutenções no contexto geral de sistema. Essa área de manutenção possui diversos tipos de demandas, dentre elas:

- As corretivas: tem por intuito corrigir o produto;
- As evolutivas: são utilizadas para melhorar o produto;
- Do usuário: aquelas que são observadas/encontradas pelos usuários. Elas podem ser desde um problema encontrado no sistema, um cadastro de uma nova lei (utilização do próprio produto) até uma alteração no banco de dados. Geralmente são reportadas por meio de documento (eletrônico ou impresso), e-mail ou telefone.

Vale ressaltar que uma demanda do tipo do usuário pode virar uma demanda corretiva ou evolutiva, caso seja necessário, de acordo classificação do analista e seus gestores. Esses gestores são pessoas da área usuária selecionadas pelas coordenadorias do órgão que a empresa presta serviço, cuja responsabilidade é gerir, avaliar, autorizar ou priorizar o que deve ser feito no sistema. Cada sistema ou módulo pode ter mais de um gestor.

A empresa utiliza RedMine como ferramenta de gerenciamento de atividade e o EDOC como ferramenta de oficialização de solicitação de demandas pela área usuária. A depender da solicitação, ela percorrerá por diversas áreas até chegar de fato na área de TI, em especial, no setor de manutenção de sistema.

3.3 APLICAÇÃO

A aplicação foi iniciada montando um *dashboard* com as fases do ciclo de vida do atendimento de uma demanda. O nome das fases ou estado da demanda é criado de forma pessoal. No estudo de caso, o *dashboard* e as fases estão representados pela figura abaixo:

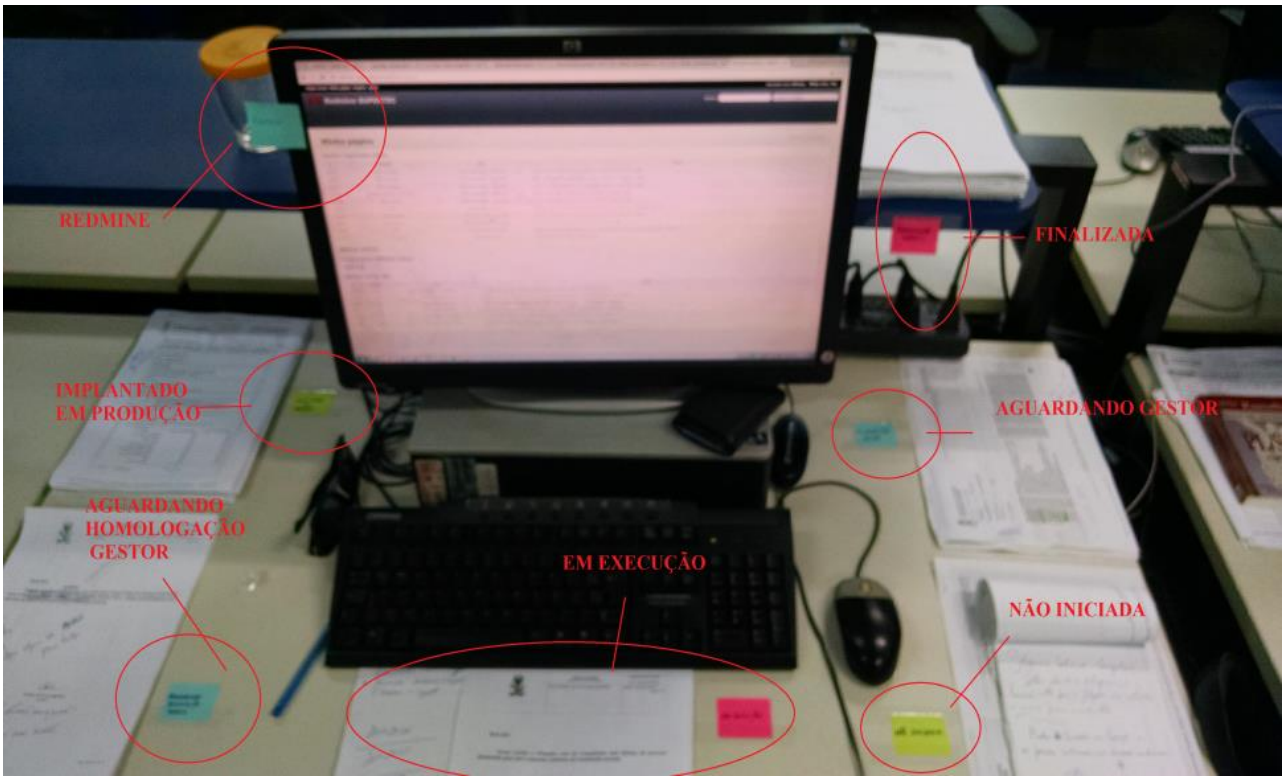

FIGURA 7: *Dashboard* e fases do ciclo de vida de uma demanda
Fonte: Elaborado pelo autor.

Em seguida, é preciso especificar claramente o significado de cada fase ou estado para que o analista de sistemas possa auto-gerenciar e expor aos seus gestores (*product-owners*):

- **Não iniciada:** Anomalia reportada.

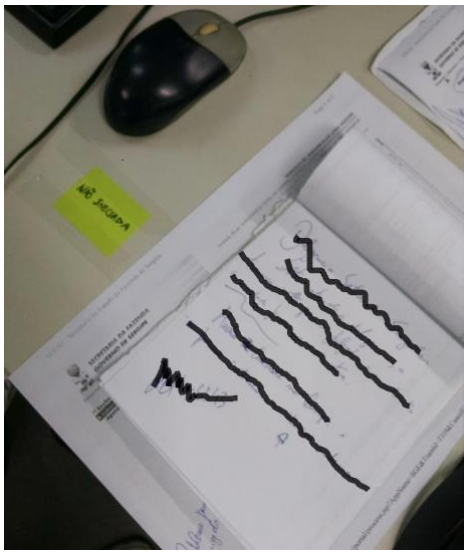

FIGURA 8: Fase - Não Iniciada
Fonte: Elaborado pelo autor.

- **Em execução:** Demanda em processo de análise ou sendo executada. Caso, ela esteja sendo executada, esta fase também abrange o primeiro nível de testes da implementação da demanda por parte do analista de sistemas.

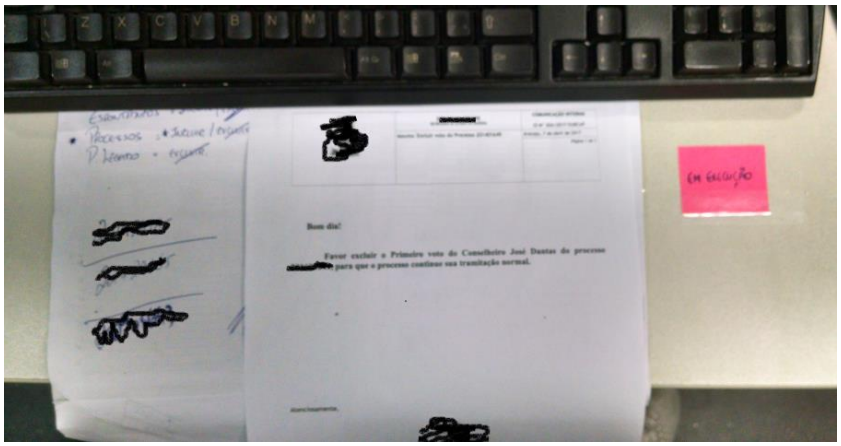

FIGURA 9: Fase – Em execução
Fonte: Elaborada pelo autor.

- ***Redmine*:** Sistema onde devem ser cadastrados as demandas e os artefatos gerados pela fase *em execução*.

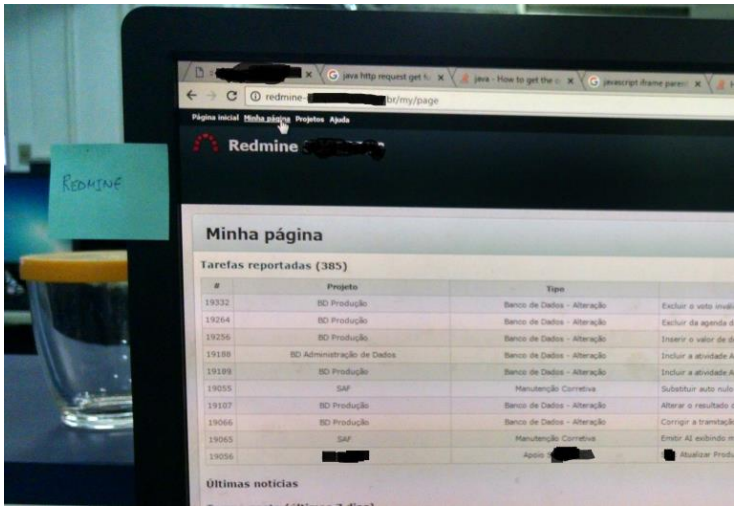

FIGURA 10: Redmine
Fonte: Elaborada pelo autor.

- **Aguardando Gestor**: Demandas que não foram iniciadas ou que estão na fase *em execução* que necessita do parecer do gestor.

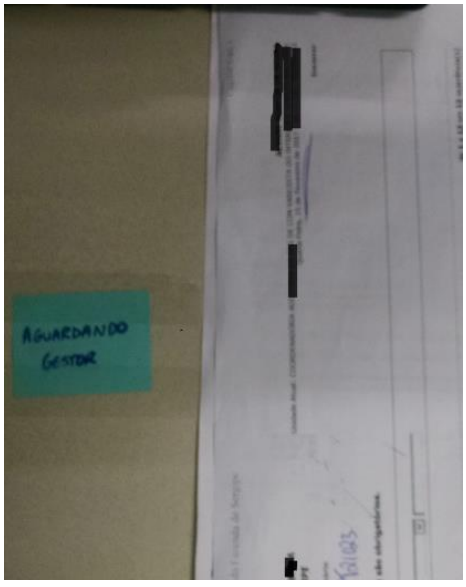

FIGURA 11: Fase – Aguardando gestor
Fonte: Elaborada pelo autor.

- **Aguardando homologação do gestor:** A demanda está aguardando a verificação e autorização do gestor para ser implantada em produção. A verificação é o processo que envolve o gestor checar se as demandas estão conforme o que foi solicitado.

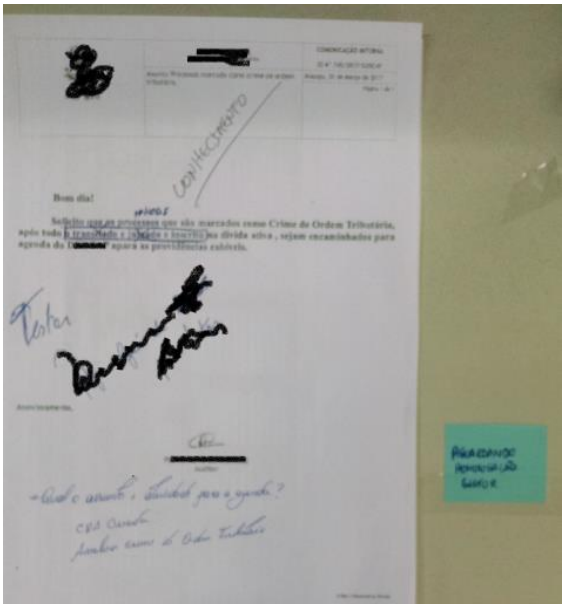

FIGURA 12: Fase – Aguardando homologação do gestor
Fonte: Elaborada pelo autor.

- **Implantado em produção:** Demanda que vai para produção. Ela permanece um mês em quarentena a fim de observar se houve alguma intercorrência ou a implantação ocorreu com sucesso.

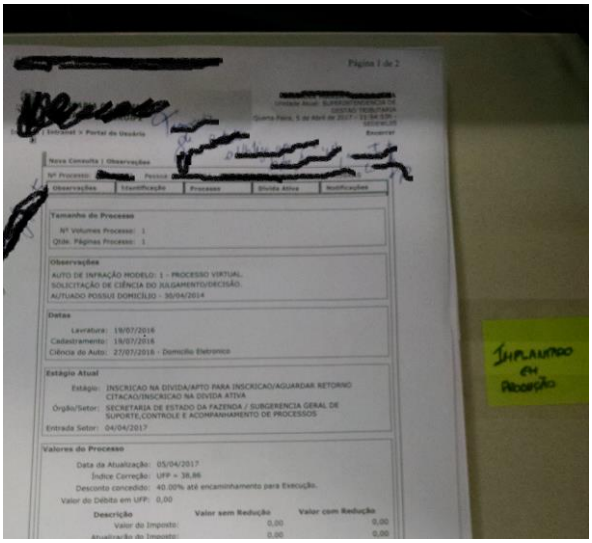

FIGURA 13: Fase – Implantada em produção
Fonte: Elaborada pelo autor.

- **Finalizada:** Demanda concluída.

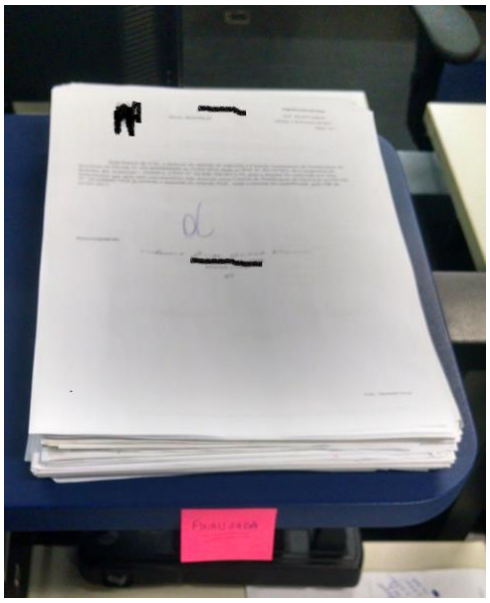

FIGURA 14: Fase- Finalizada
Fonte: Elaborada pelo autor.

O gerenciamento de demandas próprias de manutenção se inicia pela fase *não iniciada* com a interação entre área usuária e área de manutenção de sistemas, ou seja, uma anomalia do sistema é observada e reportada pelo usuário para o analista de sistemas. Ele, por sua vez, analisa a anomalia, verificando se ela é um erro de sistema ou um erro de utilização do usuário. Em seguida, o analista classifica a anomalia de acordo com tipo, urgência (quem reportou), a gravidade (impacto para o órgão e clientes deste órgão) e dificuldade para execução. Se ela for de caráter urgente e de gravidade alta, imediatamente ela é executada, senão é classificada segundo os tipos de demandas contida no cenário da empresa e colocado na fase de *aguardando gestor* para autorização. Vale ressaltar que demandas são colocadas na fase de *aguardando gestor* com intuito de elucidar questionamentos a cerca do negócio que as envolvem. Uma vez *em execução*, a demanda e seus testes são feitos. Após os testes pelo analista de sistemas, a demanda passará para fase de *aguardando homologação do gestor*. Nesta fase, o analista deverá informar sobre a mudança aos gestores para que eles possam homologar. Convencionou-se o encontro ou reunião como melhor forma de comunicação dos questionamentos e homologação das demandas, pois foi verificado que a comunicação por telefone e por meio de documento oficial, na maioria das vezes, gerava custo de tempo e esquecimento das partes. Para obter êxito com a comunicação convencionada, o analista deve observar a duração e a periodicidade de modo que elas não durem mais que 15 minutos e

sejam feitas diariamente, tornando objetivas e não cansativas. Passada a fase de homologação e autorizada à implantação, a demanda vai para fase de *implantada em produção* e o analista deve coloca-la no ambiente de produção. Após um mês sem que haja intercorrências a demanda vai para ultima fase que é *finalizada*.

3.4 RESULTADO

Observou-se uma melhora na organização e na produtividade das demandas do analista de sistemas, com o perfil de desenvolvedor, quando aplicado o modelo proposto dentre os que não possuíam processos para ajudar o gerenciamento do atendimento dessas demandas de manutenção. Checou-se o aumento da qualidade do produto, visto que as demandas finalizadas não foram mais citadas como anomalia de erro pela área usuária. Comprovou-se também o aumento da satisfação dos gestores e da equipe com o analista de sistemas da aplicação. Além disso, percebeu-se que os encontros diários influenciaram tanto na diminuição do gargalo entre fase de *aguardando homologação do gestor* e a de *implantado em produção,* quanto no aumento da colaboração entre o analista e seus gestores que desta forma observaram de perto o andamento das demandas de manutenção de seu modulo ou sistema. Houve também um compartilhamento de conhecimento entre as partes, analista, equipe e gestores, gerados pela confiança provida pelos encontros e pela qualidade das entregas das demandas finalizadas, passando assim a defender mais a equipe e a área de TI na área usuária. Isto é, a sementinha da mudança de uma imagem negativa estava plantada.

4. Conclusão

Conclui-se que o framework *scrum* é uma boa opção para o gerenciamento das próprias demandas de manutenção de sistemas. Sua adaptabilidade e facilidade de entendimento permitiram-nos utilizar seus conceitos com êxito como mostra a seção de resultados do estudo de caso. Verificou-se também que prazo de entrega, qualidade do produto, melhor aceitação do cliente e adaptação a mudanças são fatores importantes que podem ser atingidos com os conceitos das metodologias ágeis. Vale ressaltar que elas não resolvem todos os problemas, mas mostram uma maneira bastante interessante e iterativa de se planejar, executar, avaliar, checar e trabalhar com tarefas ou demandas, podendo inclusive contribuir para um ambiente de trabalho muito mais amigável, saudável e auto gerenciável.

REFERÊNCIAS

AGILE MANIFESTO, 2001, **Manifesto para desenvolvimento ágil de software.** Disponível em http://agilemanifesto.org/iso/ptbr/manifesto.html, acessado em 05/04/2017.

AUTOR DESCONHECIDO, 2009. **Métodos ágeis: Scrum**. Disponível em http://www.oficinadanet.com.br/artigo/1971/metodos_ageis_scrum, acessado em 05/04/2017.

AUTOR DESCONHECIDO, 2010. **O que é Scrum?.** Disponível em http://www.oficinadanet.com.br/artigo/gerencia/o_que_e_scrum, acessado em 05/04/2017.

BARBOSA, V; LIBARDI, O, L, P. **Métodos Ágeis**. UNICAMP – Faculdade de Tecnologia, 2010, São Paulo, 35p.

BEEDLE, M; SCHWABER, K, 2002.**Agile Software Development with Scrum**.Ed. PERSON EDUCATION, 2002, EUA.

COSTA, H, 2008. **Como anda o gerenciamento de projetos atualmente**. Disponível em: http://gerenciapratica.blogspot.com.br/2008/04/como-andam-os-projetos-atualmente.html, acessado em 05/04/2017.

D'ÁVILA, M, 2011. **Sucesso de projetos atualizado**. Disponível em: http://blog.mhavila.com.br/2011/06/18/sucesso-de-projetos-atualizado/, acessado em 05/04/2017.

FIRPO, F, 2011, **Causas de Fracasso de Projetos de Desenvolvimento de Software**. Disponível em http://analiserequisitos.blogspot.com.br/2011/06/causas-de-fracasso-de-projetos-de.html, acessado em 05/04/2017.

LARSSON, H, 2009. **Focus on the sprint backlog in the daily scrum**. Disponível em: http://henriklarsson.wordpress.com/2009/08/06/focus-on-the-sprint-backlog-in-the-daily-scrum/, acessado em 05/04/2017.

LEAL, B, 2011. **Ambiente ágil de desenvolvimento – SCRUM**. Disponível em http://brunableal.blogspot.com.br/2011/10/ambiente-agil-de-desenvolvimento-scrum.html, acessado em 05/04/2017.

MALMQUIST, P, 2011, **BurnDown Charts Makes Projects a Sport!** Disponívelem http://www.applitude.se/2011/02/burndown-chart-is-like-bowling-2/, acessadoem 05/04/2017.

NONAKA, I; TAKEUCHI, H. **Criação de conhecimento na empresa: como as empresas japonesas geram a dinâmica da inovação**. Rio de Janeiro: Campus, 358p, 1997.

PIGOT, T, 2010. **Scrum em moins de 10 minutes**. Disponível em: http://www.thierry-pigot.fr/gestion-de-projet/292/scrum-en-moins-de-10-minutes.html, acessado em 03/03/2017.

PRESSMAN, S, R, 1995. **Engenharia de Software**. São Paulo: MAKRON Books do Brasil, 1088p, 1995, 3ª Edição.

SCHWABER, K, 2004. **Agile Project Management with Scrum**.Ed. MICROSOFT PRESS, 2004, EUA.

WIKIPÉDIA, 2009. **Ciclo PDCA**. Disponível em http://pt.wikipedia.org/wiki/Ciclo_PDCA, acessado em 05/04/2017.

WIKIPÉDIA, 2012. **Rugby**. Disponível em http://pt.wikipedia.org/wiki/Rugby, acessado em 05/04/2017.

PAULA FILHO, W. P. **Engenharia de Software: fundamentos, métodos e padrões**. Rio de Janeiro: LCT- Livros Técnicos e Científicos, Editora SA, 2004. 602p. 2ª Edição.

FERNANDES, E; MEIRELLES, G; PICININI, M, 2015. **Análise das Práticas de Gerenciamento de Projetos e Desenvolvimento de Software**. Disponível em http://www.aedb.br/seget/arquivos/artigos16/17924164.pdf, acessado em 05/04/2017.

ABSTRACT

The software grew and became more complex and important for companies, so the software engineer creates new methodologies for software development to ensure quality and delivery. The Agile developments were borned to try to solve the problem. *Scrum* is the best know agile development of the world and offers a fast model of software development, able to modifications, adaptations and very effective, ensuring quality and delivery. This article presents introduce about the evolution of software engineer. Then, we are going to talk about agile manifest and the scrum agile development. After, we are going to apply case about demand management of system maintenance and finally the conclusions.

KEYWORDS: *Software Engineer. Framework Scrum. Agile Development.*