CENTRO UNIVERSITÁRIO LUTERANO DE PALMAS
Recredenciado pela Portaria Ministerial nº 1.162, de 13/10/16, D.O.U nº 198, de 14/10/2016
ASSOCIAÇÃO EDUCACIONAL LUTERANA DO BRASIL

Tayse Virgulino Ribeiro

SIDD – SCRUM ITERATION DRIVEN DEVELOPMENT: UM PROCESSO ÁGIL PARA DESENVOLVIMENTO E GERENCIAMENTO DE SOFTWARE BASEADO NO SCRUM

Palmas – TO

2018

Tayse Virgulino Ribeiro

SIDD – SCRUM ITERATION DRIVEN DEVELOPMENT: UM PROCESSO ÁGIL PARA DESENVOLVIMENTO E GERENCIAMENTO DE SOFTWARE BASEADO NO SCRUM

Projeto de Pesquisa elaborado e apresentado como requisito parcial para aprovação na disciplina de Trabalho de Conclusão de Curso II (TCC II) do curso de bacharel em Sistemas de Informação pelo Centro Universitário Luterano de Palmas (CEULP/ULBRA).

Orientador: Prof. M.Sc. Cristina D'Ornellas Filipakis Souza.

Palmas – TO

2018

Tayse Virgulino Ribeiro

SIDD – SCRUM ITERATION DRIVEN DEVELOPMENT: UM PROCESSO ÁGIL PARA DESENVOLVIMENTO E GERENCIAMENTO DE SOFTWARE BASEADO NO SCRUM

Projeto de Pesquisa elaborado e apresentado como requisito parcial para aprovação na disciplina de Trabalho de Conclusão de Curso II (TCC II) do curso de bacharel em Sistemas de Informação pelo Centro Universitário Luterano de Palmas (CEULP/ULBRA).

Orientador: Prof. M.e Cristina D'Ornellas Filipakis Souza.

Aprovado em: _____/_____/_____

BANCA EXAMINADORA

_____

Prof. M.Sc. Cristina D'Ornellas Filipakis Souza

Orientadora

Centro Universitário Luterano de Palmas – CEULP

_____

Prof. M.Sc. Fernando Luiz de Oliveira

Centro Universitário Luterano de Palmas

_____

Prof. Grad. Fábio Castro Araújo

Centro Universitário Luterano de Palmas

Palmas – TO

2018

AGRADECIMENTOS

Primeiramente a Deus por permitir que toda essa graça fosse realizada, por todas as dificuldades enfrentadas desde o início até minha conclusão, minha gratidão eterna a Deus. Ao Centro Universitário Luterano de Palmas por me acolher e colaborar para eu me tornar a pessoa e profissional capacitada que sou hoje.

Agradeço ao corpo docente que desde o início reconheceram a pessoa que sou e me acolheram não apenas como acadêmica e sim como uma amiga e até filha. Em especial minha orientadora, vulgo mãe, Prof. MSc. Cristina D'Ornellas Filipakis, não mediu esforços para me ajudar, apoiar, até mesmo diante de algumas tentativas de fraquezas me auxiliou e orientou a tornar deste Trabalho de Conclusão de Curso uma referência na área. Não poderia deixar de mencionar aos professores da minha banca, Prof. Grad. Fábio Castro Araújo e Prof. M.Sc. Fernando Luiz de Oliveira que durante o desenvolvimento deste me auxiliaram, sou grata a vocês.

Agradeço a todos os professores por me proporcionarem conhecimento para desenvolver este trabalho de conclusão de curso, não apenas o conhecimento científico e racional, mais também ética, caráter e obter uma ótima formação profissional, mas principalmente não por terem ensinado e aconselhado, mas por terem me feito realmente entender e aprender. Portanto meus mestres, esta conquista dedico também a vocês e terão meu eterno agradecimento, levarei cada conselho e aprendizado de vocês para toda vida.

Não poderia deixar de agradecer a minha família, em especial a pessoa que mais me apoiou nesta caminhada, minha mãe Rosimar Severo Virgulino, obrigada pelo amor e incentivo nos momentos mais difíceis durante esses anos da graduação, mesmo nas noites de estresses e correria. A minha irmã Tamirys Virgulino, pelo apoio, graças diárias e ajuda psicológica nas horas difíceis mesmo de longe. Enfim, aos meus avós, tios, primos e demais pessoas da família que acompanharam de perto minha evolução, sou grata por tudo.

Agradeço de modo especial a pessoa que se tornou um grande amigo de jornada, Matheus Leal. Me acompanhou durante toda etapa da graduação, e mais ainda na reta final, foi a pessoa que leu inúmeras vezes meus textos e me ajudou nas correções. Um grande amigo, que me apoio, me deu força, me proporcionou sorrisos, meu muito obrigada. A todos meus amigos que acompanharam de alguma forma, que de maneira direta ou indiretamente colaboraram com a minha formação em Bacharel em Sistemas de Informação, o meu muito obrigada.

RESUMO

RIBEIRO, Tayse Virgulino. **SIDD – Scrum Iteration Driven Development: Um processo Ágil para desenvolvimento e gerenciamento de *Software* Baseado no Scrum.** 2018. LVIII f. Trabalho de Conclusão de Curso (Graduação) – Curso de Sistemas de Informação, Centro Universitário Luterano de Palmas, Palmas/TO, 2018.

Este trabalho teve por objetivo o desenvolvimento de um processo ágil que permita o gerenciamento e desenvolvimento de *software* para empresas de desenvolvimento de *software*. No referencial teórico são apresentados alguns conceitos relacionados à Scrum, dando ênfase aos papéis e aos processos abordados para a utilização da prática ágil. Também são apresentados alguns estudos relacionados ao cenário de testes de verificação e validação, que auxiliam no processo de análise e acompanhamento de artefatos e código no desenvolvimento de um *software*. Levando em consideração estes conceitos, a metodologia do trabalho, de caráter quali-quantitativo, se baseou em um procedimento de estudo de caso descritivo e analítico sobre um processo de gerenciamento e desenvolvimento de *software* ágil. Para divulgar os resultados do trabalho, foi desenvolvida uma área de apresentação interativa do processo SIDD com informações relacionadas ao processo de gerenciamento e desenvolvimento de *software.*

PALAVRAS-CHAVE: Engenharia de *Software*, Metodologia Ágil, *Scrum*, Processo de Desenvolvimento, SIDD.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

SIDD – *Scrum Iteration Driven Development*

TDD – *Test Driven Development*

BDD – *Behavior Driven Development*

UML – *Unified Modeling Language*

HTML – *Hypertext Markup Language*

CSS – *Cascading Style Sheets*

SUMÁRIO

1 INTRODUÇÃO

A década de 1960 foi marcada por um período intitulado “crise do *software*”, uma expressão que passou a ser utilizada por representar problemas encontrados no processo de desenvolvimento de *software*. Estes problemas eram relacionados a falta de técnicas de desenvolvimento de *software,* prazo de projetos estourados e baixa qualidade do produto entregue. A partir deste período, alguns especialistas passaram a utilizar técnicas e ferramentas específicas para a otimização de processos de desenvolvimento de *software*. Fundamentado nisso, o aprimoramento e a criação de processos estruturados para o desenvolvimento de *software* passaram a contribuir com o processo de evolução da Engenharia de *Software*. Pressman (2011, p. 365) afirma que engenheiros de *software* devem se esforçar para produzir e utilizar técnicas e ferramentas para desenvolver sistemas de alta qualidade. Atualmente, a realidade para grandes e pequenas empresas é a dificuldade de compreensão dos conceitos de Engenharia de *Software* e há ausência ou até a má utilização de práticas de desenvolvimento de *software*.

Um processo ágil é caracterizado pela formação do potencial e necessidades dos envolvidos. Além disso, é caracterizado por ter como foco principal a implementação e aplicação de testes. É caracterizado por uma abordagem iterativa e incremental que, de maneira eficiente, divide em etapas o desenvolvimento para entregar o *software* funcional. Tem como principal objetivo entregar um produto com qualidade e que atenda às necessidades do cliente mediante o produto solicitado. Para desenvolver de maneira ágil é necessário que a equipe tenha como prática algumas características, como: competência, colaboração, tomada de decisão, respeito e principalmente organização. Essas características são consideradas extremamente importantes no ambiente de práticas ágeis, pois são facilitadores que auxiliam na utilização das práticas de desenvolvimento de software.

Portanto, este trabalho dá ênfase a uma metodologia de desenvolvimento de *software* ágil específica, denominada *Scrum*. De acordo com Sabbagh (2013), o *Scrum* é um *framework* ágil, simples e leve, utilizado para a gestão do desenvolvimento de produtos complexos. Identificá-lo como um *framework* significa dizer que ele é capaz de resolver um problema de um determinado domínio, e para resolver este problema, suas aplicações podem ser trabalhadas de acordo com os padrões do projeto. Além disso, o *Scrum* também segue uma abordagem iterativa e incremental para entregar valor com frequência e, assim, reduzir os riscos do projeto. O *Scrum* é baseado num conjunto de valores que trabalha para melhor

explorar o sucesso do projeto. De acordo com *Scrum* Alliance (2017) os cinco valores do *Scrum* são: foco, coragem, compreensão, comprometimento e respeito.

A partir disso, antes de iniciar uma implantação de uma prática/processo ágil de desenvolvimento de *software* numa empresa, é necessário realizar um estudo de caso a fim de compreender o ambiente e as pessoas envolvidas. A empresa deve realizar um diagnóstico no setor, para verificar se comporta o estilo de gerência exigido pelo processo e se possui indivíduos capacitados. Após a identificação destas características, todo o setor e a empresa devem se comprometer nos planejamentos que serão realizados, visto que isso é um influenciador significativo. De acordo com Awad (2005, p. 26) e Stoica, et. al (2013, p. 71), existe uma série de fatores influenciadores da metodologia ágil, como: abordagem da metodologia; maturidade da empresa/órgão e equipe; composição da equipe; nível de conhecimento do projeto; perspectiva de mudança do projeto, comunicação, cultura, documentação e o retorno de investimento do projeto.

A empresa deve criar uma cultura para que a equipe possua características de liderança e colaboração como pontos de partida para a utilização da metodologia de desenvolvimento. Logo, é necessário levar em consideração a experiência da empresa e que a equipe seja qualificada e disciplinada para utilização da prática de desenvolvimento. A composição destes fatores objetiva uma melhor abordagem de uma prática ágil de desenvolvimento de *software*.

Desse modo, o presente trabalho tem como objetivo criar um processo ágil, permitindo o desenvolvimento e gerenciamento de *software* de forma iterativa. Este processo tem como base uma pesquisa quali-quantitativa realizada com empresas palmenses em abril de 2017 por Ribeiro (2017), abordando os entrevistados quanto à utilização das práticas de desenvolvimento de *software*. Esta pesquisa objetivou identificar a abordagem da utilização e a evolução das empresas no contexto dos processos de desenvolvimento de *software*, para obtenção de um melhor entendimento da utilização de práticas ágeis. Portanto, propor um processo de gerenciamento e desenvolvimento de *software* com o intuito de proporcionar melhoria no processo de desenvolvimento de *software*.

Com base nisso, o problema que este trabalho procura resolver está relacionado a criação de um processo ágil para desenvolvimento e gerenciamento de *software* baseado no *Scrum* com o intuito de proporcionar melhoria no processo de desenvolvimento de *software* das empresas. Levantando como hipótese, se houver um entendimento da análise dos perfis das empresas e da abordagem de utilização do *Scrum*, então é possível criar um processo ágil para desenvolvimento e gerenciamento de software baseado no *Scrum* com o intuito de proporcionar melhoria no processo de desenvolvimento de *software*.

Portanto, além de explanar sobre os conceitos de metodologia ágil e as etapas do processo *Scrum*, este trabalho consiste em modelar um processo ágil fundamentado no *Scrum* em que permita o gerenciamento e desenvolvimento de *software*. Além disso, para a apresentação dos resultados, este trabalho também tem como objetivo desenvolver um site interativo para apresentação conceitual e gráfica do processo ágil SIDD.

Baseado nisso, o trabalho pretende evidenciar as melhores práticas para uma abordagem de desenvolvimento de *software* ágil, como o *Scrum*, além de estudos que mostrem a ausência do conhecimento para adesão e melhor utilização desta prática.

Com base na pesquisa realizada por Ribeiro (2017), abordando empresas de Palmas-TO quanto à utilização das práticas de desenvolvimento de *software*, foram envolvidas vinte empresas/órgão. Dentre essas empresas, sete não adotavam uma prática de desenvolvimento específica de *software*, duas adotavam práticas tradicionais de desenvolvimento, enquanto onze empresas adotavam práticas ágeis. Além disso, 100% das empresas que utilizavam práticas ágeis utilizavam como base o processo de desenvolvimento *Scrum*.

Foi possível observar que as empresas que trabalhavam com a prática de desenvolvimento ágil não necessariamente atendem a todos os princípios de uma metodologia ágil, pois grande parte de sua utilização é feita de maneira parcial. Dessa maneira, a proposta de um novo processo ágil de desenvolvimento de software capaz de direcionar as atividades de desenvolvimento e gerenciamento, além de levar em consideração o contexto da empresa, pode contribuir para que as empresas sigam os princípios desse processo em sua totalidade, sem a necessidade de técnicas complementares.

Por fim, este trabalho é estruturado da seguinte forma: seção 1, Introdução, apresenta o problema, hipótese, objetivos e justificativa que alusivas o trabalho; seção 2, Referencial Teórico, onde são apresentados os conceitos relacionados a metodologias ágeis, testes de verificação, testes de validação, metodologia ágil BDD e *Scrum*; seção 3, Materiais e Métodos, fornece uma breve introdução aos materiais e tecnologias utilizadas, apresenta e explica a metodologia escolhida para o desenvolvimento do trabalho, que consiste numa abordagem quali-quantitativa; seção 4, Resultados, apresenta o que obtido de resultado na execução do trabalho; seção 5, Considerações Finais, apresenta o que foi concluído na execução deste trabalho e, posteriormente, são listadas as referências que serviram como base para a construção do trabalho.

2 REFERENCIAL TEÓRICO

Esta seção irá abordar os temas Metodologias Ágeis, Testes de Verificação, Teste de Validação, BDD e *Scrum*, podendo assim ter uma visão conceitual e detalhada das técnicas que influenciam no desenvolvimento ágil.

2.1 METODOLOGIAS ÁGEIS

Após a "crise do *software"*, em meados dos anos 90, as metodologias ágeis passaram a ser conhecidas como métodos de melhoria de processos de desenvolvimento. No ano de 2001, foi escrito um Manifesto Ágil por Kent Beck e outros dezesseis profissionais de *softwares*. Esse manifesto gerou doze princípios que explicam e complementam o conceito agilidade na área de desenvolvimento de *software*. A partir desse ano, o termo "Metodologias Ágeis" tornou-se popular.

Este manifestado assinado em 2001 corresponde a um conjunto de valores e práticas para o desenvolvimento de *software* ágil. Pressman (2011) afirma que a engenharia de *software* combina uma filosofia com o conjunto dos princípios ágeis.

> "A filosofia defende a satisfação do cliente e a entrega de incremental prévio; equipes de projetos pequenas e altamente motivadas; métodos informais; artefatos de engenharia de *software* mínimos e, acima de tudo, simplicidade no desenvolvimento geral. Os princípios de desenvolvimento priorizam a entrega mais que a análise e projeto (embora essas atividades não sejam desencorajadas); também priorizam a comunicação ativa e contínua entre desenvolvedores e clientes" (PRESSMAN, 2011, p. 81).

É importante compreender os conceitos do manifesto ágil, que objetivam valores e práticas de melhorias de desenvolvimento de *software*. Segundo o *Agile Alliance* (2001) os princípios consistem em:

1. **Satisfação do cliente por meio de entrega antecipada do *software*:** o foco deste princípio está em gerar retorno para o cliente, ou seja, satisfazê-lo. A fim de constantemente proporcionar partes entregáveis do produto.
2. **Promover um bom atendimento ao cliente diante os pedidos de alterações, pois para os processos ágeis é um meio vantajoso na competitividade da relação com o cliente:** o intuito de adotar um processo de desenvolvimento de *software* ágil é trabalhar com s*prints*, permitindo um curto fluxo de trabalho e retorno frequente. Assim é possível o acompanhamento da evolução do produto por parte do cliente. Esse princípio objetiva

tornar o desenvolvimento e gerenciamento do produto mais previsível, ou seja, ser adaptável a mudanças.

3. **Entregar o software em constante funcionamento de maneira frequente:** a entrega constante do produto permite obter-se *feedback* sobre o que foi produzido. Este princípio objetiva incentivar diversas entregas com o intuito de reduzir os riscos no decorrer do desenvolvimento do produto.
4. **Existir trabalho em grupo, independente do setor:** é necessário que independente do setor as partes interessadas tenham como objetivo gerar valores. Para atingir esse objetivo é necessário que ambas as partes contribuam para o gerenciamento e desenvolvimento do produto.
5. **Construir projetos em torno de indivíduos motivados:** para construir projetos em torno de indivíduos motivados é necessário fornecer ambiente e suporte para a realização do trabalho. O uso e disponibilização correta das ferramentas como instrumentos é essencial para o bom desempenho de um projeto composto por responsáveis motivados.
6. **Método de conversa aberto e presencial:** é necessário comunicar-se para haver uma eficiência na informação transmitida. A eficiência no modelo de comunicação é necessária para o bom desenvolvimento entre as partes interessadas, como para sincronia e padronização da comunicação no ambiente de desenvolvimento dos projetos.
7. **Software em funcionamento é a principal medida de progresso:** o progresso do projeto é equivalente as entregas frequentes. Estas entregas podem ser caracterizadas para o cliente como valor ou não. Logo, sendo artefato ou documentação, o que foi executado é considerado como progresso.
8. **Processos ágeis promovem um desenvolvimento sustentável, os indivíduos devem estar capacitados para manter um ritmo constante:** para o ritmo da equipe se manter sempre constante é necessário entender o nível máximo de produtividade da equipe. E um bom ritmo de trabalho representa compromisso no prazo de entrega e qualidade no desenvolvimento do produto. Este nível de produtividade irá influenciar diretamente na insatisfação ou satisfação dos membros do time, logo, a uma maior ou menor produtividade e qualidade na execução do produto.
9. **Atenção contínua para com a excelência técnica:** este princípio objetiva um melhor gerenciamento e desenvolvimento do produto por meio de um trabalho ágil, com qualidade, atenção e excelência em técnica.
10. **Simplicidade no desenvolvimento:** simplicidade também é qualidade. Desenvolver de maneira simples é ser sustentável em código, é ser flexível e é ser rápido.

11. **Arquiteturas planejadas e organização:** esse princípio reflete diretamente na auto-organização dos membros e da equipe por um todo, que tem por objetivo sustentar todo o desenvolvimento e gerenciamento do produto. A autonomia proporciona eficiência, equipes eficientes trabalham de modo organizado e responsável, buscando sempre apresentar o melhor do seu trabalho.
12. **Autoavaliação do trabalho para manter uma melhor sintonia:** A autoavaliação da equipe objetiva inspecionar a abordagem de trabalho, identificando pontos positivos e negativos, em busca de uma melhoria contínua. É uma autoavaliação do processo de desenvolvimento e gerenciamento de *software* utilizado pela equipe. Esse princípio proporciona inspecionar de maneira constante a eficácia da equipe mediante o processo de desenvolvimento adotado.

Os dozes princípios ágeis baseiam-se em quatro valores muito importantes para o desenvolvimento de *software* ágil. Libardi e Barbosa (2010, p. 4) explicam que estes valores consistem em:

1. Indivíduos e interação entre eles mais que processos e ferramentas: os *softwares* são construídos por uma equipe, com o intuito de trabalhar juntas. Processos e ferramentas são importantes no desenvolvimento, mas não é mais importante do que trabalhar em conjunto.
2. *Software* em funcionamento mais que a documentação: a documentação é um recurso de auxílio a equipe de desenvolvimento, mas entender o *software* em funcionamento contribui para uma melhor compreensão do seu uso.
3. Colaboração com o cliente mais que a negociação de contratos: o cliente é uma das partes interessadas, ele é o especialista do domínio e tem suas expectativas sobre o que espera do *software.* O termo de contrato existe para certificar as responsabilidades, mas não deve substituir uma boa comunicação.
4. Responder a mudanças mais que seguir planos: mudanças constituem uma realidade natural e podem acontecer inúmeras vezes, pois as pessoas mudam de ideia, as ferramentas e processos evoluem. E mesmo que mudanças estejam na realidade do ambiente de desenvolvimento, é necessário que a execução de um produto tenha um planejamento.

Estes conceitos são considerados como preferências para a utilização de um método ágil e não apenas como uma alternativa de abordagem de desenvolvimento (Agile Alliance, 2001).

A adesão de uma metodologia ágil consiste na sua importância de planejamento para a criação de um *software*. Na atualidade, a diversidade de metodologias ágeis de gerenciamento e desenvolvimento de *software* é considerada grande, algumas destas metodologias são: *Agile Modeling, Agile Unified Process, Behavior Driven Development, Crystal, Dynamic Systems Development Methodology, Extreme Programming, Feature Driven Development, Lean Software Development, Microsoft Solutions Framework, Rapid Application Development, Rational Unified Process, Scrum e Test Driven Development*.

Apesar das diferenças entre as metodologias ágeis, todas trabalham com a possibilidade de um *software* possuir qualidade, agilidade no processo de desenvolvimento, leveza e satisfação do cliente. Um dos fatores mais importantes da adoção do processo ágil é o fator humano no que diz respeito às suas habilidades. Para uma equipe ser considerada ágil ela precisa obter características como competência, colaboração, tomada de decisão, confiança, respeito e principalmente organização. “Essas características são consideradas importantes no processo ágil, pois ele se mantém nas habilidades citadas, que foram criados para facilitar e dinamizar a produção de uma equipe/empresa no momento de desenvolver um *software*” (PRESSMAN, 2011, p. 86).

As metodologias ágeis são a composição de processos de desenvolvimento e gerenciamento de software que incentivam um trabalho orientado a inspeção e adaptação. Elas têm como objetivo principal entregar e agregar valor ao produto para o cliente. Esses métodos são caracterizados por proporcionar entregas frequentes e funcionais, considerados, portanto métodos ágeis. Na perspectiva ágil, é recomendado o uso de técnicas de testes, a aplicação dessas técnicas é uma tarefa que trabalha de modo paralelo com o processo de desenvolvimento do *software*. Portanto, os métodos ágeis dispõem-se de conjuntos de técnicas ágeis, como por exemplo, *Test Driven Development* e *Behavior Driven Development*.

Os testes são atividades que são planejadas com antecedência e executadas de forma disciplinada. Baseado nisso, para mensurar o nível de qualidade de um *software* é necessária aplicação de elementos específicos. Esses elementos podem ser caracterizados como funcionalidade, confiabilidade, usabilidade, eficiência, portabilidade, entre outros. As aplicações desses atributos de qualidade estão envolvidas desde a etapa de concepção do projeto até a execução do teste. Diante disso, o principal objetivo dos testes é revelar falhas para que sejam corrigidas e fornecer informações sobre sua qualidade em relação a funcionalidade que será executada.

2.2 TESTES DE VERIFICAÇÃO

Uma das bases dos testes de *softwares* são os testes de verificação, esses testes iniciam-se após a disponibilização dos artefatos. De acordo com Hass (2008, p. 18), os Testes de Verificação devem ser capazes de responder o seguinte questionamento: “Está sendo construído o produto corretamente?”. Os Testes de Verificação devem ser sistematizados, planejados e devidamente aplicados. Conforme é apresentado por Bartié (2002, p. 67), os testes de verificação objetivam avaliar se a documentação gerada e todas as atividades estão sendo desempenhadas de maneira adequada, a fim de gerar um modelo de *software* adequado às necessidades do cliente.

Em conformidade com Lawanna (2012, p. 36), no teste de verificação deve-se incluir revisões técnicas, inspeções, orientações e esforços de *software*. Para que seja possível verificar requisitos, componentes, realizar testes e auditoria no produto. Bartié (2002, p. 69) afirma que, para que seja possível proporcionar melhoria na eficiência do processo de verificação, é necessário obedecer a algumas regras. Dessa forma, os testes de verificação passaram a obter maior eficiência no processo de detecção de erros de um *software*. As regras estão distribuídas em oito etapas, que são:

- Profundidade das Análises e Discussões: nesta etapa deve ser analisado e revisado cada elemento do produto. Deve-se analisar a coerência dos elementos, se suas estruturas estão bem flexíveis e claras. Como também, avaliar meios em que a entrega do produto seja alcançada de maneira mais simples e fácil.
- Uniformidade das Atividades: para se obter sucesso na execução das atividades é necessário treinamento adequado do revisor para que seja capaz de estruturar de maneira correta seus trabalhos.
- Continuidade e Frequência: as atividades do teste de verificação devem ser executadas ao fim de cada mudança, de modo a manter sua totalidade na documentação e garantir que as decisões tomadas sejam devidamente analisadas e compartilhadas pelas partes interessadas.
- Revisores Experientes: os revisores devem ser profissionais de experiência, e que tenham domínio das atividades trabalhadas.
- Presença de um Moderador nas Reuniões: a principal função de um moderador é garantir com que os participantes estejam recebendo devida atenção. Além disso, tem como missão manter foco nos assuntos abordados durante a reunião.

- Revisões Curtas e Bem Focadas: as reuniões serão mais efetivas se forem curtas e bem direcionadas. Quanto mais a reunião se prolongar, maior será a dispersão dos revisores, uma vez que mais assuntos serão abordados pelos revisores.
- Identificar Problemas, não os Resolver: para evitar que falhas ocorram é necessário que nas reuniões não sejam discutidos como será solucionado o problema. Nesta etapa é necessário que a equipe identifique e relate os problemas. Após isto, deve-se estabelecer o responsável para estudá-lo e corrigi-lo.
- Concluir as Revisões: é necessário haver um relatório que apresenta a conclusão das verificações, este relatório deve conter de modo detalhado os motivos da mudança de decisões e definições da documentação. O relatório será um instrumento de auxílio para identificar as revisões para futuras análises.

O cumprimento correto das etapas irá proporcionar uma maior eficiência no processo de verificação no desenvolvimento de *software*.

2.2.1 Papéis existentes em uma Reunião de Verificação

Para a realização das reuniões de verificação alguns papéis são definidos para a distribuição correta das atividades. É necessário que cada profissional desempenhe de maneira satisfatória o seu papel. Segundo Bartié (2002, p. 72) os papéis existentes numa reunião de verificação são:

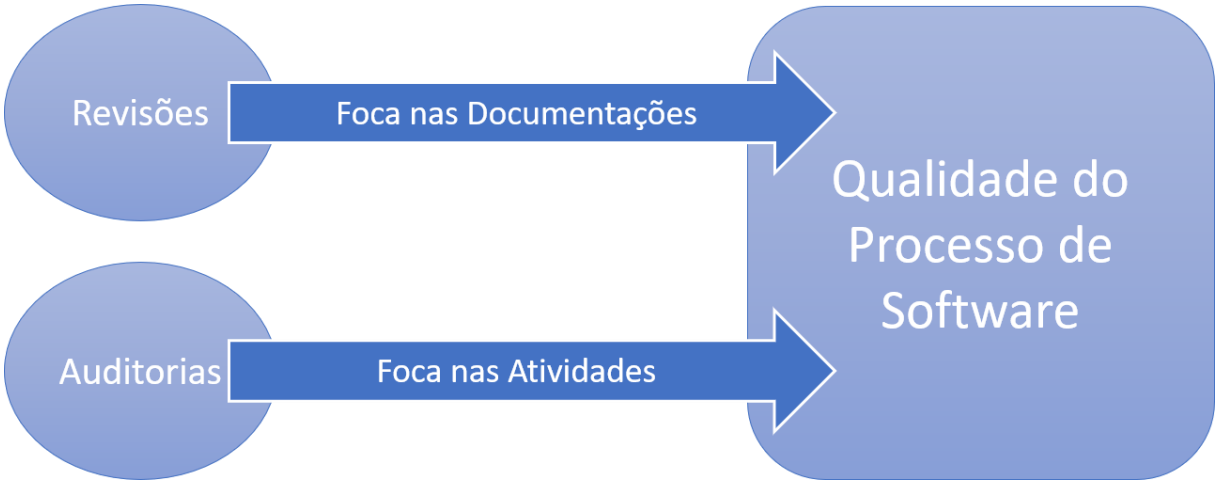
- Moderador: o principal papel do moderador é fazer com que seja cumprida a agenda estabelecida para a reunião. Além disso, deve manter a reunião focada, evitando dispersões durante a reunião.
- Escrivão: tem como principal atividade manter registrado os assuntos abordados e ações propostas para futura execução. O escrivão será exclusivamente a “memória” da reunião, ou seja, tudo será documentado por ele.
- Autor: “o autor é o criador do documento a ser criado. O autor deve explicar o documento e fornecer informações fundamentais à compreensão do conteúdo do material” (BARTIÉ, 2002, p. 73). O autor será responsável por acompanhar o início do processo e fazendo curtas participações nas reuniões, quando solicitado.
- Revisor: é o responsável que terá como função exclusivamente a revisão do documento e identificação de problemas. Deve trabalhar unicamente para obter melhoria do documento.

Preferencialmente, os responsáveis com as atribuições das atividades devem ser profissionais da área de qualidade.

2.2.2 Métodos Estruturados de Verificação

Os testes de verificação concentram-se em dois aspectos: atividades que trabalham com a documentação, chamadas de revisões, e as que avaliam as atividades, chamadas de auditorias. Essas atividades são conhecidas como métodos estruturados de verificação, conforme apresentado na Figura 1.

Figura 1 - Métodos estruturados de verificação



Fonte: Adaptado de Bartié, 2002, p.74

As **revisões** concentram-se em atividades direcionadas a análise de documentos. As execuções dessas atividades são caracterizadas por algumas fases, que são: na fase inicial da concepção do documento é aplicada a **revisão isolada**. Na fase da validação é aplicada a **revisão formal**. E por fim, na fase de divulgação, são aplicadas as **reuniões de acompanhamento**. Na Figura 2 são apresentadas as fases de criação dos documentos e revisões.

Figura 2 - Fases de criação de documentos e suas Revisões

Na fase de criação, a **revisão isolada** trata-se de uma verificação individual por um responsável que não seja o autor do documento. Essa etapa executa algumas atividades que visam revisar e identificar a maior quantidade de problemas possíveis. As principais atividades citadas por Bartié (2002, pp. 75-76) são:

- Detecção prematura de erros
- Objetiva a maior detecção de possíveis problemas
- Antecipa a revisão dos documentos gerados
- Acontece na fase de concepção do documento
- Não deve acontecer de forma unilateral
- Não deve ser acontecer revisões superficiais
- Caso ocorra desacordos, uma reunião para análise deve ser proposta

Na fase de validação, a **revisão formal** trata-se de reuniões realizadas com grupos de especialistas responsáveis por identificar falhas ocorrentes em documentos que são gerados nas etapas de desenvolvimento. São apresentadas por Bartié (2002, pp. 76-77) algumas características que descrevem a etapa de revisão formal, que são:

- O autor do documento não deve conduzir a reunião
- Os membros da equipe devem ter conhecimento do domínio, e além disso, visões diferentes
- Deve ser realizada toda a documentação do processo
- Deve ser apresentado problemas detectados e propostas de correções
- Realização da revisão somente no que foi modificado

Na fase de divulgação, as **reuniões de acompanhamento** têm por objetivo divulgar o que está sendo desenvolvido, além de familiarizar os revisores participantes das reuniões. Essa forma de verificação aborda algumas principais características apresentadas por Bartié (2002 pp. 77-78), que são:

- É uma verificação menos formal que as reuniões de revisão
- O autor do documento é o único que se prepara para a reunião, com o intuito de direcionar a reunião
- Todos participantes da etapa de desenvolvimento devem estar presentes
- No final da reunião, o que foi discutido na reunião deve ser acordado por todos por meio da assinatura de um documento apresentado

Para a realização da atividade de revisão é necessária a organização da execução das etapas, pois cada fase contém sua própria característica e particularidade.

A **auditoria** é um método de verificação baseado nas atividades de um processo de engenharia de *software*. Esse método é guiado exclusivamente por auditores de qualidade e tem como objetivo avaliar se a equipe está respeitando o processo de desenvolvimento. Este processo consiste na realização das reuniões de revisão, concepção da documentação, feedback do andamento do processo e verificação dos riscos no projeto.

De acordo com Bartié (2002, p. 79), é necessária a efetivação de algumas boas práticas para o processo de verificação. Esse conjunto de boas práticas é composto por:

- Planejamento das reuniões de verificação: tem por objetivo determinar o alvo do trabalho e formação do grupo que participará das atividades que serão realizadas. Além disso, estabelecer características de complexidade, importância e impacto da documentação.
- Preparando os profissionais: nessa atividade é necessário que todos os participantes tenham conhecimento sobre o que será apresentado na reunião, cada participante deve contribuir para agregar conhecimento com os outros membros que estão participando da reunião.
- Executar a verificação de documentos: o autor do documento deverá garantir que todos os pontos da documentação foram questionados e avaliados pelos revisores. Para a condução desse processo algumas atividades deverão ser executadas, que são:
  - ✓ Definição de um tópico
  - ✓ Levantamento e discussão de uma questão
  - ✓ Apresentação da existência de erros por revisores
  - ✓ Detalhamento e registro do defeito
  - ✓ Levantamento de outras questões, até que todas tenham sido analisadas
  - ✓ Uma nova definição de tópico, até a discussão de todos os identificados
- Aplicação do C*hecklist*: é um instrumento que direciona a verificação, caracterizado com um guia. Cada etapa de desenvolvimento deve conter um *checklist*. Com isso, cada *checklist* deve conter pontos específicos para verificação, a fim de avaliar o que está sendo desenvolvido. Na Figura 3 é apresentado um exemplo de *checklist*.

Figura 3 - Exemplo de *checklist*

Checklist de Casos de Uso	
As especificações seguem os fluxos básicos, alternativos e as exceções?	**SIM ( ) NÃO ( )**
A nomenclatura dos casos de uso seguem um padrão geral na documentação?	**SIM ( ) NÃO ( )**
As especificações atingem um nível básico necessário de detalhamento do requisito?	**SIM ( ) NÃO ( )**
As especificações das exceções estão coerentes e estão sendo aplicadas aos resultados corretamente?	**SIM ( ) NÃO ( )**
Cada caso de uso está descrito com clareza, concisão e sem ambiguidade?	**SIM ( ) NÃO ( )**
Os casos de uso descrevem as respostas do sistema ao usuário devido às condições de erro?	**SIM ( ) NÃO ( )**
O caso de uso contempla um comportamento supérfluo?	**SIM ( ) NÃO ( )**
Os casos de uso contribuem coletivamente para todos os comportamentos necessários do sistema?	**SIM ( ) NÃO ( )**
As exceções estão gerando respostas coerentes a serem aplicadas pelo sistema?	**SIM ( ) NÃO ( )**
Cada caso de uso está associado à, pelo menos, um ator?	**SIM ( ) NÃO ( )**
Existem casos de usos conflitantes com outros casos de usos ?	**SIM ( ) NÃO ( )**

Cada fase do processo de verificação é importante para compreensão do problema e definição da solução. Baseado nisso, Bartié (2002, p. 83) afirma que pelo fato da etapa de verificação se basear nas etapas iniciais do projeto, ela torna-se fundamental no processo de qualidade da concepção de desenvolvimento de um *software*.

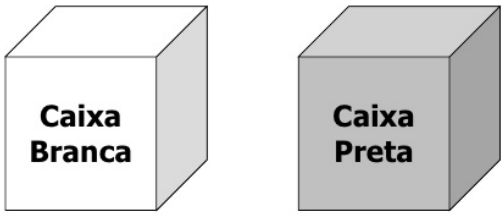
2.3 TESTES DE VALIDAÇÃO

Os Testes de Validação iniciam-se após a montagem individual de todos os componentes do *software* e da correção de seus erros. Segundo Bartié (2002, p. 103), os testes de validação objetivam identificar o maior número de erros possíveis nos componentes do *software*. Além disso, o teste de validação baseia-se no planejamento dos testes e nos requisitos mais críticos que podem contribuir no aumento da eficiência de detecção de defeitos.

Os Testes de Validação possuem duas abordagens de processo de validação, que são: estratégia caixa branca e estratégia caixa preta. Essas estratégias determinam o problema e o

procedimento para a execução dos testes. Na Figura 4 será apresentada as duas abordagens estratégias do teste de validação.

Figura 4 - Estratégias fundamentais dos testes

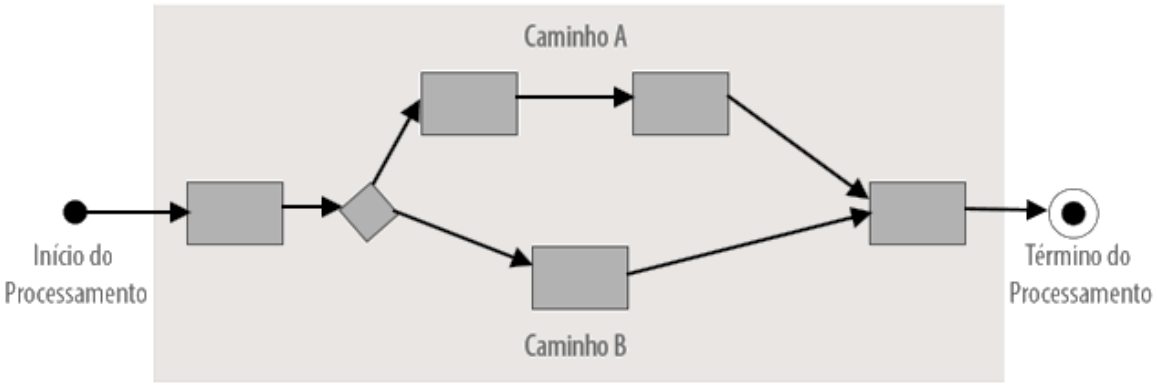


Fonte: Adaptado de Bartié, 2002, p.104

As estratégias apresentadas na figura acima são complementares, ou seja, a aplicação de testes de caixa branca e preta significa a possibilidade de obtenção maior qualidade no processo.

De acordo com Bartié (2002, p. 104), os testes de caixa branca são baseados na arquitetura interna do *software*. Afirma também que esses testes objetivam identificar defeitos nas estruturas internas dos programas por meio de simulações de performance no código. A execução do teste de caixa branca deve ser realizada por profissionais de teste que tenham conhecimento relacionado a tecnologia utilizada no *software*. Além disso, conter conhecimento sobre toda estrutura interna a fim de realizar o processo de validação nos devidos componentes do *software*, conforme ilustrado na Figura 5.

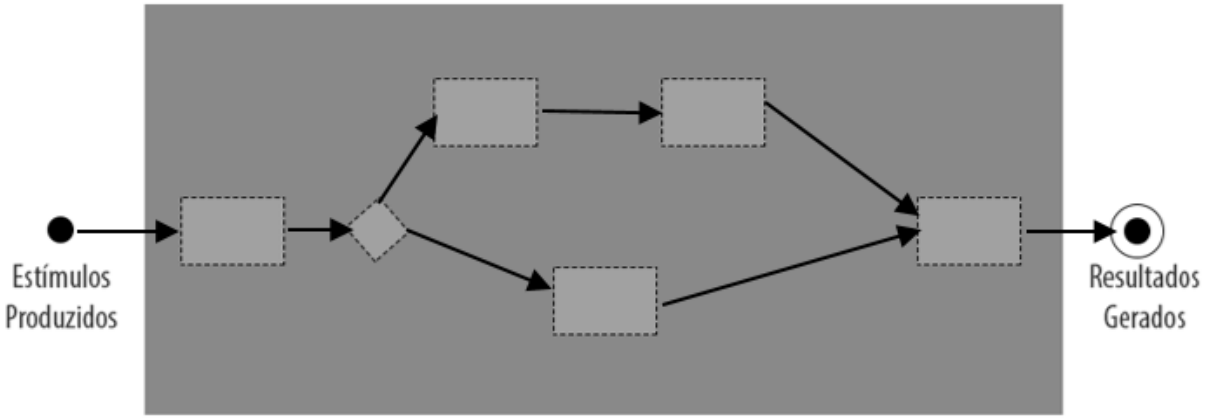
Figura 5 - Visão de testes de caixa branca



Fonte: Adaptado de Bartié, 2002, p.104

A estratégia de caixa preta tem por objetivo promover que os requisitos do sistema sejam integralmente atendidos pelo *software* construído. Bartié (2002, p. 105) aponta que a grande vantagem do teste de caixa preta é o fato de não requisitar conhecimento da tecnologia empregada, que por fim reduz a dificuldade de encontrar profissional capacitado a modelar os testes da aplicação. Na figura 6 é apresentado o fluxo dos testes de caixa preta.

Figura 6 - Visão de testes de caixa preta



Fonte: Adaptado de Bartié, 2002, p.105

"Os testes de caixa preta são conhecidos por serem mais simples de se implementar do que os testes de caixa branca" (BARTIÉ, 2002, p. 106). Ainda assim, Bartié aponta que ambos podem ser complexos, pois exigem esforço para planejar e automatizar os procedimentos.

Para a realização de um processo de desenvolvimento de *software* é necessário um modelo de aplicação eficiente, a fim de suportar mudanças eventuais depois da concepção da aplicação. Por meio disso, é possível a aplicação dos testes progressivos e regressivos. O teste progressivo atua na etapa de construção do produto, ou seja, a elaboração do teste é de acordo com o progresso de desenvolvimento do produto. Porém, o teste regressivo trabalha com a ideia de assegurar que modificações no decorrer do processo de desenvolvimento não afetem outras partes do produto. Assim, para cada nova versão disponibilizada, um novo teste deverá ser realizado.

De acordo com Bartié (2002, p. 121), os métodos de validação baseiam-se em testes dinâmicos com o intuito de avaliar se o comportamento do *software* está de acordo com o esperado. Dessa forma, os métodos de validação visam identificar todos os cenários possíveis de testes. Assim, para a obtenção dos cenários de testes, são construídos os chamados casos de testes.

A técnica de casos de testes está associada ao monitoramento da qualidade do *software*, avaliando histórico de evolução da aplicação dos testes. Para entender a técnica de casos de testes, será apresentada uma aplicação no seguinte cenário: como visitante do sistema eu quero criar uma conta de usuário no sistema. Abaixo segue a criação do caso de teste do cenário CT001.

Cenário de teste

1. Fluxos
   a) Abrir a tela de cadastro de usuários. [1]
   b) Informar os dados do campo do formulário de usuário. [2]
   c) Clicar em ‘Cadastrar’. [2]
2. Cenários
   a) CT001: Validação dos campos informados no formulário, verificar se os campos foram preenchidos correspondente (Campos obrigatórios, campos válidos) [1];
   b) CT001.1: Validação do CPF do usuário[1];
   c) CT002: Usuário já cadastrado; [2]
   d) CT003: Cancelar cadastro de usuário. [2]
   e) CT004: Cadastrado realizado com sucesso. [3]

Casos de teste

Caso de teste	CT001 - Validação de formulário
Pré-condições	Campos não preenchidos.
Procedimento	1. O usuário escolhe a opção de cadastrar usuário. 2. O sistema retorna um formulário com os campos necessários para cadastrar um novo usuário. 3. O usuário informa os dados solicitados. 4. O sistema retorna os dados informados pelo usuário e apresenta uma mensagem de acordo com o avaliado. 5. O sistema exibe uma mensagem que o formulário não foi preenchido corretamente.
Resultado esperado	Retorna o formulário com os campos válidos para o cadastro de um novo usuário.
Dados de entrada	Um acesso não existente, informações válidas.
Ambiente de execução	• IE - Internet Explorer • Safari • Mozilla Firefox • Google Chrome
Prioridade	Alta.

De acordo com a aplicação da técnica de testes apresentada acima, é possível identificar a criação de caso de testes de um dos cenários aplicados. Portanto, quanto mais o

teste garantir conformidade, maior a tendência do *software* ser de qualidade. Mas para isso, é necessário simular o maior número de cenários possíveis de um teste. Desse modo, Bartié (2002, p. 104) afirma que para a obtenção de resultados efetivos no processo de validação do *software* é necessário um planejamento de todas as atividades que serão realizadas.

2.4 BEHAVIOR DRIVEN DEVELOPEMENT (BDD)

O *Behavior Driven Development* (BDD) é um processo de desenvolvimento de *software* orientado a comportamento que se originou em resposta ao *Test Driven Development* (TDD). O *Test Driven Development* é caracterizado por desenvolver orientado a testes, ou seja, o teste é planejado e escrito antes da etapa de codificação. Dan North foi o responsável pela criação do TDD e sua evolução para o BDD no ano de 2003. De acordo com North (2006), "Ele evoluiu a partir de práticas ágeis estabelecidas e foi projetado para torná-los mais acessíveis e eficazes para equipes novas para entrega ágil de *software*. O BDD cresceu para abranger a imagem mais ampla de análise ágil e testes automatizados de aceitação".

O BDD enfatiza o comportamento do *software*, através de uma linguagem natural, proporcionando uma comunicação entendível entre os envolvidos no projeto, minimizando falhas de comunicação entre os membros do projeto (CHIAVEGATTO, et. al, 2013, p. 335).

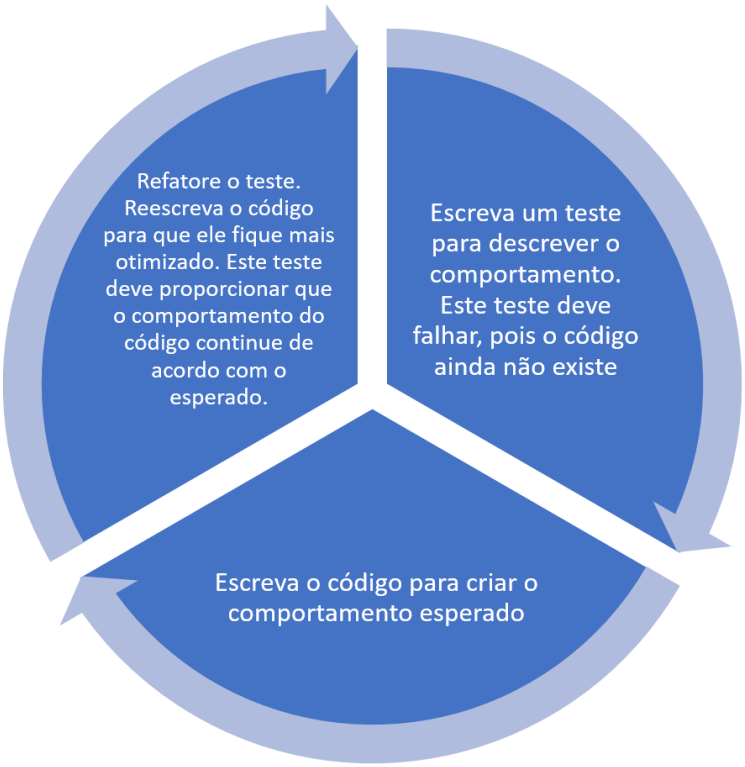
De acordo com *Tutorials Point* (2016), o BDD tem por objetivo fornecer um processo e ferramentas compartilhadas que facilitem a comunicação entre as partes interessadas. Essa metodologia indica o funcionamento do sistema, sua legibilidade e visibilidade. E, por fim, não verifica apenas o funcionamento, mas os interesses do cliente para o produto. "O *Behavior Driven Development* uma técnica que encoraja a colaboração entre desenvolvedores, setores de qualidade e pessoas não técnicas ou de negócio em um projeto de *software"* (AUDY, 2015).

De acordo com *Tutorials Point* (2016), algumas etapas são levadas em consideração num desenvolvimento orientado a comportamento. As partes interessadas, cliente e equipe de desenvolvimento realizam de maneira colaborativa a coleta de especificações/requisitos. Após isso, o dono do produto deverá especificar o comportamento desejado de cada requisito. Com essas especificações, a equipe de desenvolvimento irá analisar e fazer questionamentos, caso necessário. Portanto, será oferecida uma análise comparativa do produto atual com o produto proposto, a fim de identificar qual obteve o melhor funcionamento. "O BDD concentra-se em cenários baseados em ideias com comunicação contínua com o cliente à medida que o

desenvolvimento avança. Não se baseia em promessas e busca facilitar a absorção de mudanças" (TUTORIALS POINT, 2016).

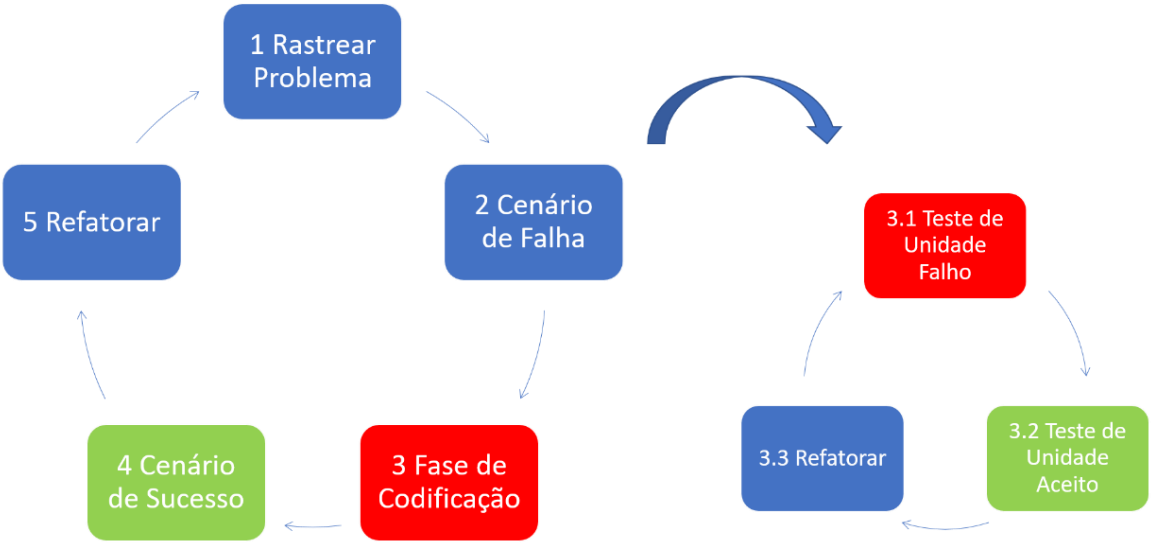
Segundo North (2006), o BDD descreve a utilização do *software*, a fim de promover colaboração e comunicação entre os envolvidos no projeto. Com isso, busca trabalhar os requisitos de modo que sejam tratados adequadamente para a etapa de implementação. Na Figura 7 é apresentado o fluxo de funcionamento do BDD.

Figura 7 - Fluxo de funcionamento do BDD



O desenvolvimento orientado a comportamento é uma abordagem ágil que objetiva a interação entre as partes interessadas. Esta abordagem trabalha diretamente escrevendo casos de testes para pessoas técnicas e não-técnicas possam compreender. Esse tipo de abordagem contribui para o desenvolvimento compartilhado, ou seja, visa proporcionar uma melhor colaboração durante o processo de desenvolvimento de *software*. Com isso, o BDD pode ser visto como dois ciclos relacionados, conforme é ilustrado na Figura 8.

Figura 8 - Processo do BDD e sua relação com TDD



Fonte: Adaptado de FARAHMANDIAN, 2016

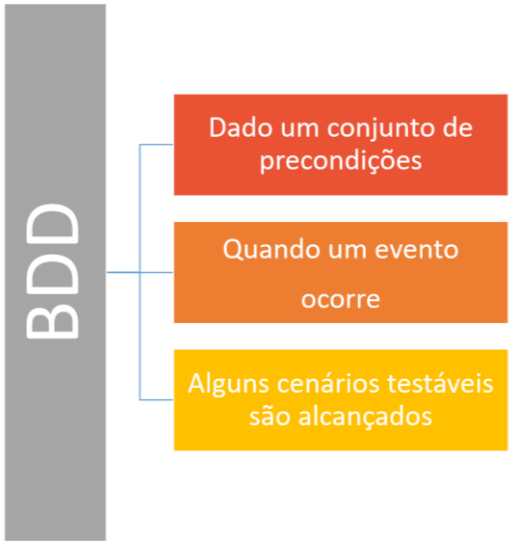
A Figura 8 apresenta a relação existente entre os dois processos de testes que são o TDD e o BDD. No TDD, o processo é iniciado a partir da escrita do teste, após isso inicia-se a etapa de codificação da aplicação do teste criado. Baseado nisso, utilizando os princípios de desenvolvimento guiado por comportamento do BDD, a ordem de prioridade dos testes é definida de acordo com a sua relevância para o usuário.

O BDD deve aderir uma estrutura que aborda de maneira corretamente as histórias e os cenários de seu comportamento. Logo, devem ser definidos, como:

1) História
    a) Como um [papel]
    b) Eu quero [recurso]
    c) Então [benefício]
2) Cenário
    a) Dado algum contexto inicial,
    b) Quando ocorre um evento,
    c) Em seguida, alguns resultados devem ser assegurados.

Segundo *Tutorials Point* (2016), isto significa que “Quando um recurso é executado, o benefício resultante é para a pessoa que joga o papel”. Na Figura 9 é possível observar a apresentação do cenário do processo do BDD.

Figura 9 - Cenário do BDD



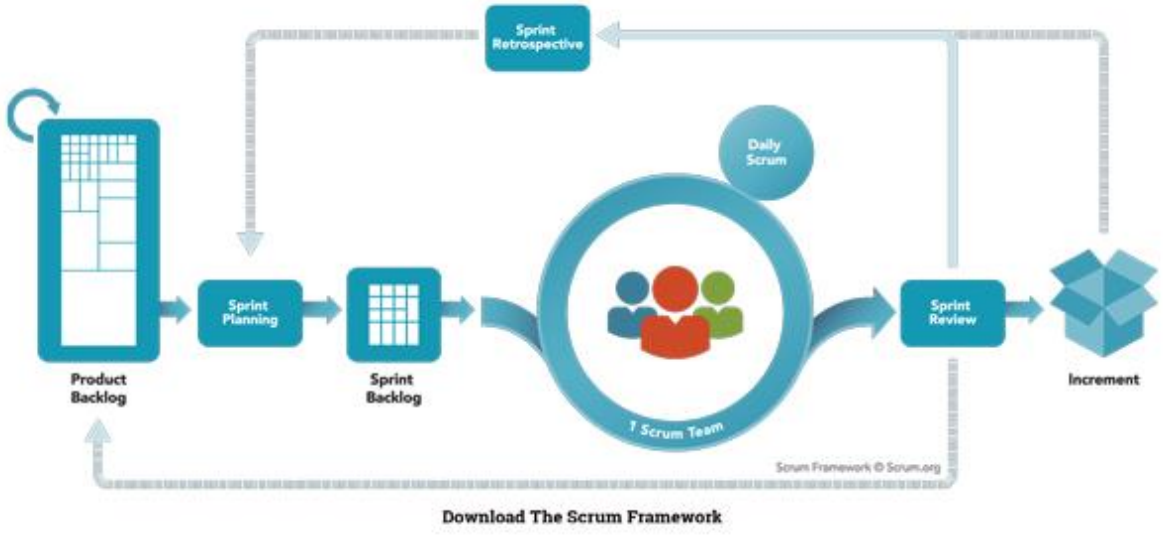
Segundo Audy (2015), o BDD trata-se de um processo colaborativo para a construção de cenários principais e alternativos. Com esta técnica é possível escrever cenários de maneira que seja possível verificar o comportamento do *software* com o que foi solicitado pelo cliente.

2.5 SCRUM

O *Scrum* é uma metodologia ágil, utilizada para o gerenciamento e desenvolvimento de produtos complexos. É uma abordagem iterativa e incremental que visa entregar o produto para o cliente. É baseado num conjunto de valores que trabalham para melhor explorar o sucesso do projeto. De acordo com o *Scrum Alliance* (2017) os cinco valores do *Scrum* são: foco, coragem, compreensão, comprometimento e respeito.

A Metodologia Ágil *Scrum* tem por objetivo orientar nas atividades de coleta de requisitos, análise, projeto, evolução e entrega. O fluxo é orientado por *sprints*, ou seja, a composição e complexidade da *sprint* é definida de acordo com o produto, esta *sprint* é composta por atividades: requisitos, análise, projeto, evolução e entrega, como ilustrado na Figura 10.

Figura 10 - Ciclo do *Scrum*



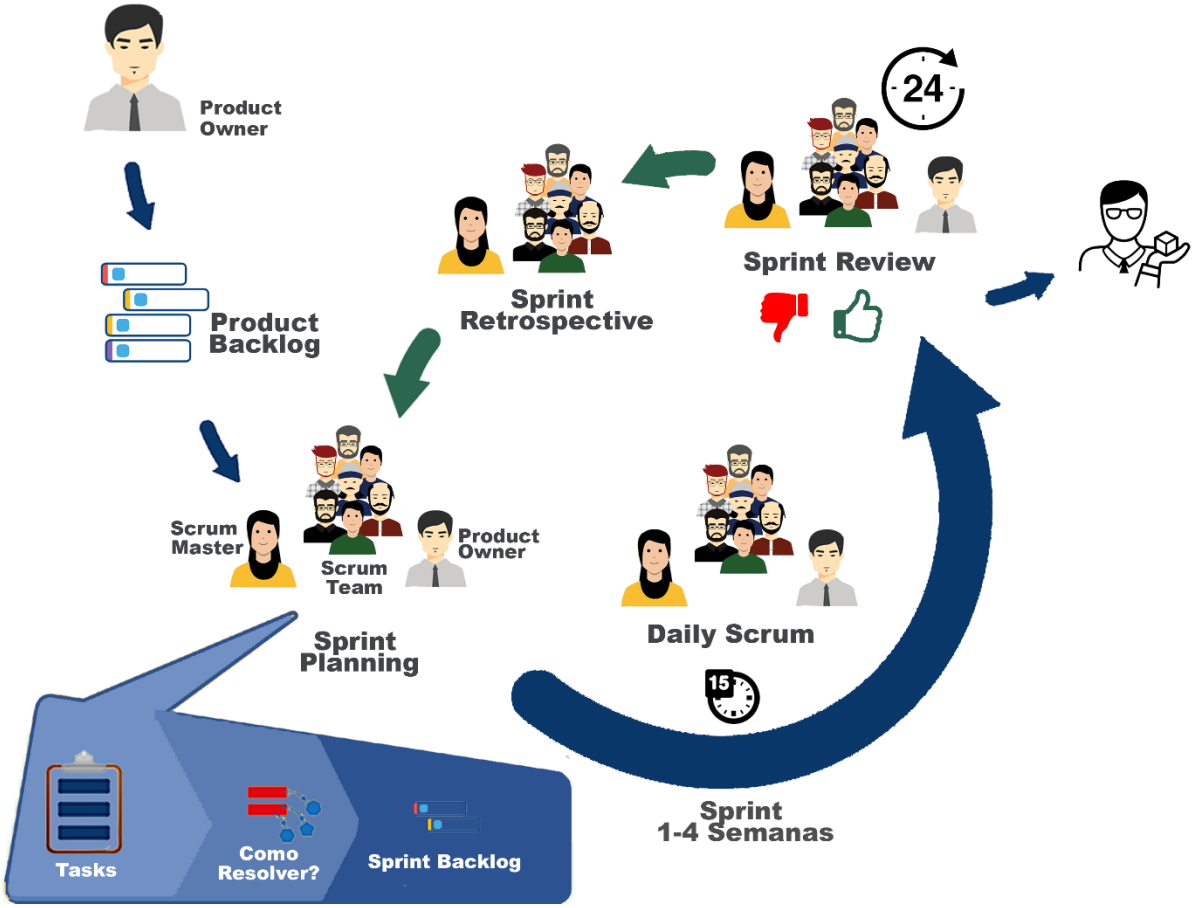
Fonte: Adaptado de Schwaber; West, 2017

De acordo com Schwaber e West (2017), os eventos do *Scrum* são utilizados para criar regularidade e minimizar a necessidade de reuniões não planejadas no processo de *Scrum*. O evento segue concepção de horário, ou seja, uma vez que um *Sprint* começa, sua duração é fixa e não pode ser alterada. Os artefatos definidos pelo *Scrum* são projetados especificamente para transpor as informações mais relevantes para que todos tenham o mesmo entendimento do artefato.

2.5.1 Eventos do Scrum

O ciclo de desenvolvimento do *Scrum* é chamado de *Sprint*, que é composta pelos eventos do *Scrum*. Durante a execução da *Sprint* são realizadas algumas reuniões, que são*: Sprint Planning*, *Daily Scrum*, *Sprint Review* e *Sprint Retrospective*. De acordo com Schwaber e Sutherland (2013, p. 8), os eventos são criados para estabelecer uma rotina e minimizar a necessidade de reuniões não definidas no *Scrum*. Segundo Sabbagh (2013, p. 194), os eventos possuem duração definida, chamada de *time-boxed*, de tal modo que todo evento tem uma duração máxima. “Estes eventos são especificamente projetados para permitir uma transparência e 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 e SUTHERLAND, 2013, p. 8). Na Figura 11 é apresentado o ciclo de trabalho dos eventos do *Scrum* relacionados aos respectivos papéis.

Figura 11 - Ciclo de eventos do *Scrum*



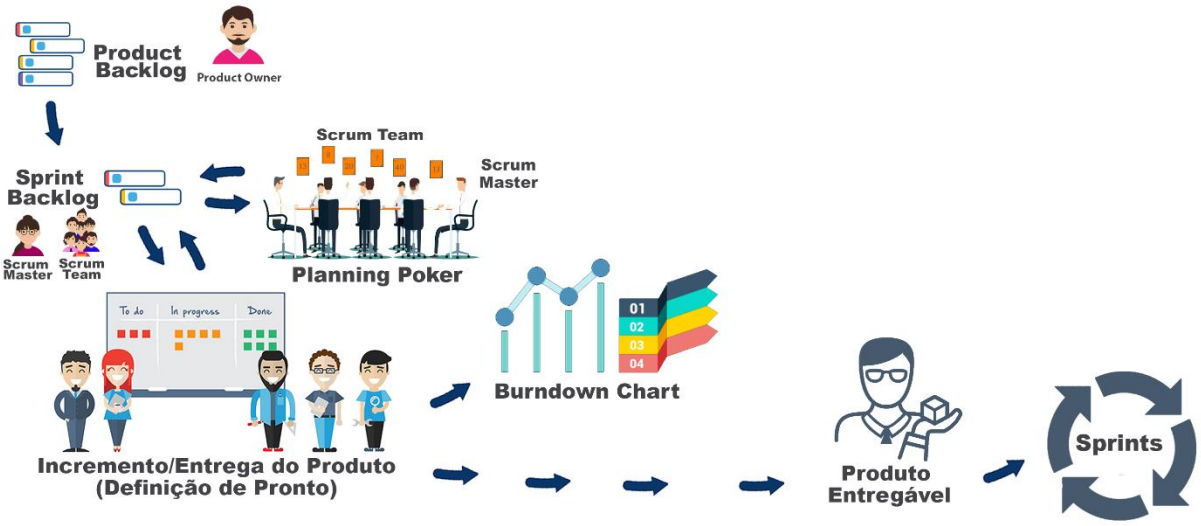
A Figura 11 ilustra o ciclo de eventos do *Scrum*. Nela são apresentados todos os eventos e papéis envolvidos em um processo de desenvolvimento que utiliza a metodologia *Scrum*. Os eventos do *Scrum* são interdependentes, ou seja, muitos dos eventos dependem de outros para executar. O *Product Backlog* é uma lista de requisitos, também chamados de *User Stories*, que representam o produto. Qualquer responsável da equipe pode alterar o *Backlog*, mas é o *Product Owner* que é responsável por definir a ordem de prioridade em que as *User Stories* serão desenvolvidas. Após a concepção do *Product Backlog*, é realizado o planejamento das atividades, chamada de *Sprint Planning*. Nessa etapa são realizados questionamentos sobre os pontos apresentados no *Product Backlog*, como também soluções e, por fim, são gerados conjuntos de atividades da *Sprint* e alocadas para seus respectivos responsáveis. Com a realização do planejamento da *Sprint*, a próxima etapa é a execução da *Sprint*. Ela é realizada durante todo o projeto, uma após a outra e tende a variar entre uma e quatro semanas. Além disso, para ser possível essa execução, é necessária a participação de todo o *Team Scrum*: equipe de desenvolvimento, *Product Owner* e *Scrum Master*. Para que a *Sprint* seja executada com eficiência, são realizadas as *Daily Scrum*. A *Daily Scrum* é uma reunião de curta duração realizada diariamente pelo time de desenvolvimento do *Scrum*. Com

a realização dessa reunião é esperado que sejam apresentados os impedimentos ocorridos, os avanços obtidos e um planejamento informal das atividades do próximo dia. Ao final de toda *Sprint* é realizada a revisão da *Sprint*. No *Sprint Review* o Time de Desenvolvimento e o *Product Owner* identificam se todas as atividades foram executadas conforme apresentado e determinado no *Sprint Planning*. Se após a verificação do *Sprint Review* for identificado algum problema, esses problemas são analisados na *Sprint Retrospective*. A *Sprint Retrospective* tem por objetivo também a inspeção e adequação das atividades da equipe de desenvolvimento, com o intuito de buscar a melhoria contínua no projeto. Se a realização da *Sprint* for de acordo com o esperado, o resultado obtido ao final implica em um produto entregável para o cliente.

2.5.2 Artefatos

Um artefato é um documento que, de acordo com um padrão previamente estabelecido, descreve alguma característica do produto, que pode se referir à parte técnica, financeira, estrutural etc. Os artefatos são produzidos com o objetivo de descrever e compartilhar as informações conhecidas acerca do produto, de modo que todos os envolvidos no projeto possam ter acesso. Um artefato pode ser: um documento (especificação do *software*), um modelo (diagrama de classes) e um item de um modelo (descrição de uma classe). Cada tipo de artefato tem como objetivo representar uma característica do produto, de forma estática ou dinâmica, de forma que o conjunto de artefatos fornece os insumos necessários para o entendimento do funcionamento do produto. A ocorrência dos eventos do *Scrum* define o uso de alguns artefatos, que são: o *Product Backlog*, *Sprint Backlog* e Incremento/Entrega do Produto. Os Artefatos representam o que foi realizado durante algum evento específico do *Scrum*, proporcionando uma oportunidade para inspeção do que foi trabalhado. “Os artefatos definidos para o *Scrum* são especificamente projetados para maximizar a transparência das informações chave de modo que todos tenham o mesmo entendimento dos artefatos.” (SCHWABER e SUTHERLAND, 2013, p. 13). Na Figura 12 é apresentado o ciclo dos artefatos do *Scrum* relacionados aos respectivos papéis e eventos.

Figura 12 - Ciclo de artefatos do *Scrum*



A produção dos artefatos está diretamente relacionada com a participação do *Scrum Team* e o desenvolvimento dos eventos do *Scrum*. O *Product Backlog* é definido pelo *Product Owner*, que tem por objetivo definir funcionalidades desejadas para o produto. Nessa etapa é definida a ordem de prioridade de execução das funcionalidades, de forma que seja flexível a mudanças.

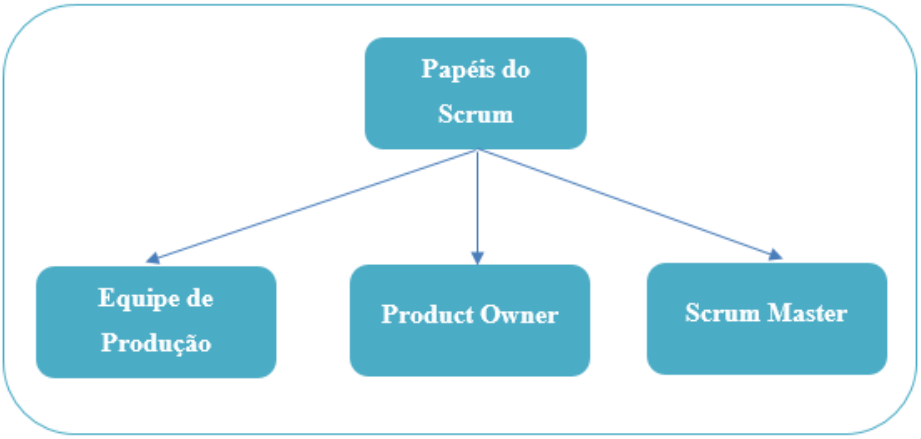
Após a definição do *Product Backlog* é necessário determinar a lista de tarefas que serão executadas pelo *Scrum Team* durante a *Sprint*. Essa lista de tarefas tem como base a ordem de prioridade das atividades definidas pelo *Product Owner*. O *Scrum Team* define tempo e a quantidade de itens para a primeira *Sprint Backlog*. Nessa etapa é necessária a participação constante do *Scrum Master*, com o intuito de liderar e monitorar a execução das atividades. O *Scrum Team* utiliza o *Burndown Chart* para apresentar o progresso diários das equipes na execução da Sprint. E por fim, para tornar a execução da *Sprint Backlog* mais lúdica é realizada a atividade de *Planning Poker*, essa atividade é caracterizada por ser um meio de estimar a dificuldade de cada item do *Product Backlog*. Com isso, é possível proporcionar uma visão abrangente sobre as funcionalidades da *Sprint* que será executada e, consequentemente, um nivelamento da equipe.

Portanto, após a execução da(s) *Sprint*(*s*) planejada(s) é possível definir se o produto está pronto para entrega. Na etapa de Incremento/Entrega do Produto é realizado um acordo formal pelo *Scrum Master* para definir os critérios mínimos para que um item possa ser considerado potencialmente pronto. Nessa etapa devem ser considerados apenas itens potencialmente entregáveis, pois é levado em consideração tudo o que foi acordado sobre a funcionalidade avaliada. Deste modo, é necessário que a transparência nos processos e melhoria contínua faça parte de todo o projeto.

2.5.3 Papéis fundamentais

No *Scrum* são atribuídos apenas três papéis: *Product Owner*, *Scrum Master* e a Equipe de Produção. O objetivo deste time é trabalhar, buscando bons resultados com qualidade e agilidade, de forma proativa e colaborativa, na Figura 13 é ilustrado o fluxo da composição do *Team Scrum*.

Figura 13 - Fluxo de Papéis do *Team Scrum*



Fonte: Adaptado de Sabbagh, 2013, p.58

O *Product Owner*, o *Scrum Master* e a Equipe de Produção formam o *Team Scrum*. Cada papel tem sua importância no fluxo da equipe do *Team Scrum*. Com isso, nas próximas seções serão detalhados os papéis do *Team Scrum* para uma melhor compreensão.

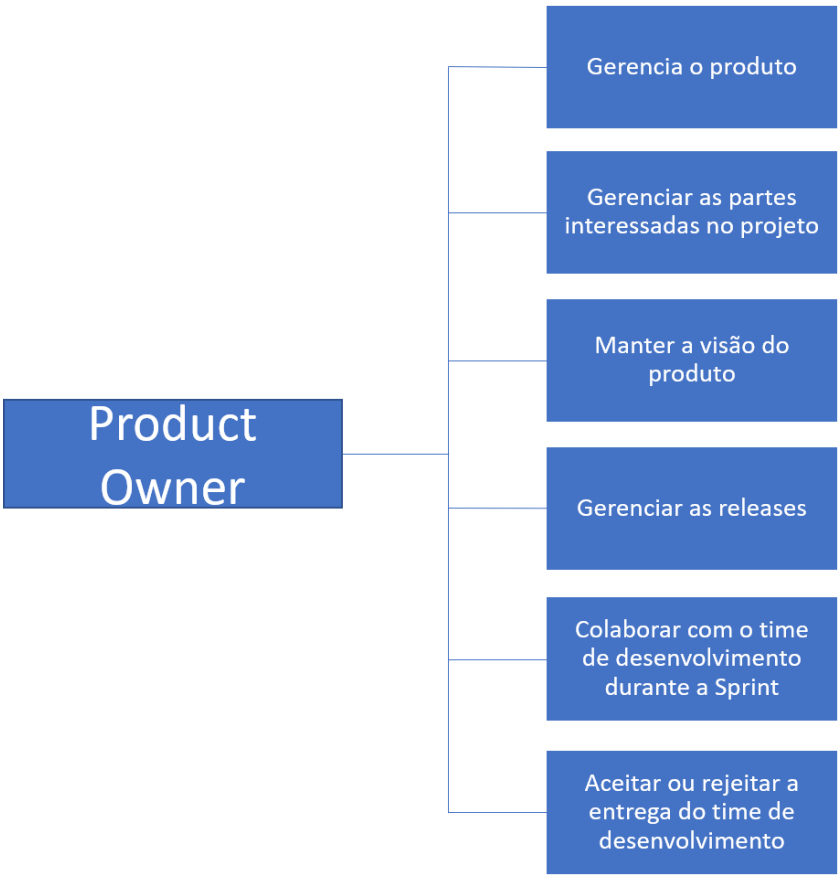
2.5.3.1 Product Owner

O *Product Owner* é a pessoa responsável por garantir o retorno de investimento do produto. Também conhecido como P. O., é o responsável pelas decisões de negócios e por ter o contato direto com os clientes e demais partes interessadas. De acordo com Sabbagh (2013, p. 79) o *Product Owner* é responsável por gerenciar, inserir, detalhar, remover e priorizar as necessidades de negócios do produto no *Product Backlog*, de acordo com o contato direto com o cliente e as partes interessadas.

Segundo Sabbagh (2013, p.80) e Leitão (2010, p.35) o *Product Owner* desempenha um papel importante em um projeto *Scrum*, ele é responsável por grande parte das tomadas de decisões necessárias, como características do produto e a prioridade de execução dos requisitos. Além disso, é responsável por esclarecer e tirar dúvidas sobre o produto, ou seja,

ele representa o especialista do domínio. De acordo com as orientações do *Scrum*, o *Product Owner* deve estar disponível o máximo possível para a equipe de desenvolvimento, a fim de certificar que o projeto esteja de acordo com as necessidades estabelecidas. Na Figura 14 é possível identificar as principais funções do *Product Owner*, segundo Sabbagh (2013).

Figura 14 - Funções do *Product Owner*



O *Product Owner* é responsável pela gestão de toda a parte interessada, ou seja, está relacionado com todos os envolvidos que contribuem para o desenvolvimento do produto final. “Ele é responsável por gerenciar todo o produto, tendo consigo o poder da palavra final” (SABBAGH, 2013, p. 82). É o único que pode alterar o *Product Backlog* e que contém toda a visão do produto final. Sua participação é essencial durante a avaliação de desenvolvimento do produto, logo a escolha do papel de *Product Owner* deve ser bem analisada. O papel de *Product Owner* pode ser representado pelo próprio cliente ou pode ser alguém especificamente contratada pela organização prestadora de serviço para a execução deste papel.

Existem alguns problemas ao escolher o responsável pelo papel de *Product Owner*, e alguns cenários contribuem para o potencial fracasso da execução desse papel. Um ponto

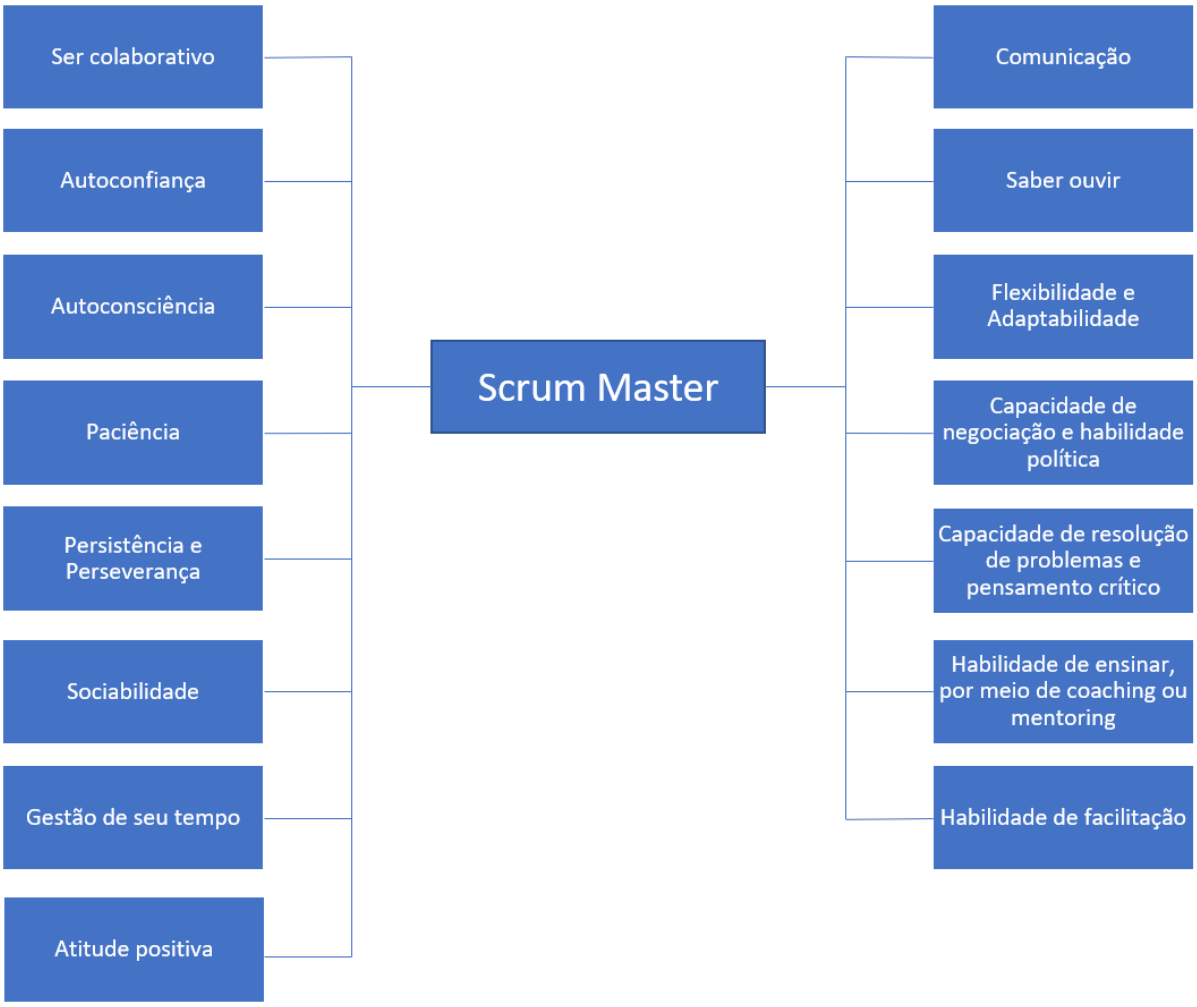
considerado negativo é atribuir uma pessoa que não seja especialista do domínio e que não entenda qual o objetivo do produto.

2.5.3.2 Scrum Master

O papel do *Scrum Master* é facilitar o trabalho do time de desenvolvimento. Seu principal objetivo é garantir com que a equipe siga os princípios do *Scrum*. Além disso, tem por responsabilidade ajudar tanto o Time de Desenvolvimento quanto o *Product Owner* no decorrer do desenvolvimento do projeto. De acordo com Schwaber e Sutherland (2013, p. 7), o *Scrum Master* atende a equipe de desenvolvimento de modo a: treinar em autogerenciamento e interdisciplinaridade, ensinar na concepção de valores, remover impedimentos no processo de desenvolvimento e facilitar os eventos do *Scrum*. Além disso, o *Scrum Master* contribui para a Organização de modo a: treinar e liderar na prática do Scrum e aumentar a eficácia na aplicação do *Scrum* nos processos da Organização.

Para a realização satisfatória do trabalho do *Scrum Master*, é necessário que o responsável possua algumas competências. Sabbagh (2013, pp. 102-103) define um conjunto dessas competências, denominadas *soft skills*, que são apresentadas na Figura 15.

Figura 15 - Competências do *Scrum Master*



As competências apresentadas na Figura 15 são referentes às habilidades comportamentais e pessoais que são desejáveis para um *Scrum Master*, de acordo com Sabbagh (2013, pp. 102-103). Desse modo, o *Scrum Master* deve proporcionar maior produtividade e qualidade no trabalho, facilitando no desenvolvimento dos eventos do *Scrum*. Portanto, o *Scrum Master* deve facilitar o trabalho do Time *Scrum*, de forma a promover habilidades de organização, comunicação e melhoria contínua no processo de desenvolvimento.

2.5.3.3 Time de Desenvolvimento

O time de desenvolvimento tem como responsabilidade o desenvolvimento do produto e entrega dos itens do *backlog*, além de manter o projeto sempre alinhado com as necessidades do *Product Owner*. Desse modo, de acordo com Rad e Turley (2013, p. 22), o time de desenvolvimento deve contemplar características, como: especialidade na área de

atuação, auto-organização, foco, motivação, multidisciplinaridade e agilidade. Além disso, a composição da equipe não deve ser alterada de modo constante, caso haja a necessidade de mudança, não é aconselhável alteração no decorrer da execução de uma *sprint* e sim ao término.

Segundo Sabbagh (2013, pp. 63-68), para a realização do trabalho do time de desenvolvimento é necessária a execução de atividades, como:

- Planejamento do trabalho: é realizada na reunião de *Sprint Planning* a negociação com o *Product Owner* sobre o que será realizado no decorrer da *Sprint*.
- Realização das tarefas da *Sprint*: é realizado durante a *Sprint*, as tarefas necessárias para a concepção do produto, de acordo com os itens especificados no *Sprint Backlog*.
- Alinhamento do projeto com as necessidades do *Product Owner*: para haver uma boa interação com o *Product Owner* é necessário que haja comunicação sempre que possível sobre os itens da *Sprint* que está sendo executada.
- Identificação e informação de impedimentos ao *Scrum Master*: com a ajuda do *Scrum Master*, o time de desenvolvimento busca prevenir-se de impedimentos durante a realização do seu trabalho. Para que isso ocorra, é necessário que os impedimentos identificados durante a execução das tarefas sejam informados o mais rápido possível.
- Repasse de *Feedback*: o *feedback* constante deve ser realizado durante toda a execução da *Sprint*. A realização do *feedback* permite adequações e incrementos no produto e processo de desenvolvimento de acordo com o que foi repassado pelo cliente. Essas adequações e incrementos são discutidos e avaliados na reunião de revisão da *Sprint*.
- Entrega constante de valor para o cliente: o time de desenvolvimento tem por prioridade entregar o que é de valor para o cliente. Com isso, ao gerar valor em cada *Sprint*, a equipe possibilita a entrega constante de valor para o cliente.

De acordo com Schwaber e Sutherland (2013, p. 6), o time de desenvolvimento do *Scrum* consiste em profissionais especializados que tem por objetivo a entrega de um produto potencialmente pronto ao final de cada *Sprint*. Portanto, a execução das atividades citadas acima resulta no aperfeiçoamento e eficiência no desenvolvimento do projeto.

3 MATERIAIS E MÉTODOS

3.1 MATERIAIS

Para o desenvolvimento deste trabalho, foram utilizados os seguintes materiais e tecnologias:

- *Unified Modeling Language* (UML): a Linguagem de Modelagem Unificada é uma linguagem para estruturação de projetos de *softwares.* Segundo Costa (2001), a UML define um conjunto básico de diagramas e notações que representam múltiplas perspectivas do sistema sobre análise e desenvolvimento. Esta linguagem de modelagem é caracterizada uma ferramenta de suporte para ambientes de desenvolvimento. Por esta razão, este tipo de padronização trabalha diretamente com uma metodologia afim de representar as múltiplas perspectivas de representação de um projeto. De acordo com Siegel (2005), a Linguagem de Modelagem Unificada está dividida em três categorias: diagramas de estrutura; diagramas de comportamento; e diagramas de interações. Os Diagramas de Estrutura incluem o Diagrama de Classe, Diagrama de Objetos, Diagrama de Componentes, Diagrama de Estrutura Composta, Diagrama de Pacote e Diagrama de Implantação. Os Diagramas de comportamento incluem o Diagrama de Caso de Uso, Diagrama de atividade e Diagrama de máquina de estado. Os Diagramas de Interação incluem o Diagrama de Sequência, Diagrama de Comunicação, Diagrama de Temporização e Diagrama de Visão de Interação.
- *HyperText Markup Language* (HTML): de acordo com W3C (2017), HTML é uma linguagem ou idioma de marcação para criar páginas da web. Esta linguagem objetiva uma estrutura de marcação, caracterizando por blocos e representados por *tags*. Estas *tags* representam um determinado conteúdo especificado de acordo com a etiqueta concedida.
- *Cascading Style Sheets* (CSS): segundo a W3C (2017), CSS é um mecanismo simples para adicionar estilo (por exemplo, fontes, cores) aos documentos da Web. Este mecanismo é conhecido atualmente como folha de estilo, o CSS tem como objetivo descrever como os elementos do HTML serão exibidos (W3Schools, 2017).
- *JQuery*: de acordo com *Jquery* e W3Schools (2017), é uma biblioteca de *JavaScript* rápida, pequena e rica em recursos. Esta biblioteca trabalha com manipulação de documentos HTML, manipulação de eventos, animação e *Ajax*. Além disso é uma biblioteca simples e compatível com diversas versões de navegadores.

3.2 MÉTODOS

O desenvolvimento deste trabalho teve início com a busca e estudo do referencial teórico com base na análise da pesquisa quali-quantitativa realizada com empresas palmenses em 2017 por Ribeiro (2017). Com base nessa pesquisa, foi possível realizar os seguintes procedimentos: análise do processo de desenvolvimento *Scrum* para a identificação de práticas adequadas, definição das etapas do processo, como também a definição dos artefatos do processo, construção do modelo de negócios, além do desenvolvimento da documentação e implementação da representação visual do Processo de Gerenciamento e Desenvolvimento de *Software.*

Para a realização desta metodologia, alguns procedimentos foram apresentados como principais para o desenvolvimento do processo de gerenciamento e desenvolvimento de *software.* Nos tópicos abaixo serão apresentados os procedimentos principais que foram realizados antes da elaboração por completa da metodologia deste trabalho. Para a modelagem do processo de gerenciamento e desenvolvimento de *software* foi utilizada uma metodologia quali-quantitativa, cujas etapas são:

- Determinar o domínio e o escopo do processo de desenvolvimento e gerenciamento: consiste em definir algumas premissas: qual o domínio de consistência do processo de desenvolvimento e gerenciamento? Para qual contexto que o processo será aplicado? Quais os principais problemas que o processo de desenvolvimento e gerenciamento deverá solucionar? Quem irá utilizar o processo de desenvolvimento e gerenciamento? As conclusões para estas questões podem ser alteradas no decorrer do projeto, porém elas compreendem a demarcação do escopo do problema. Com isso, esta metodologia tem como objetivo a compreensão do contexto de aplicação do processo de desenvolvimento e gerenciamento de *software.*
- Considerar a reutilização de metodologias ágeis existentes: esta etapa consiste em verificar a disponibilidade de metodologias ágeis que auxiliam no desenvolvimento do *software*, pois a disposição de processos e ferramentas que possam auxiliar no desenvolvimento e compreensão do processo é de grande importância para a criação e consistência do processo.
- Definir ciclo de vida do processo: esta etapa consiste no amadurecimento da metodologia em desenvolvimento, apresentando de maneira detalhada o funcionamento do processo. A composição dessa etapa deverá ser apresentada de

maneira a apresentar as interações do desenvolvimento do processo, a fim de detalhar o funcionamento do método de desenvolvimento e gerenciamento de *software*.

Para a implementação do site do processo, foram realizados os seguintes procedimentos:

- Estudo dos padrões W3C, HTML, CSS: consiste em proporcionar entendimento acerca de padrões de marcação e estilo para criação de ambientes virtuais;
- Estudo de uma biblioteca para a manipulação de uma página web: tem o objetivo de proporcionar entendimento acerca da biblioteca de *JavaScript* para que seja possível a manipulação de documentos HTML, manipulação de eventos e animações;
- Estudo de técnicas de acessibilidade para apresentação de informação: esta etapa é muito importante para que o processo desenvolvido possa ser disponibilizado virtualmente, pois consiste na compreensão de padrões de acessibilidade para páginas web; e
- Criação de um módulo para a manipulação visual da documentação do processo: esta etapa consistiu no desenvolvimento de um ambiente virtual web para a apresentação do processo de desenvolvimento e gerenciamento de *software* criado no trabalho.

Portanto, a composição da metodologia de desenvolvimento deste trabalho foi concebida conforme ilustrado na Figura 16.

Figura 16 - Fluxo do processo das etapas desenvolvidas do projeto

Conforme apresentado na Figura 16, acima, a metodologia deste trabalho é composta por um conjunto de etapas inter-relacionados. Estas etapas serão detalhadas nos tópicos seguintes:

- Estudo do referencial teórico: esta etapa visa a apresentação de uma visão geral dos conceitos abordados no trabalho, como: metodologias ágeis, testes de verificação, testes de validação, *Scrum* e BDD. Ainda, propõe um aprendizado sobre a utilização de processos de desenvolvimento ágil e indica algumas abordagens de qualidade, como testes de verificação e validação no processo de desenvolvimento de *software*.
- Análise dos resultados da pesquisa: O desenvolvimento deste trabalho envolveu a análise da pesquisa quali-quantitativa realizada com empresas palmenses em 2017 por Ribeiro (2017). O objetivo foi de avaliar a evolução e necessidade na utilização de processos de desenvolvimento nas empresas e órgãos participantes na pesquisa, bem como, coletar informações sobre a utilização de processos de desenvolvimento de *software* em Palmas – TO. A pesquisa teve como propósito obter um melhor conhecimento relacionado a realidade da área de desenvolvimento de *software* no mercado de TI e dos conceitos apresentados neste trabalho, assim como a utilização de processos ágeis. A escolha do processo de desenvolvimento *Scrum* ocorreu por meio da análise da pesquisa realizada por Ribeiro (2017). Esta pesquisa apresentou que mais de 60% das empresas de Palmas-TO que adotam práticas ágeis utilizam o *Scrum*. Ainda, a análise possibilitou observar que as empresas que trabalham com prática de desenvolvimento ágil não necessariamente atendem a todos os princípios de uma metodologia ágil, pois grande parte de sua utilização é feita de maneira parcial.
- Construção do modelo de negócios: para o desenvolvimento deste trabalho, foi criado um modelo de negócio para o processo, a fim de descrever os aspectos relacionados à utilização do processo de gerenciamento e desenvolvimento no mercado de trabalho. Para isso, foram analisados os seguintes aspectos: segmentos de clientes; proposta de valor; canais; relacionamento com clientes; fontes de receita; recursos principais; atividades chaves; parceiros chaves; e estrutura de custos. Dessa forma, foram verificados os benefícios que esse processo proporcionará para os seus clientes, os recursos necessários para que sua utilização seja possível e a forma como será estabelecida a comunicação com o cliente.
- Análise do processo de desenvolvimento *Scrum* para a identificação de práticas adequadas ao cenário: essa análise foi realizada com o intuito de entender o cenário de desenvolvimento de *software* em empresas privadas, com o objetivo de definir uma proposta de um processo ágil de gerenciamento e desenvolvimento de *software*

adequado à realidade das empresas. E, com isso, aplicar um treinamento referente à abordagem e sua utilização a fim de melhor atender às necessidades das empresas.

- Definição das etapas do processo: essa etapa tem como objetivo apresentar a definição do fluxo de funcionamento do processo de desenvolvimento e gerenciamento de *software* que será desenvolvido neste trabalho. O fluxo será apresentado de maneira gráfica, relacionando os papéis às etapas em que estão envolvidos.
- Definição dos artefatos do processo: nessa etapa foram definidos os artefatos utilizados para a composição do processo e apresentação da documentação. Inicialmente, os artefatos da metodologia *Scrum* foram analisados a fim de identificar qual deles poderão ser utilizados no processo desenvolvido neste trabalho. Posteriormente, foram analisadas as adaptações da metodologia *Scrum* realizadas pelas empresas que participaram da pesquisa, a fim de identificar aquelas que proporcionaram melhores resultados. Por fim, foram analisadas as melhores práticas apresentadas no referencial teórico, que se adequam ao cenário das empresas de desenvolvimento de *software*.
- Modelagem do Processo e desenvolvimento do site do SIDD: nesta etapa foi planejada a criação da documentação do processo de desenvolvimento e gerenciamento de *software* e apresentação visual do fluxo funcional do processo.

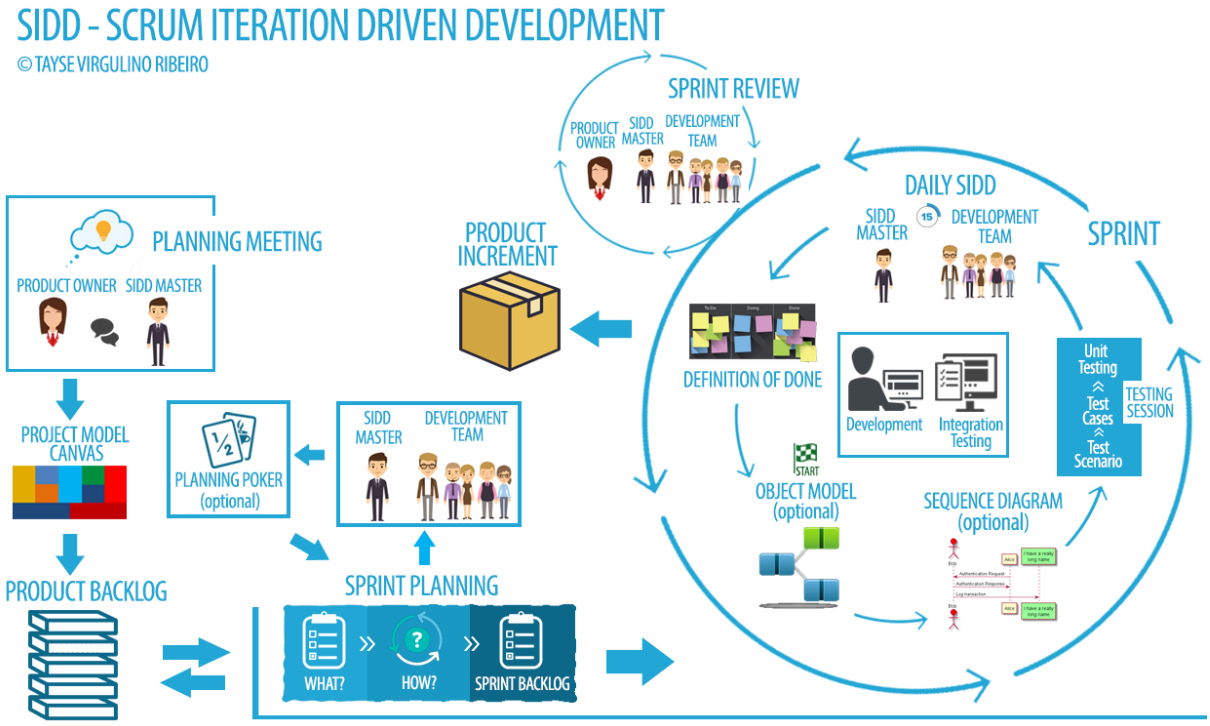
4 RESULTADOS

Esta seção tem por objetivo descrever e apresentar o fluxo do trabalho realizado para a modelagem do processo de desenvolvimento e gerenciamento de software SIDD.

4.1 ESTRUTURA

A **Erro! Fonte de referência não encontrada.**Figura 17 apresenta o fluxo de trabalho do p rocesso desenvolvido.

Figura 17 - Estrutura do Processo



A divisão do processo consiste na realização dos eventos, na execução dos artefatos e na participação do SIDD *Team*. O processo SIDD é composto pelos seguintes eventos: ***Planning Meeting***, ***Sprint Planning***, ***Planning Poker*** *(*opcional*)*, ***Sprint***, ***Daily*** **SIDD** e ***Sprint Review***. Na execução desses eventos são gerados os seguintes artefatos: ***Project Model Canvas***, ***Product Backlog***, ***Sprint Backlog***, ***Object Model*** *(*opcional*)*, ***Sequence Diagram*** (opcional), ***Testing Session***, ***Definition of Done*** e ***Product Increment***. Com isso, para a realização do ciclo do SIDD é necessária a participação do **SIDD *Team***. Esse time é composto por pessoas que exercem os seguintes papéis, que são: ***Product Owner***, **SIDD *Master*** e ***Development Team***. As seções a seguir descrevem as características destes itens de forma mais detalhada.

4.1.1 Papéis e seus fluxos de trabalhos

No processo SIDD são definidos três papéis: *Product Owner*, SIDD *Master* e *Development Team*. O conjunto desses papéis formam o SIDD *Team*. Esses papéis atuam na busca de resultados que prezam por qualidade e agilidade, de forma proativa e colaborativa.

Assim como no *Scrum*, o *Product Owner* é o responsável pelas decisões de negócios e por ter o contato direto com partes interessadas para a execução do produto. Assim como no Scrum, o processo SIDD representa o especialista do domínio, que tem como prioridade esclarecer dúvidas sobre o desenvolvimento do produto e certificar-se de que o projeto esteja de acordo com as necessidades estabelecidas pelo cliente.

O papel do SIDD *Master* tem como responsabilidade liderar e facilitar o trabalho do *Development Team*, de forma a promover habilidades de organização, comunicação e melhoria contínua no processo de desenvolvimento. O SIDD *Master* é um dos únicos papéis que está presente o tempo todo no processo de concepção do produto, pois ele tem como objetivo monitorar o andamento de todas as atividades e manter o projeto sempre alinhado com as necessidades do *Product Owner*.

Por fim, o papel do *Development Team* tem como responsabilidade o desenvolvimento do produto. Desse modo, o time deve contemplar algumas características, como: especialidade na área de atuação do projeto em desenvolvimento, como também auto-organização, foco, motivação, disciplina, agilidade e trabalho em equipe.

O desempenho de um papel está diretamente relacionado a um conjunto de eventos executados e de respectivos artefatos desenvolvidos. Portanto, nesta seção serão apresentados os fluxos de trabalhos dos papéis diretamente relacionados com seus eventos e artefatos.

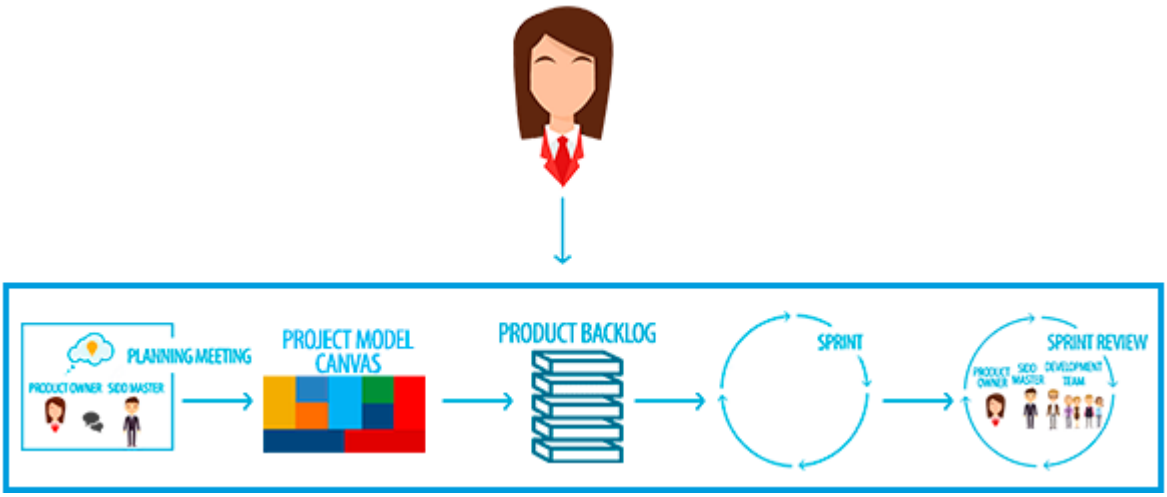
A divisão do processo SIDD consiste na realização dos seguintes eventos: *Planning Meeting*, *Sprint Planning*, *Planning Poker* (opcional), *Sprint, Daily SIDD e Sprint Review*. O objetivo dos eventos no SIDD está relacionado em proporcionar um controle do fluxo de trabalho no decorrer do processo de desenvolvimento. A realização desses eventos está diretamente relacionada ao desempenho de determinados papéis e de respectivos artefatos desenvolvidos.

Portanto, na execução dos eventos do processo SIDD são gerados os seguintes artefatos: *Project Model Canvas*, *Product Backlog*, *Sprint Backlog*, *Object Model* (opcional), *Sequence Diagram* (opcional), *Testing Session*, *Definition of Done* e *Product Increment*.

Na Figura 18 é apresentado o fluxo de trabalho do *Product Owner*, a sua participação no processo SIDD inicia na *Planning Meeting*. Nesta reunião é discutida a necessidade do cliente e o que será gerado de valor. A *Planning Meeting* é uma reunião de planejamento que

tem por objetivo coletar as informações iniciais do projeto para que seja possível identificar as necessidades do cliente. Nessa reunião busca-se desenvolver uma proposta de valor capaz de suprir essas necessidades. Essa reunião ocorre no primeiro contato com o cliente, que envolve os papéis de *Product Owner* e SIDD *Master*.

Figura 18 - Fluxo de trabalho do Product Owner



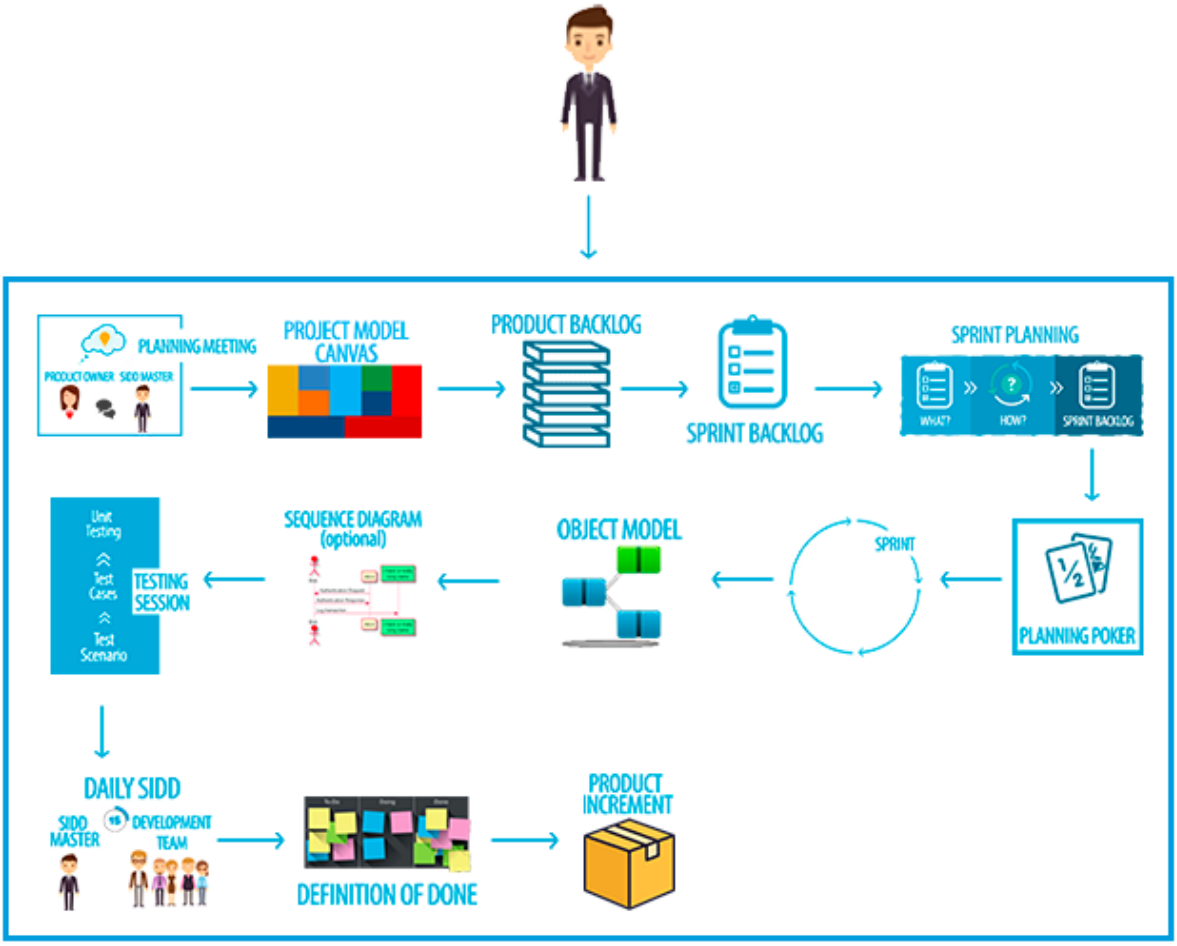
Conforme apresentado na Figura 18, acima, o *Product Owner* e o SIDD *Master* trabalham para definir o *Project Model Canvas*. O *Project Model Canvas* é um modelo de gerenciamento estratégico que permite facilitar o planejamento de novos projetos, sem tanta burocracia e complexidade.

Baseado nisso, são definidos os itens que compõem o *Product Backlog*. Este artefato é caracterizado por ser uma lista de itens contendo todas as funcionalidades desejadas de um produto. Nessa etapa é definida a prioridade de execução das funcionalidades, de forma que sejam flexíveis a mudanças.

Por fim, participa em conjunto com SIDD *Master* e *Development Team* no desempenho da atividade de *Sprint Review*, auxiliando na avaliação em relação ao cumprimento do objetivo da *Sprint* que é um conjunto de atividades realizadas durante todo o projeto, uma após a outra, caracterizando após a *Sprint Review* como a entrega do produto.

Na Figura 19 é apresentado o fluxo de atividades do SIDD *Master*. Conforme é apresentado, sua participação inicia na *Planning Meeting*. Após isso, o *Product Owner* e o SIDD *Master* trabalham para definir o *Project Model Canvas*. Como consequência, são definidos os itens que compõem o *Product Backlog*.

Figura 19 - Fluxo de trabalho do SIDD Master



Além disso, conforme é apresentado na Figura 19, após o *Product Backlog* é realizado o planejamento do *Sprint Backlog* no evento de *Sprint Planning*. A *Sprint Planning* é uma reunião de planejamento da *Sprint*, na qual as funcionalidades são descritas e posteriormente, atribuídas como tarefas de acordo com sua prioridade. O *Sprint Backlog* é uma lista de tarefas, esses itens são extraídos do *Product Backlog*, pela equipe, com base nas prioridades definidas pelo *Product Owner*. Nessa etapa, todo o SIDD *Team* define tempo e quantidade de itens para o *Sprint Backlog*.

No ato do planejamento do *Sprint Backlog* o SIDD *Master* e o *Development Team* realizam também o *Planning Poker*. O *Planning Poker* é uma técnica utilizada para determinar o esforço para o desenvolvimento de uma determinada tarefa. É caracterizada por ser um meio de estimar a dificuldade e tempo de cada item do *Product Backlog*. É uma técnica e ao mesmo tempo um jogo que é utilizado pelo *Development Team*, baseado no consenso e que pode ser aplicado de diversos modos.

Durante a execução da *Sprint*, o SIDD *Master* participa auxiliando e liderando o *Development Team* na construção dos artefatos que auxiliam no desenvolvimento do produto,

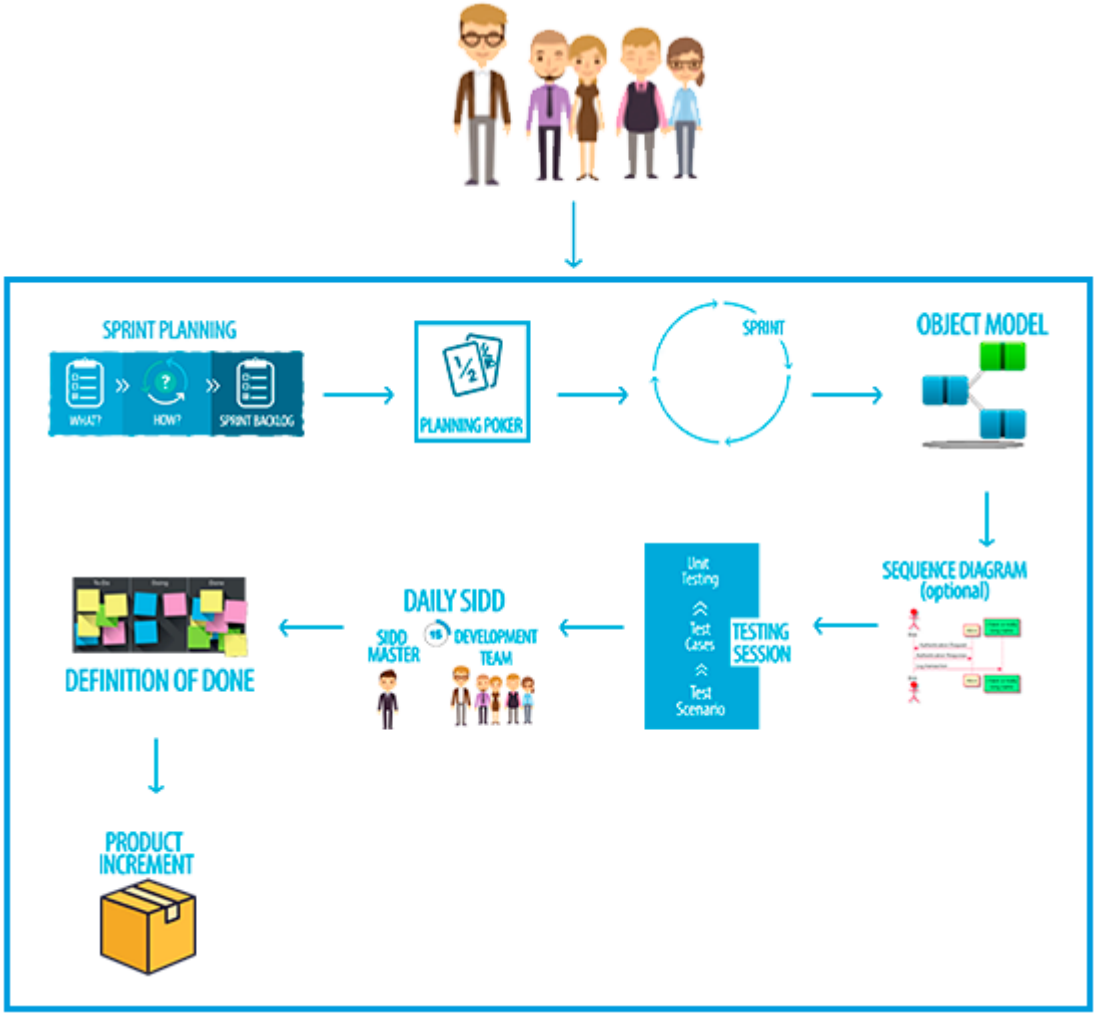
como: *Object Model (opcional), Sequence Diagram* (*opcional*), *Testing Session*, *Definition of Done* e *Product Increment*.

O *Object Model* é um modelo que busca capturar uma estrutura básica de cada objeto, apresentando tecnicamente, seus relacionamentos, atributos e classes. Já o *Sequence Diagram* é um artefato utilizado para representar de maneira detalhada a sequência dos processos em formato de linha de tempo. Além disso, é um diagrama que tem como objetivo apresentar de maneira dinâmica como são realizadas as trocas de mensagens entre os objetos em cada operação. No *Testing Session* será necessário que o SIDD *Master* defina uma política de testes de *software* para que o *Development Team* possa realizar a implantação. Essa sessão haverá três etapas de testes: Test Scenario, Test Cases e Unit Testing. Cada item do *Sprint Backlog* irá passar pela Sessão de teste. Ao final de cada etapa deverá ser gerado um relatório a critério do SIDD *Master* com as políticas implantadas identificando erros encontrados e propostas de solução.

O *Definition of Done* é um acordo realizado entre o SIDD *Master* e o *Product Owner* para definir critérios mínimos para um item potencialmente pronto. Ao contemplar os critérios descritos no acordo, os itens são caracterizados como itens entregáveis e são inseridos/incluídos no *Product Increment*. O *Product Increment* é a soma de todos os itens do *Sprint Backlog* concluídos durante uma *Sprint*, sendo caracterizado como implantação da atividade. Ao final desta, o item deve ser avaliado por todo SIDD *Team*, definido como pronto e por fim liberado.

A **Erro! Fonte de referência não encontrada.**Figura 20 apresenta, o fluxo de t rabalho do *Development Team* no processo desenvolvido nesse trabalho.

Figura 20 - Fluxo de trabalho do Development Team



Conforme apresentado na Figura 20, o *Development Team* inicia sua participação no evento *Sprint Planning*. No ato desse planejamento, o time realiza também o *Planning Poker*. Após isso, na execução da *Sprint* o *Development Team* contribui na realização da reunião diária, conhecida como *Daily* SIDD.

A *Daily* SIDD é uma reunião que tem uma curta duração de 15 minutos para o time sincronizar as atividades e criar um plano informal para as próximas 24 horas. Com essa reunião espera-se que seja apresentado o progresso alcançado no projeto no dia anterior e os impedimentos relacionados às atividades que ainda precisam ser desenvolvidas. A *Daily* SIDD é realizada durante toda a *Sprint*. Nessa reunião estão envolvidos os papéis SIDD *Master* e *Development Team*.

Durante a execução da *Sprint* o time participa na geração de artefatos que auxiliam no desenvolvimento do produto, como: *Object Model* (opcional), *Sequence Diagram* (opcional), *Testing Session*, *Definition of Done* e *Product Increment*.

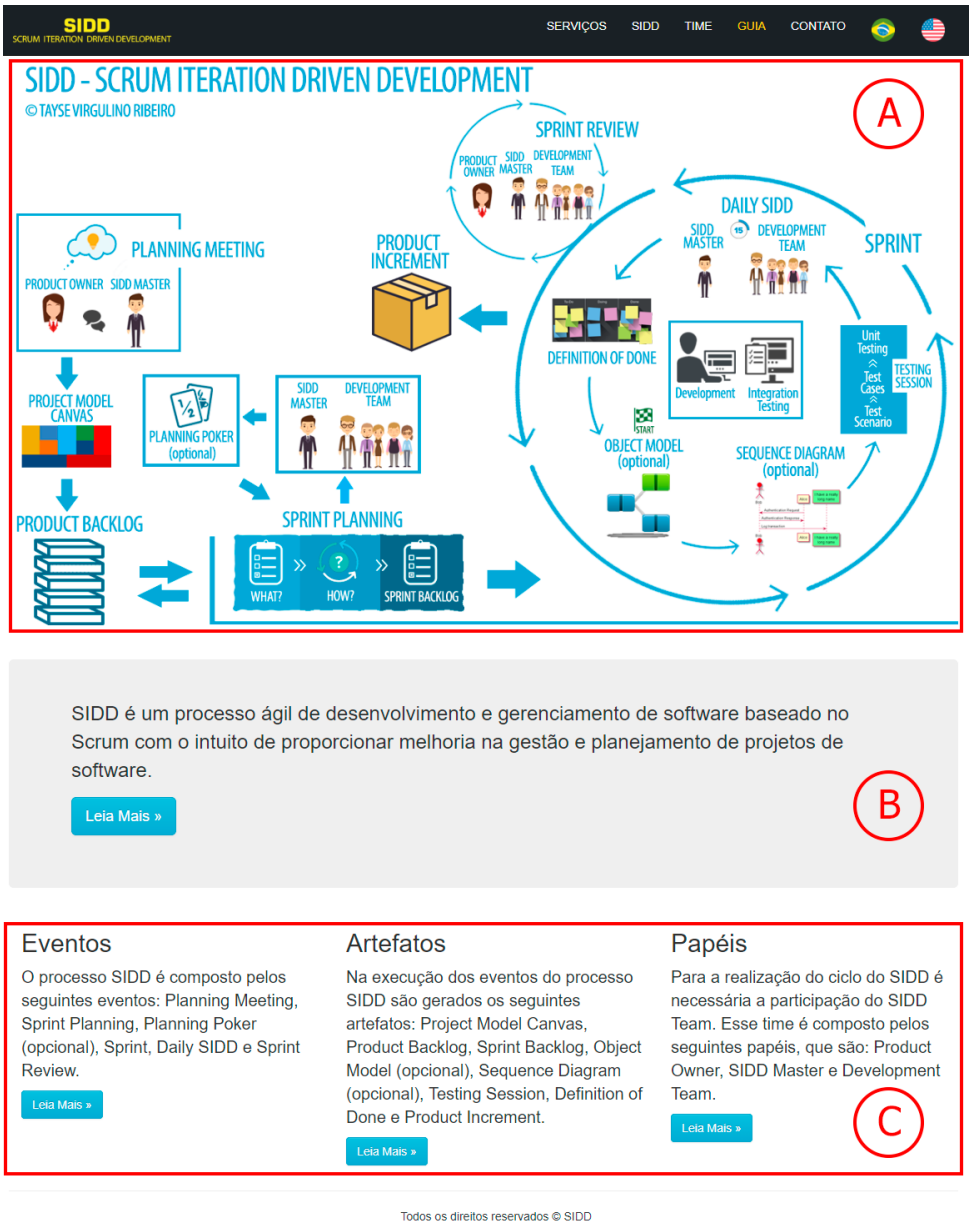
Nesta seção foi apresentada a estrutura do processo de desenvolvimento e gerenciamento *Scrum Iteration Driven Development.* A estrutura foi descrita de modo a representar a relação entre os papéis, eventos e artefatos durante a execução do processo de desenvolvimento de *software.* A partir disso, será apresentado o site que foi desenvolvido com o intuito de se tornar uma ferramenta de auxílio durante o desenvolvimento de um projeto que fizer uso do processo SIDD.

4.2 APRESENTAÇÃO DO SITE SCRUM ITERATION DRIVEN DEVELOPMENT

Para auxílio na implantação do processo de gerenciamento e desenvolvimento de *software* SIDD foi desenvolvido um site para apresentação gráfica e textual do processo. Essa área interativa foi desenvolvida com informações relacionadas ao desenvolvimento e gerenciamento de software do processo, que pode ser acessado através do seguinte link http://metodologiasidd.com.br/. O desenvolvimento do site foi visando principalmente o desenvolvimento da área de guia do processo (http://metodologiasidd.com.br/guide.html), no qual é possível obter a orientação, apresentando as principais áreas do processo, que são: papéis, eventos e artefatos.

Como é ilustrado na Figura 21, na página inicial do Guia do Processo SIDD é apresentado o fluxo principal do processo (A), como também uma breve descrição sobre o processo (B) e por fim, os links de acesso para as áreas mais importantes do *Scrum Iteration Driven Development*, que são os eventos, artefatos e papéis (C).

Figura 21 - Página inicial do Guia do SIDD

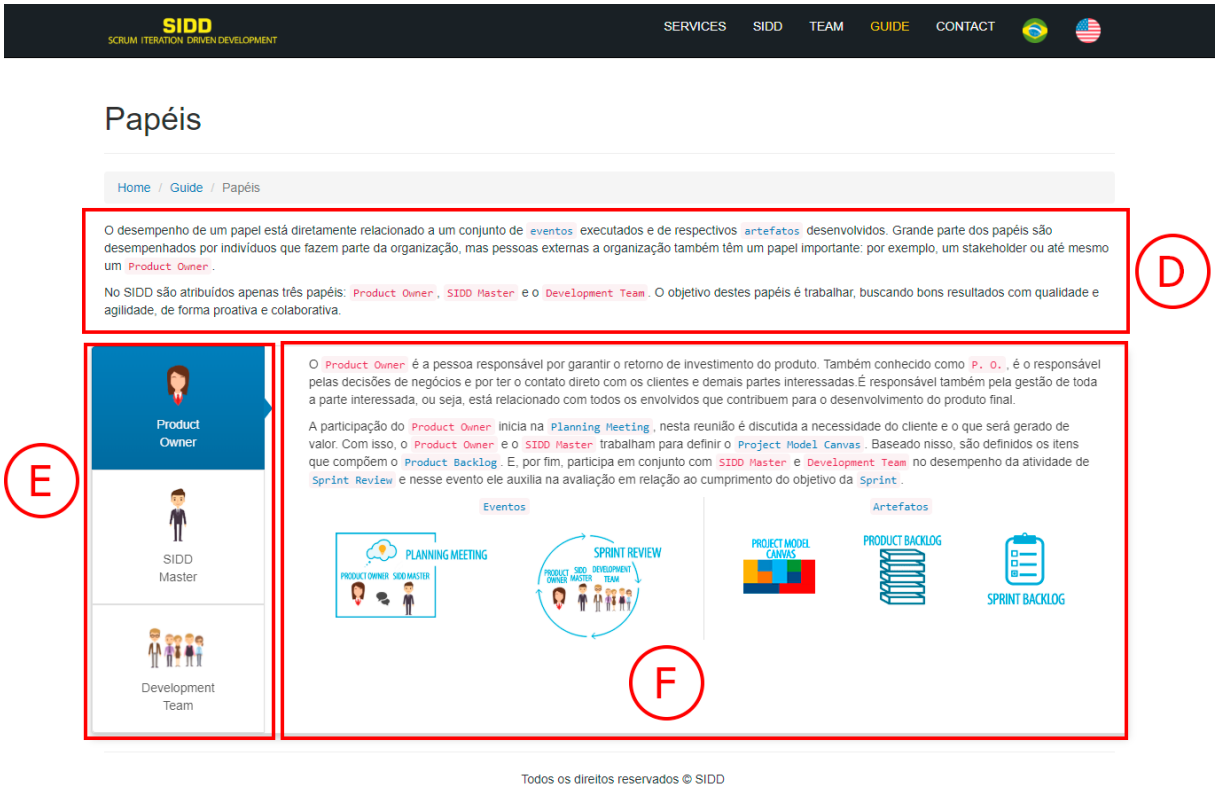


Na Figura 22 é apresentada a descrição dos papéis e como esses são desempenhados no processo. Nessa página são apresentados os papéis de *Product Owner*, SIDD *Master* e *Development Team*, que juntos formam o SIDD *Team*. A primeira área da página (D) descreve as atividades relacionadas a um papel durante a execução do processo de desenvolvimento. Além disso, alguns itens presentes no texto são links que podem ser utilizados para acessar mais informações relacionadas a eles. Como auxílio, é apresentado de maneira detalhada cada papel do SIDD *Team*.

A descrição disposta na página permite uma navegação entre os papéis por meio de um grupo de abas (E) que contém como conteúdo os papéis de *Product Owner*, SIDD *Master* e *Development Team*. Cada papel compõe-se de uma descrição detalhada (F), além de

apresentar as relações existentes entre ele, os eventos realizados e os artefatos criados durante o processo. A descrição é composta de links relacionados sobre a execução do processo e ilustrações para apresentar os detalhes envolvidos em cada evento e os artefatos que estão diretamente relacionados a determinado papel.

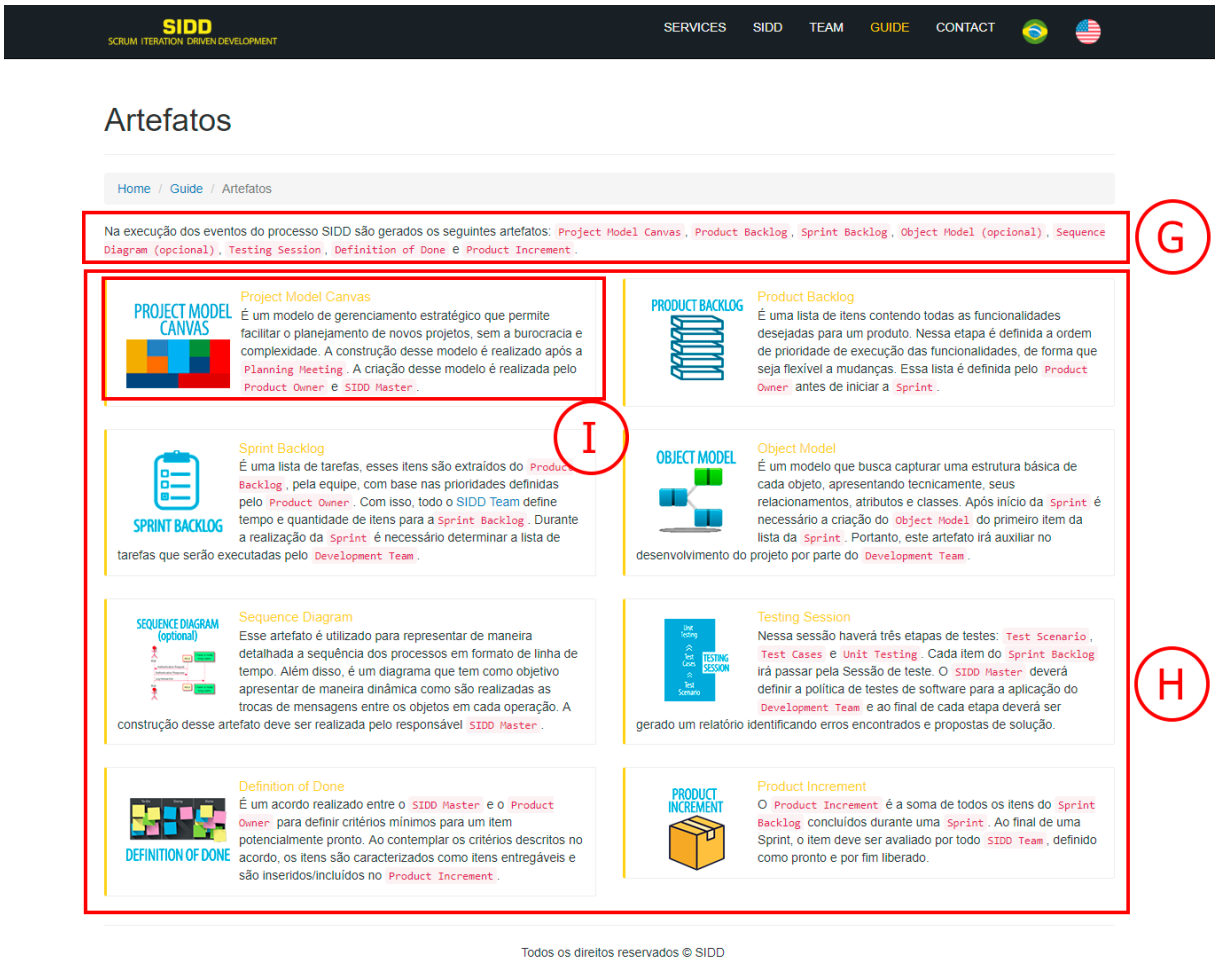
Figura 22 - Página de Apresentação dos Papéis



Posteriormente, na Figura 23 são apresentados os Artefatos que são executados pelos papéis durante os eventos do SIDD. Na execução dos eventos do processo SIDD são gerados os seguintes artefatos: *Project Model Canvas*, *Product Backlog*, *Sprint Backlog*, *Object Model (opcional)*, *Sequence Diagram (opcional)*, *Testing Session*, *Definition of Done* e *Product Increment*. A estrutura da página é dividida em três áreas, sendo elas: G, H e I.

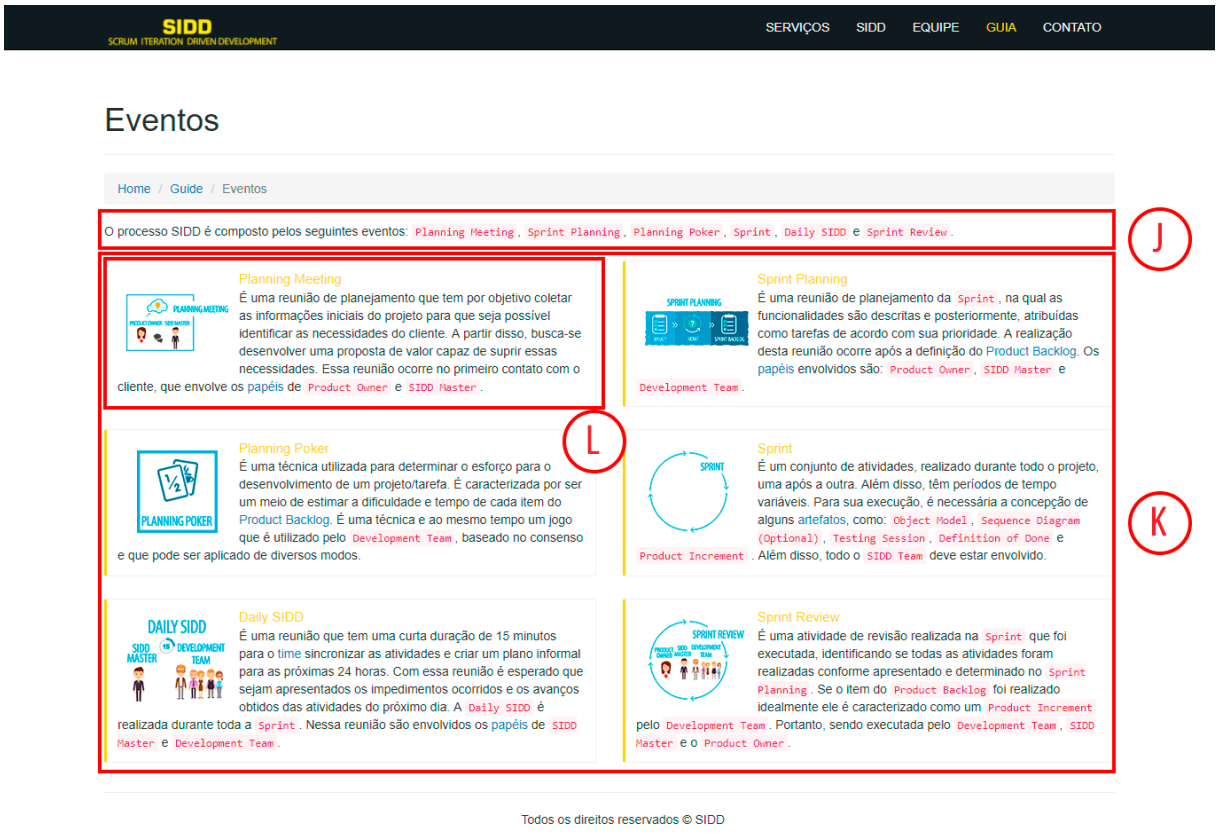
A primeira área da página (G) apresenta os artefatos que o processo SIDD possui. Posteriormente, é apresentada uma listagem (H) com todos os artefatos que são criados durante a realização do processo. Por fim, a última área (I) é caracterizada por descrever um dos artefatos construídos durante o processo de desenvolvimento. Portanto, essa estrutura (I) é utilizada como modelo para os demais artefatos, a fim de ilustrá-los e descrever detalhes sobre sua elaboração.

Figura 23 - Página de Apresentação dos Artefatos



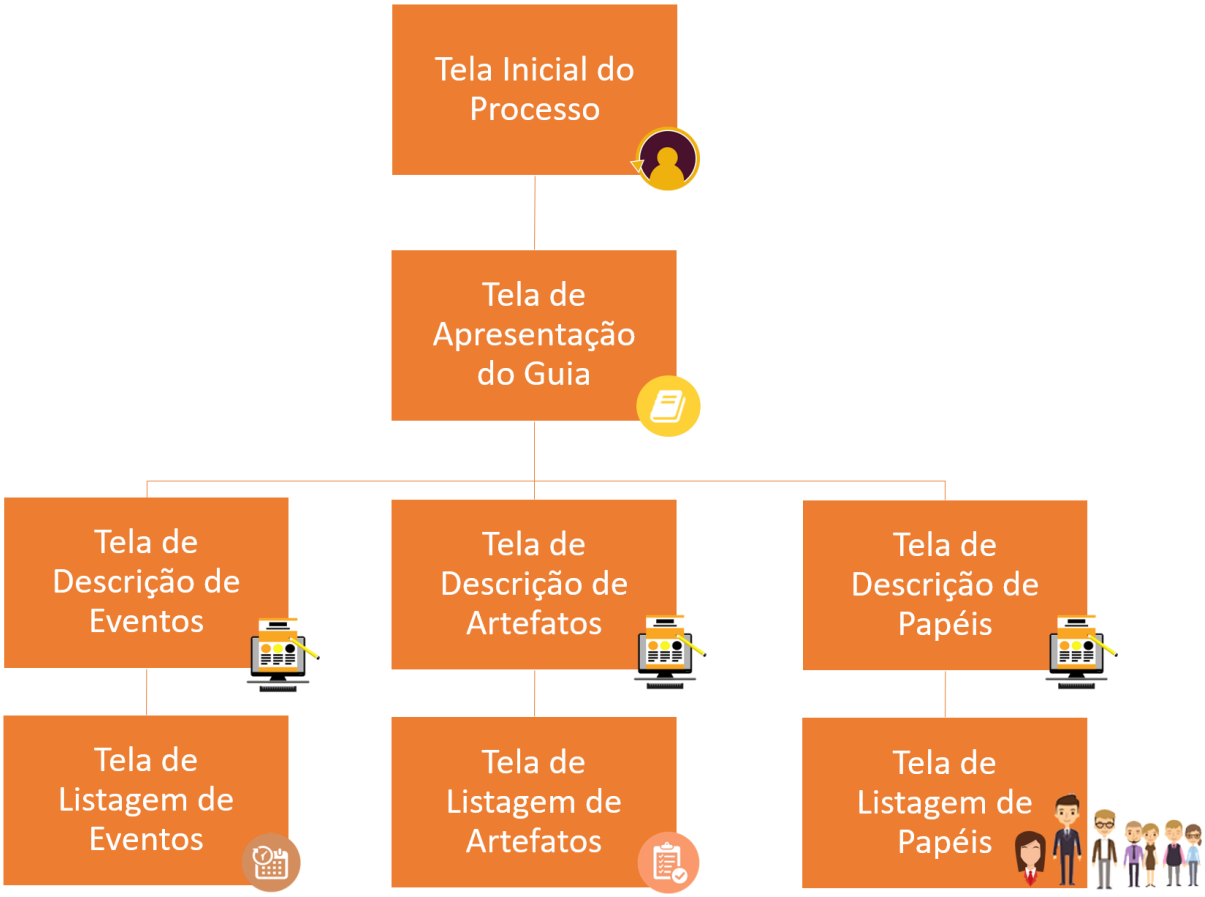
Por fim, na Figura 24 são apresentados os eventos executados durante o processo SIDD. Esse processo é composto pelos seguintes eventos: *Planning Meeting*, *Sprint Planning*, *Planning Poker* (opcional), *Sprint*, *Daily* SIDD e *Sprint Review*. Com isso, segue como padrão a estrutura da página de artefatos, evidenciando as principais áreas de apresentação do evento. Desse modo, os eventos contidos durante a execução do processo são detalhados, ilustrando e descrevendo a sua execução.

Figura 24 - Página de Apresentação dos Eventos



Para apresentar o nível de acesso das áreas mais importantes do Guia do *Scrum Iteration Driven Development* é apresentado um diagrama de navegação do site do processo, como é ilustrado na Figura 25.

Figura 25 - Diagrama de Navegação do Site do SIDD



O principal objetivo do ambiente desenvolvido foi apresentá-lo como um guia do processo, apresentando as informações necessárias a fim de auxiliar gestores de projetos na utilização do processo de desenvolvimento e gerenciamento de *software* em seus projetos.

5 CONSIDERAÇÕES FINAIS

Na análise dos resultados da pesquisa quali-quantitativa realizada com empresas palmenses em abril de 2017 por Ribeiro (2017), foi possível observar que as empresas que trabalhavam com a prática de desenvolvimento ágil não necessariamente atendiam a todos os princípios de uma metodologia ágil, pois grande parte de sua utilização era feita de maneira parcial. Além disso, percebeu-se que a utilização do Scrum como metodologia era base nas empresas de desenvolvimento de *software* da região. Deste modo, essa pesquisa teve por objetivo questionar os entrevistados sobre a utilização de práticas de desenvolvimento de *software* e realizar uma análise das informações obtidas.

Dessa maneira, a proposta de um novo processo ágil capaz de direcionar as atividades de desenvolvimento e gerenciamento, além de levar em consideração o contexto da empresa, pode contribuir para que as empresas sigam os princípios desse processo em sua totalidade, sem depender necessariamente de técnicas complementares.

Portanto, além de explanar sobre os conceitos de metodologia ágil e as etapas do processo Scrum, este trabalho consistiu em modelar um processo ágil fundamentado no Scrum que permita o gerenciamento e desenvolvimento de *software*. A partir disso foi modelado o processo Scrum *Iteration Driven Development*, um processo ágil, com o objetivo de auxiliar na gestão e o planejamento de projetos de *software*.

Dessa forma, a utilização do processo SIDD conta com pontos positivos no auxílio da gestão de atividades de gerenciamento de projetos, como a criação de artefatos específicos que auxiliam no gerenciamento, como o *Project Model Canvas* e no desenvolvimento do *software*, como Object Model (opcional), *Sequence Diagram* e *Testing Session*. Porém, o processo tende a perder em agilidade diante de outros processos ágeis, como o Scrum, pois é um processo mais tradicional, pelo fato de conter a execução de alguns artefatos a mais que o processo base Scrum, como o *Project Model Canvas*, *Object Model* (opcional), *Sequence Diagram* (opcional) e *Testing Session*.

Neste trabalho foi proposto como base a utilização da metodologia Scrum. Nessa utilização, algumas considerações foram realizadas para a concepção do processo SIDD no que se refere à execução de determinados eventos e artefatos. No SIDD, o evento de *Planning Poker* é considerado como opcional no andamento do processo, portanto a sua não utilização não compromete a essência do processo. A fim de fornecer uma maior capacidade de gerência ao processo, foi adicionado em seu ciclo o artefato de *Project Model Canvas*. Este artefato

auxilia no gerenciamento do projeto porque permite um planejamento estratégico e sem muita complexidade. E para fornecer uma maior capacidade de desenvolvimento no processo, foram adicionados os artefatos de *Object Model* e *Sequence Diagram*. Estes artefatos auxiliam no desenvolvimento do projeto porque são artefatos que visam capturar informações básicas da estrutura e do fluxo de funcionamento do projeto a fim de guiar no desenvolvimento do projeto. No entanto, estes artefatos são considerados como opcionais no processo SIDD, permitindo que os usuários possam optar por uma versão mais ágil do processo. Estes artefatos citados foram propostos com o intuito de melhorar o processo tanto em seu eixo gerencial, como também auxiliar nas etapas de desenvolvimento do projeto.

Durante o desenvolvimento deste trabalho foram encontradas dificuldades relacionadas a análise do processo de desenvolvimento ágil Scrum para a identificação de práticas adequadas ao cenário das empresas palmenses participantes da pesquisa quali-quantitativa em 2017 por Ribeiro (2017). Como também, a modelagem de um processo que atendesse as necessidades de modo a suprir os requisitos da metodologia já abordados na pesquisa. Por fim, definir as etapas do processo de desenvolvimento e gerenciamento de *software*.

O presente trabalho tem como trabalhos futuros a implantação do processo SIDD num departamento desenvolvimento de *software* em uma das empresas palmenses participantes da pesquisa quali-quantitativa. Posteriormente, testar suas etapas de modo a verificar se a abordagem escolhida na modelagem do novo processo atendeu as necessidades da empresa. Com isso, realizar análise da aplicabilidade do processo com base numa coleta de informações por meio de uma entrevista com as empresas que participaram da pesquisa quali-quantitativa. Por fim, investigar os principais problemas abordados na pesquisa realizada por Ribeiro (2017) com a proposta de adequação do novo processo, a fim de verificar se houve uma melhoria no processo de desenvolvimento de *software*.

REFERÊNCIAS

PRESSMAN, Roger S. **Engenharia de Software: uma abordagem profissional**; tradução Ariovaldo Griesi, Mário Moro Fecchio. 7.ed. Porto Alegre: AMGH Editora Ltd., 2011, 930 p.

AWAD, M.A.**A Comparison between Agile and Traditional Software Development Methodologies**. University of Western Australia, 2005, 77 p.

STOICA, M.; MIRCEA, M.; GHILIC-MICU, B.. **Software Development: Agile vs. Traditional**. Bucharest University of Economic Studies, Romania, v. 17, n. 4, 2013, 76 p.

SABBAGH, R. **Scrum: Gestão ágil para projetos de sucesso**. São Paulo: Casa do Código, 2013, 280 p.

ALLIANCE, Scrum. **Scrum Values**. Disponível em: <https://www.scrumalliance.org/why-scrum/core-scrum-values-roles>. Acesso em: 25 ago. 2017.

ALLIANCE, Agile. **12 Principles Behind the Agile Manifesto**. Disponível em: <https://www.agilealliance.org/agile101/12-principles-behind-the-agile-manifesto/>. Acesso em: 26 ago. 2017.

SCHWABER, Ken; WEST, Dave. **Scrum.org.** Disponível em: <https://www.scrum.org/resources/what-is-scrum>. Acesso em: 26 ago. 2017.

RIBEIRO, Tayse V. **Pesquisa Quantiqualitativa sobre Processos de Desenvolvimento de Software Utilizados por Empresas de Palmas-TO**. Tocantins: Palmas, 2017, 86 p.

COSTA, Carlos A.**A Aplicação da Linguagem de Modelagem Unificada (UML) para o Suporte ao Projeto de Sistemas Computacionais dentro de um Modelo de Referência**. Departamento de Engenharia Mecânica Universidade de Caxias do Sul (UCS). Caxias do Sul – RS, v.8, n.1, p.19-36, abr. 2001.

W3C. **HTML** "HTML 5". 2017. Disponível em: <https://www.w3.org/html/>. Acesso em: setembro de 2017.

W3C. **CSS** "Página inicial do Cascading Style Sheets". 2017. Disponível em: <https://www.w3.org/css/>. Acesso em: setembro de 2017.

W3Schools.**Tutorial CSS**. 2017. Disponível em: <https://www.w3schools.com/css/>. Acesso em: setembro de 2017.

JQuery. **JQuery write less, do more**. 2017. Disponível em: <https://jquery.com/>. Acesso em: setembro de 2017.

W3Schools.**jQuery Tutorial**. 2017. Disponível em: < https://www.w3schools.com/jquery/>. Acesso em: setembro de 2017.

Tutorials Point. **Behavior Driven Development Tutorial**. Tutorials Point Simply Easy Learning, 2016, 36 p.

SIEGEL, J. OMG UML. **Introduction to OMG's unified modeling language™ (UML®)**. Julho, 2005. Disponível em: < http://www.uml.org/what-is-uml.htm >. Acesso em: setembro de 2017.

LIBARD, Paula L. O.; BARBOSA, Vladimir. **Métodos Ágeis**. Junho, 2010. Disponível em: <www.ft.unicamp.br/liag/Gerenciamento/monografias/monogafia_metodos_ageis.pdf>. Acesso em: 26 ago. 2017.

NORTH, Dan. **Behavior Modification: the evolution of behavior-driven**. Março, 2006. Disponível em: <https://www.stickyminds.com/sites/default/files/magazine/file/2012/XDD10836filelistfilename1_0.pdf>. Acesso em: setembro de 2017.

AUDY, Jorge. **Scrum 360. Um Guia Completo e Prático de Agilidade**; São Paulo: Casa do Código, 2015, 163 p.

BARTIÉ, Alexandre. **Garantia de Qualidade de Software: adquirindo maturidade organizacional**;13ª.ed. Rio de Janeiro: Elsevier Editoria Ltd., 2002, 930 p.

CHIAVEGATTO, R.; PINHEIRO, V.; VEIRA, Andréia F.; CLINEY, J,; OLIVEIRA, Erbett H.; BARROSO, E.; AMORIM, A.; CONTE, T., **Especificação e Automação Colaborativas de Testes utilizando a técnica BDD**. Disponível em: <http://www.lbd.dcc.ufmg.br/colecoes/sbqs/2013/028.pdf >. Acesso em: setembro de 2017.

OSTERWALDER, Alexander.; PIGNEUR, Yves.**Business Model Generation - Inovação em Modelos de Negócios**. 2011. Disponível em: <http://brazil.enactusglobal.org/wp-content/uploads/sites/2/2017/01/Business-Model-Generation.pdf>. Acesso em: setembro de 2017.

FARAHMANDIAN, Vahid. **TDD vs. ATDD vs. BDD**. Novembro, 2016. Disponível em: <https://www.linkedin.com/pulse/tdd-vs-atdd-bdd-vahid-farahmandian/>. Acesso em: setembro de 2017.

LEITÃO, Michele de Vasconcelos. **Aplicação de Scrum em Ambiente de Desenvolvimento de Software Educativo**. Junho, 2010. Disponível em: <http://tcc.ecomp.poli.br/20101/TCC_final_Michele.pdf>. Acesso em: setembro de 2017.

SCHWABER, K.; SUTHERLAND, J., **Guia do Scrum™ - Um guia definitivo para o Scrum: As regras do jogo**. Scrum.Org and ScrumInc., 2013, 19 p.

RAD, N.; TURLEY, F., **The Scrum Master Training Manual - A Guide to Passing the Professional Scrum Master (PSM) Exam - Version 1.2**. Management Plaza, 2013, 86 p.

HASS, A. M. J. **Guide to Advanced Software Testing.** Artech House. 2008.

LAWANNA, A. **The Theory of Software Testing**. Faculty of Science and Technology, Bangkok, Thailand. 2012.