DOI: 10.5748/9788599693148-15CONTECSI/PS-5764

EXSCRUM – A Software Development Process Based on Practices Included in Agile Management and Engineering Methods

José Augusto de Sena Quaresma, 0000-0002-1021-8359, (Universidade Federal do Pará, Pará, Brasil), augustoquaresma@ufpa.br
Emanuel Amoras Rodrigues, 0000-0003-2551-4393, (Universidade Federal do Amapá, Amapá, Brasil), emanorodrigues@gmail.com
Igor Ernesto Ferreira Costa, 0000-0002-2503-4077, (Universidade Federal do Pará, Pará, Brasil), iggor16@gmail.com
Hugo Riviere Silva Moraes, 0000-0001-7877-3992, (Universidade Federal do Pará, Pará, Brasil), hugo.moraes@tucurui.ufpa.com

Colaboradores:
Alexandre Farias Baía, 0000-0002-3844-8148, (Universidade Federal do Pará, Pará, Brasil), alexfarb@gmail.com
George Tassiano Melo Pereira, 0000-0003-4306-447X, (Universidade Federal do Pará, Pará, Brasil), george.melo7@gmail.com
Antonilson da Silva Alcântara, 0000-0001-8024-0220, (Universidade Federal do Pará, Pará, Brasil), antonilsonalcantara@gmail.com
Sandro Ronaldo Bezerra Oliveira, 0000-0002-8929-5145, (Universidade Federal do Pará, Pará, Brasil), sbro@ufpa.br

ABSTRACT: The process for software development is one of the most important for the software engineering area. It was identified as research trend agile methodologies for software development, but a case that agglutinates the best practices of today's agile methods has not yet been found in the literature. In this way, this work addresses the creation of an integrated process for software development with a focus on engineering management, based on the Scrum method and with some subprocesses from the XP, TDD and FDD methodologies, in order to aggregate the best characteristics of the methods agile in a single process.

Keywords: Software Engineering; Agile Methods; Integrated Process; Software Development Process.

EXSCRUM – Um Processo de Desenvolvimento de Software Baseado em Práticas de Métodos Ágeis de Gestão e Engenharia

RESUMO: O processo para desenvolvimento de software é um dos mais importantes para a área de engenharia de software. Foi identificado como tendência de pesquisa as metodologias ágeis para o desenvolvimento de software, porém ainda não foi encontrado na literatura um caso que aglutinasse as melhores práticas dos métodos ágeis mais procurados atualmente. Diante disso, este trabalho aborda a criação de um processo integrado para desenvolvimento de software com foco em gestão em engenharia, baseado no método Scrum e com alguns subprocessos oriundos das metodologias XP, TDD e FDD, com o objetivo de agregar as melhores características dos métodos ágeis em um único processo.

Palavras-chave: Engenharia de Software; Métodos Ágeis; Processo integrado; Processo de desenvolvimento de software.

Agradecimentos: Este trabalho pertence ao projeto SPIDER/UFPA (*http://www.spider.ufpa.br*).

1. INTRODUÇÃO

Os métodos ágeis vêm desempenhando um papel fundamental no desenvolvimento de software moderno. Mais do que uma alternativa, o uso destes métodos está se consolidando como uma ótima solução para a indústria de Tecnologia da Informação (TI), pois prioriza a comunicação entre as pessoas e o valor que o projeto de software agrega aos *stakeholders* mais do que o cumprimento de prazos, custo e escopo inicialmente definidos.

Segundo o relatório do caso de PRIKLADNICKI *et al.* (2014: 21), projetos bem-sucedidos são aqueles finalizados no prazo, dentro do orçamento e que contemplam todas as funcionalidades originalmente especificadas. Estas definições têm sido questionadas porque existem casos em que projetos são considerados de sucesso, mas não atendem às intenções dos usuários nem agregam valor ao negócio. E, alguns projetos podem ser considerados falhos de acordo com as métricas tradicionais, mas se atenderem bem ao público-alvo conclui-se o êxito dos mesmos (PRIKLADNICKI *et al*., 2014: 22). Desta forma, estes métodos conflitam com o conceito tradicional de sucesso à medida que priorizam valores como: interação entre pessoas, qualidade de software, colaboração com o cliente e resposta ágil à mudanças de escopo.

Não é o simples fato de utilizar métodos ágeis, o qual irá garantir a entrega de software totalmente confiável e atenderá sempre com sucesso os interesses do cliente. A utilização destes métodos possibilita alcançar melhores resultados. Pois, como foi citado, projetos que são tradicionalmente considerados bem-sucedidos muitas vezes não trazem benefícios para o negócio (AZEVEDO, 2008).

Os métodos ágeis atendem projetos de escopo aberto, ou seja preveem mudanças tanto de funcionalidades quanto de atividades. Ao contrário dos métodos tradicionais, os quais são aplicados apenas em situações onde se tem requisitos estáveis e previsíveis. Eles contemplam equipes pequenas, o desenvolvimento rápido é fundamental, priorizando entregáveis de software em curto período de tempo. Entretanto, adotar a abordagem ágil nas empresas preconiza a incorporação dela na cultura organizacional, e necessariamente precisa estar alinhada com os objetivos de negócio (SOARES, 2004a; SOARES, 2004b).

Neste contexto, métodos como *Scrum*, XP – *eXtreme Programming* e FDD – *Feature Driven Development*, podem ser modificados e mesclados para atender as necessidades das organizações no que diz respeito aos processos e a comunicação entre os indivíduos envolvidos no desenvolvimento de software e o cliente.

1.1. Justificativa

Os métodos ágeis consistem em um conjunto de metodologias que servem para acelerar o ritmo dos processos de desenvolvimento de software. Os mais comuns são: Scrum, *eXtreme Programming* (XP), *Test Driven Development* (TDD) e *Feature Driven Development* (FDD). Cada uma destas técnicas elencadas possuem características diferentes entre si, algumas se conflitam em seus princípios, outras se complementam, por exemplo, o XP utiliza como base para o processo de desenvolvimento de testes as práticas descritas pelo método TDD. Enquanto o *Scrum* tem foco na gestão do desenvolvimento, por sua vez o método FDD detalha melhor os procedimentos para descobrir requisitos funcionais. Portanto, este trabalho visa criar um processo ágil que contenha a junção das melhores práticas destas técnicas supracitadas.

A elaboração de um processo unificado que faz uso das melhores práticas de cada um dos quatro métodos supracitados justifica-se pois eles são comumente utilizados nas empresas. Porém, tais empresas raramente conseguem aplicar na íntegra todas as práticas ou princípios de um ou mais métodos ágeis, isto é, quase sempre utilizam apenas algumas

práticas dos métodos desejados ou até mesmo há uma adaptação do método conforme o contexto da empresa para que seja possível alcançar sucesso nos projetos, pois seguir estritamente cada prática de um método ágil em questão requer grande esforço para fazer com que cada pessoa envolvida nos processos já consolidados aceite abertamente as mudanças, além de requerer pessoas bastante qualificadas ou que tenham vasto conhecimento sobre tal método.

Outro fator é a tendência de crescimento da utilização dos quatro métodos ágeis: XP, *Scrum*, TDD e FDD, segundo pesquisa na ferramenta *Google Trends*, o qual mostra que nos últimos cinco anos são os métodos mais pesquisados. Portanto, conclui-se que o interesse pela utilização destes aumentou (GOOGLE TRENDS, 2018). Este fato evidencia a necessidade de haver um processo unificado com as melhores ou principais práticas abordadas por esses métodos em questão.

1.2. Motivação

Este trabalho tem sido motivado por estudos na disciplina de Tecnologia em Processos de Software, ofertada pelo Programa de Pós-Graduação em Ciência da Computação da Universidade Federal do Pará no segundo semestre de 2017. Dentro da disciplina, o professor dividiu a turma em quatro grupos a fim de que cada um pesquisasse sobre um método entre os quatro anteriormente citados. A cada grupo foi proposto desenvolver um seminário e uma dinâmica tratando de seu respectivo método. Ao fim da componente curricular, o docente propôs à turma a unificação dos processos de cada método, o que gerou o processo integrado que está apresentado neste artigo.

Diante disto, observou-se que cada método tem suas particularidades e que utilizados individualmente não possibilitam atender por sua completude as áreas necessárias no desenvolvimento do projeto, como por exemplo, o XP e o TDD não apresentam práticas suficientes para guiar uma boa gestão do desenvolvimento de software ou planejamento, pois prescrevem apenas o jogo do planejamento para tais ações, enquanto o *Scrum* e o FDD não apresentam procedimentos detalhados de como melhor desenvolver o software, havendo possibilidade de utilizar em conjunto outros métodos para que haja complemento de suas práticas.

É importante ressaltar que foram encontrados poucos materiais relatando a aplicação prática dentro de uma organização. Esta dificuldade ficou evidente quando tentou-se encontrar materiais acadêmicos, seja de estudos de caso aplicando os métodos na íntegra ou mesclando as principais práticas dentre eles.

1.3. Objetivo

Nestas circunstâncias este trabalho tem como intuito apresentar uma abordagem prescritiva (processo modelado) abrangendo as práticas mais relevantes dos quatro métodos abordados. Portanto, o objetivo foi definido no sentido de mostrar para as empresas que é possível integrar as principais práticas dos quatro métodos em questão, a fim de solucionar os problemas existentes no gerenciamento e desenvolvimento de projetos de *software*.

1.4. Estrutura

Além desta seção introdutória, este artigo é composto das seguintes seções: a Seção 2 apresenta uma fundamentação teórica sobre métodos ágeis; a apresentação do processo integrado, seção 3; o processo integrado com a sua construção, características, perfis e o contexto são discutidos na seção 4; a avaliação do processo a partir de uma matriz SWOT – *Strengths, Weakness, Opportunities and Threats*, é apresenta na seção 5; e, por fim, na seção 6 as considerações finais do trabalho são discutidas.

2. FUNDAMENTAÇÃO TEÓRICA

Na década de 1980 e início de 90 a metodologia tradicional de desenvolvimento era muito forte. Existia uma visão de que um produto seria de qualidade se seguisse as fases rigorosas e inflexíveis de desenvolvimento, propostas pelo modelo clássico ou cascata. No entanto, as médias e pequenas empresas tinham um overhead muito grande, gastando mais tempo planejando que de fato desenvolvendo e entregando valor para o cliente (PRESSMAN, 2011).

A insatisfação das empresas levou os desenvolvedores de software, na década de 1990 a propor métodos mais ágeis de desenvolvimento. Então, ocorreu o marco conhecido como Manifesto Ágil (Manifesto para Desenvolvimento Ágil de Software, 2016), trazendo o seguinte lema: indivíduos e interações mais que processos e ferramentas; software em funcionamento mais que documentação abrangente; colaboração com o cliente mais que negociação de contratos; e responder a mudanças mais que seguir um plano. O foco em vez de se concentrar nos processos, como acontecia antigamente, passa a ser nas interações entre os envolvidos, no seu comprometimento em entregar valor ao cliente.

Neste contexto de métodos ágeis, vários modelos foram propostos, mas os que serão utilizados neste trabalho serão: *eXtreme Programming* – XP (BECK, 2000), SCRUM (SCHWABER e SUTHERLAND, 2013), *Test Driven Development* – TDD (BECK, 2010). Foram escolhidos esses métodos por serem considerados os mais populares segundo o GOOGLE TRENDS (2018).

O XP (BECK, 2000) é um dos mais conhecidos e amplamente usados entre os métodos ágeis. Aborda o desenvolvimento iterativo e o envolvimento do cliente em níveis extremos. Ele prega algumas boas práticas como: o uso de metáforas, que tem o poder de transmitir ideias complexas de forma simples e clara, devendo ser estabelecidas de acordo com o vocabulário que o cliente está habituado no dia a dia; ideia de refatoração, que é o processo de reorganizar o código fonte de um software para melhorar sua qualidade interna, facilitar a leitura e diminuir o tempo gasto com manutenção; entrega de releases, que são um conjunto de funcionalidades que representa uma pequena versão do sistema com recursos completos e relevantes para o cliente, onde cada release é fragmentado em pequenas partes chamadas iterações; programação em pares, a fim de produzir um software de qualidade e apoiar o nivelamento de conhecimento da equipe; e o uso de estórias, que são pequenos cartões em que o cliente escreve as funcionalidades que deseja no sistema.

Outro método ágil usado neste trabalho foi o SCRUM (SCHWABER e SUTHERLAND, 2013), também muito utilizado nas organizações para gerenciamento ágil de projetos. De forma geral, o SCRUM tem três partes principais em seu modelo: papéis, cerimônias e artefatos. Essas três partes são utilizadas no que é chamado de ciclo de desenvolvimento, que é a Sprint. Cada Sprint possui suas fases e utilizam papéis, cerimônias e artefatos para alcançar seu objetivo final que foi definido com o cliente.

O TDD é uma técnica que consiste no desenvolvimento de testes de unidade antes da criação do código. Estes testes servirão de guia durante o processo de desenvolvimento. Inicialmente, esta técnica foi uma prática proposta pelo método XP, no entanto, tem sido utilizada por outras metodologias ágeis. BECK (2010) afirma que o desenvolvimento orientado por testes (TDD) é uma abordagem evolutiva para o desenvolvimento, onde deve-se escrever um teste antes de escrever um código de produção suficiente para realizar esse teste e realizar a sua refatoração.

Segundo MILLER (2004), o TDD reivindica um desenvolvimento incremental do código que inicia com testes, incluindo frequentemente testes de regressão. Uma das regras do TDD é a seguinte: “se você não consegue escrever um teste para aquilo que você está codificando, então você nem deveria pensar sobre codificar”.

Por fim, o FDD é um modelo prático de processo para a engenharia de software orientado a objeto (RETAMAL, 2011). Possui a característica de desenvolvimento que as funções valorizadas pelo cliente devem ser passíveis de entrega em duas semanas ou menos. A *feature* é o principal objeto desse método, sendo que a *feature* representa a funcionalidade ou tarefa a ser desenvolvida no produto. No modelo do FDD pode-se destacar a prática Detalhar por *feature*, cujo propósito é, para cada *feature*, identificar e detalhar qual a melhor solução para o projeto ou design da *feature*.

3. RESULTADOS DA PESQUISA

Nesta seção está disposta a apresentação inicial do processo integrado, bem como a denominação que os autores e colaboradores atribuíram a ele, ExScrum.

3.1. ExScrum

O processo *ExScrum* tem como base o modelo clássico do *Scrum*. O processo foi modificado e adaptado a fim de manter uma coesão entre as práticas que foram acrescentadas. Tais práticas têm origem nos seguintes métodos: FDD (Construção do Modelo Abrangente), XP (Planejamento da *Release*) e TDD (Implementar Teste Unitário e Implementar Teste de Integração). A Figura 1 exibe o processo macro do *ExScrum*.

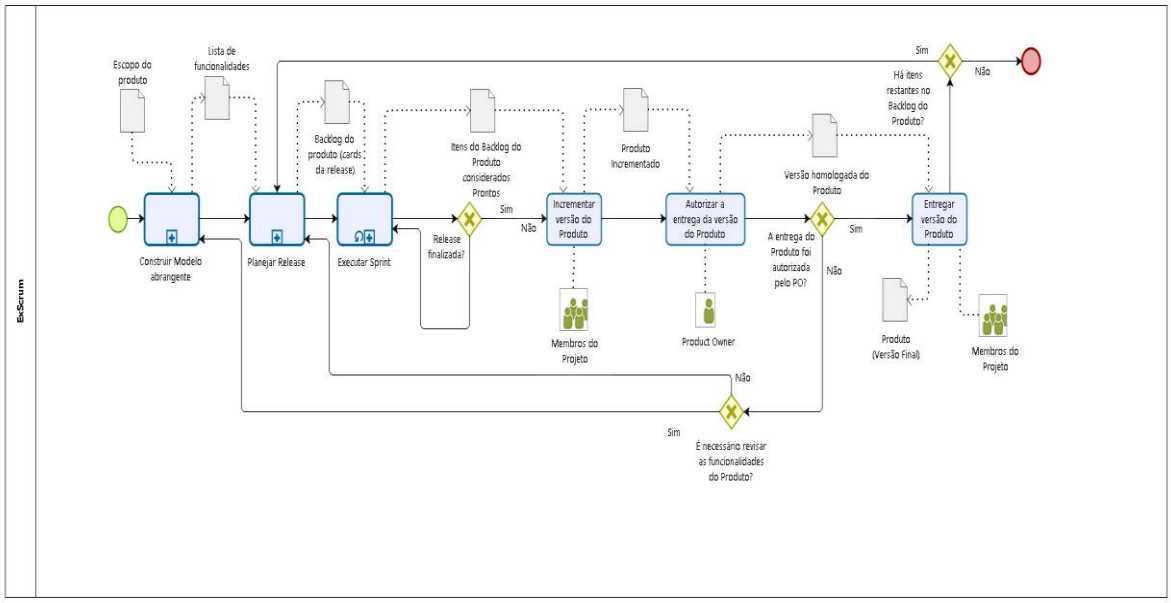

Figura1 - Modelo Macro do *ExScrum*.
Fonte: Próprio Autores - 2018

O processo inicia-se com o artefato que contém o escopo do produto, e a partir dele é realizado o primeiro subprocesso, Construir Modelo Abrangente. Este modelo do processo encontra-se a seguir na Figura 2.

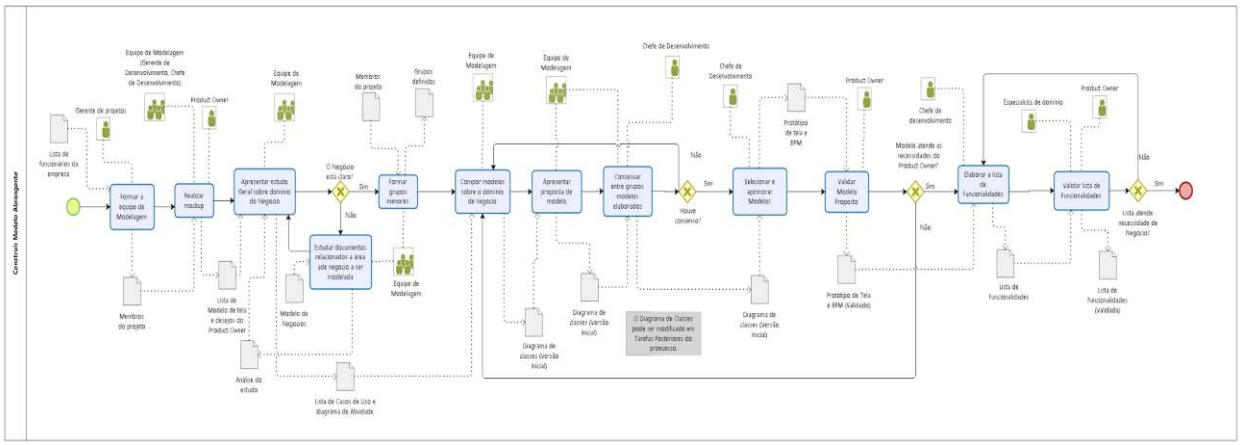

Figura 2 - Sub-processo Construir Modelo Abrangente.
Fonte: Próprio Autores - 2018

4. Processo Integrado

Uma vez montado o primeiro escopo do produto, é necessário organizar as ideias contidas em sua concepção, para isso o Modelo Abrangente faz-se presente. É realizado um levantamento dos funcionários disponíveis para trabalhar naquele projeto de desenvolvimento, tarefa esta realizada pelo Gerente de Projetos, obtendo-se assim os membros disponíveis para aquele trabalho.

O próximo passo é reunir a equipe de modelagem com o *Product Owner* e desenvolver o *mockup* do produto, no qual consiste em um protótipo daquilo que se pretende desenvolver e uma vez pronto, tem-se o modelo de tela do produto com as ideias trazidas pelo *Product Owner*. Com a finalização desta etapa, a equipe de modelagem realiza um estudo de domínio de negócio para observar a clareza do produto a ser desenvolvido perante o mercado. Se estiver claro, prossegue-se o sub-processo, caso contrário, estuda-se novamente sobre o domínio de negócio. Esta etapa é realizada novamente até que atinja o nível de clareza necessário.

Então, divide-se os membros do projeto em equipes menores e define-se grupos para cada tarefa necessária para a modelagem do produto, realizando a seguir a composição dos modelos de negócio, originando-se a partir desta etapa o Diagrama de Classes Inicial. Este deverá passar pelas seguintes etapas de validação: Apresentar Proposta de Modelo e Consensuar entre os grupos os modelos elaborados. Se houver o consenso, segue-se o subprocesso, se não, retorna-se à composição dos modelos. Esta etapa é exibida na Figura 3.

Com os modelos validados pela equipe de modelagem, o Chefe de Desenvolvimento recebe os Diagramas de Classe devidamente validados e os aprimora, gerando o protótipo de tela e o BPM – *Business Process Modeling* do produto. A seguir, o *Product Owner* inspeciona estes artefatos e realiza a validação. Caso não estejam de acordo com o desejo do *Product Owner*, retorna-se para a composição dos modelos sobre o domínio de negócio, se estiver de acordo, o Chefe de Desenvolvimento elabora uma Lista de Funcionalidades. Esta lista é validada pelo Especialista de Domínio e o *Product Owner*, se estiver dentro do que foi pedido pelo *Product Owner*, este subprocesso encerra-se e passa-se para o Planejamento da *Release*, caso não esteja, retorna-se para a elaboração da Lista de Funcionalidades.

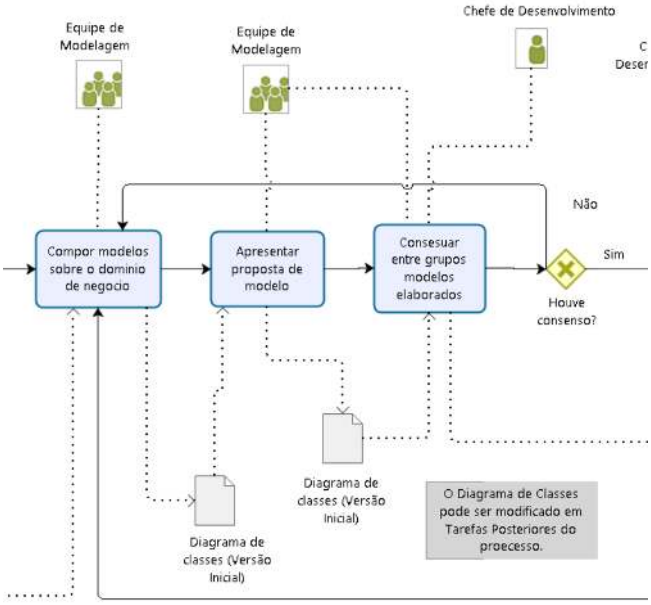

Figura 3 - Etapa de Validação do Diagrama de Classes Inicial
Fonte: Próprio Autores - 2018

4.1. Planejar *Release*

Segundo BOURQUE e FAIRLEY (2014), o planejamento da *release* define quando os vários conjuntos de funcionalidades ou de produtos utilizáveis serão entregues ao cliente. Tem como principal objetivo criar um cronograma de planejamento da *release* e permitir que o Time *Scrum* tenha uma visão geral do cronograma da *release* e entrega para o produto que estão desenvolvendo, para que possam ajustar-se de acordo com as expectativas do Dono do Produto e dos *stakeholders* relevantes (principalmente do patrocinador do projeto). A estratégia para os lançamentos do produto depende da política de cada organização, as principais saídas do planejamento da *release* são: o cronograma de planejamento da *release*, onde contém quais entregáveis devem ser liberados para os clientes com os planos e datas de lançamentos; a duração das *Sprints*, as *sprints* podem ter *time-box* que variam de duas a quatro semanas; os Clientes alvos para as *releases*, nem todos os entregáveis são para todos os *stakeholders*, podendo ter lançamentos para *stakeholders* específicos; o *backlog* do produto priorizado e refinado, podendo ter o detalhamento do *backlog* do produto mantendo a saúde do *backlog*, onde as estórias podem ser esclarecidas para o time de desenvolvimento.

O processo de planejamento da *release*, exibido na Figura 4, foi desenvolvido buscando a integração entre os quatros modelos. Assim, tem-se como ponto de partida o processo Escrever Estórias, feita pelos membros do projeto, este processo tem como entrada a Lista de Funcionalidades que foi gerada na etapa anterior e como saída as Estórias de Usuários. Se tudo ocorrer bem, o fluxo segue para o próximo processo, Calcular Esforço, processo de responsabilidade do time de desenvolvimento que tem como entrada as estórias criadas na etapa anterior e como saída os *Cards* com esforços. O processo conta com uma condicional, ou seja, caso as estórias forem grandes passa para o próximo processo que é Quebrar em Estórias Menores e em seguida vai para o processo Reescrever Estórias para pode voltar ao processo de Calcular Esforço. Caso esteja tudo correto, o fluxo segue para o próximo processo, Escolher Critério de Ordenação, realizado

pelos Membros do projeto, tendo como entrada a Forma de ordenação e como saída a Forma de ordenação escolhida.

O fluxo segue para o próximo processo, Ordenar por Critérios Escolhidos, que tem como entrada os *Cards* com esforço e a Forma de ordenação escolhida, onde os responsáveis por este processo são os Membros do Projeto, e tem-se como saída os *Cards* priorizados. O fluxo segue para o próximo processo, Selecionar o Escopo, de responsabilidade dos Membros do projeto, que tem como entrada os *Cards* Priorizados, que foram gerados no processo anterior e sua saída é o *Backlog* do produto (*Cards* da *release*), que é entrada para o próximo processo, Definir Quantidade de *Sprints*. Este último processo é de responsabilidade dos Membros do projeto, tendo como saída os *Cards* por *Sprint*, que é a entrada direta para o processo seguinte, Analisar Ritmo de Desenvolvimento, de responsabilidade do Time de desenvolvimento e que tem como entrada o Desempenho do time e gera como saída o *Backlog* do produto (*Cards* de *release*). Este último processo conta com uma condicional com a seguinte pergunta: O ritmo está adequado à quantidade de estórias da *Sprint*? Caso não esteja, ele volta para o processo Escolher Critério de Ordenação e segue o fluxo a partir deste processo, ou caso ele esteja de acordo, ele entra em uma outra condicional com a seguinte pergunta: Existe uma nova estória? Caso não exista, o fluxo é encerrado, e caso positivo, o fluxo retorna para o processo inicial de Escrever Estórias.

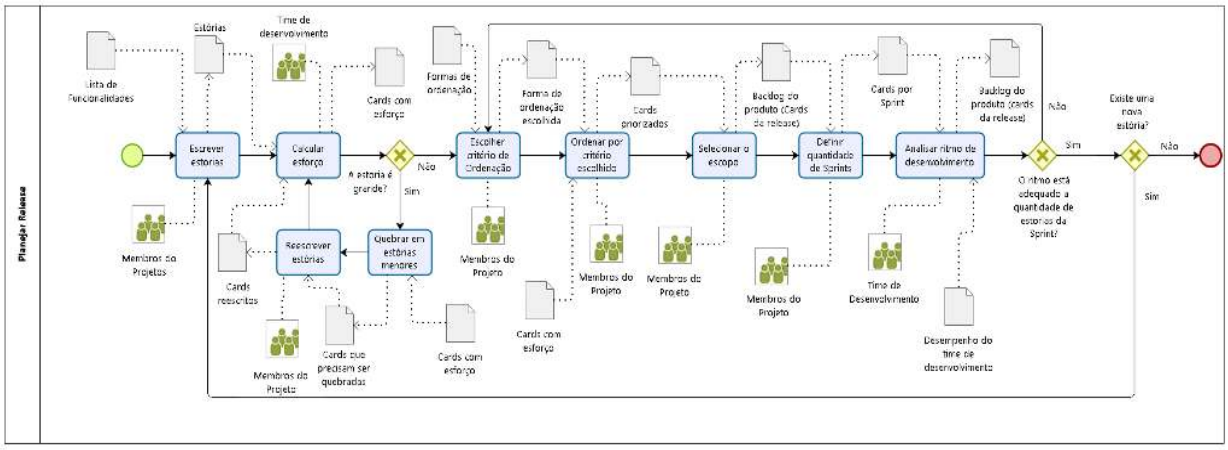

Figura 4 - Modelo do Subprocesso Planejar *Release*.
Fonte: Próprio Autores - 2018

4.2. Executar *Sprint*

A etapa de execução da *Sprint*, exibida por completo na Figura 5, é oriunda do modelo clássico do *Scrum*, onde o subprocesso inicia-se com o *Backlog* Priorizado do Produto, originado do subprocesso Planejar *Release*. Este artefato alimenta a etapa Planejar *Sprint*, onde os membros do projeto definem a duração da *Sprint* e quais itens do *Backlog* fazem parte desta, gerando-se assim o *Backlog* da *Sprint*. Nesta etapa, ainda podem ser adicionados os itens prontos do *Backlog* como referências para um melhor planejamento da *sprint* atual. Além disso, as melhorias das *sprints* anteriormente realizadas são adicionadas como novos *cards* de atividades a serem desenvolvidas.

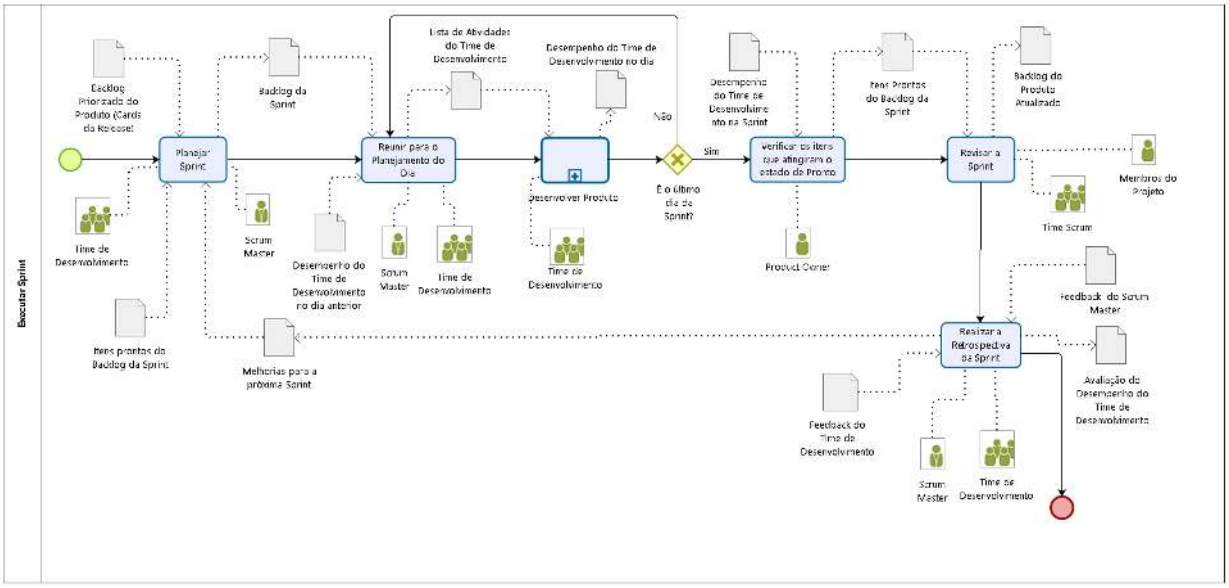

Figura 5 - Modelo de Execução da *Sprint*.
Fonte: Próprio Autores - 2018

Uma vez planejada a *sprint*, iniciam-se os trabalhos com a reunião diária da *sprint*. É realizada uma reunião informal, que tem duração de no máximo quinze minutos. A reunião é assegurada pelo *Scrum Master* e acontece em pé, entre o mesmo e o Time de Desenvolvimento. São colocadas em pauta as seguintes perguntas, onde cada membro do time deve responder: “o que eu fiz ontem?”; “o que eu farei hoje?”; e “quais os impedimentos estou tendo?”. Uma boa prática para alcançar o objetivo da *sprint* é planejar para que ela seja realizada no mesmo local e no mesmo horário. Na ocasião são planejadas as próximas vinte e quatro horas e observado o que pode ser melhorado em relação ao dia anterior. Com isso é gerada a Lista de Atividades do Time de Desenvolvimento para aquele dia. A partir disto, inicia-se o subprocesso Desenvolver Produto**,** que aplica o método TDD.

4.2.1. Desenvolver Produto

O subprocesso é iniciado com a atividade denominada Selecionar Funcionalidades, onde os Membros do Projeto utilizam a Lista de Atividades criada na reunião diária da *sprint* para selecionar a funcionalidade que será implementada no dia. Após a seleção, o time faz o levantamento dos requisitos necessários para implementá-la e realiza a atividade Criar Cenários de Teste. O artefato de saída desta atividade é a Lista de Cenários, como pode ser observado na Figura 6. Esta lista compõem os cenários de teste criados pelo time.

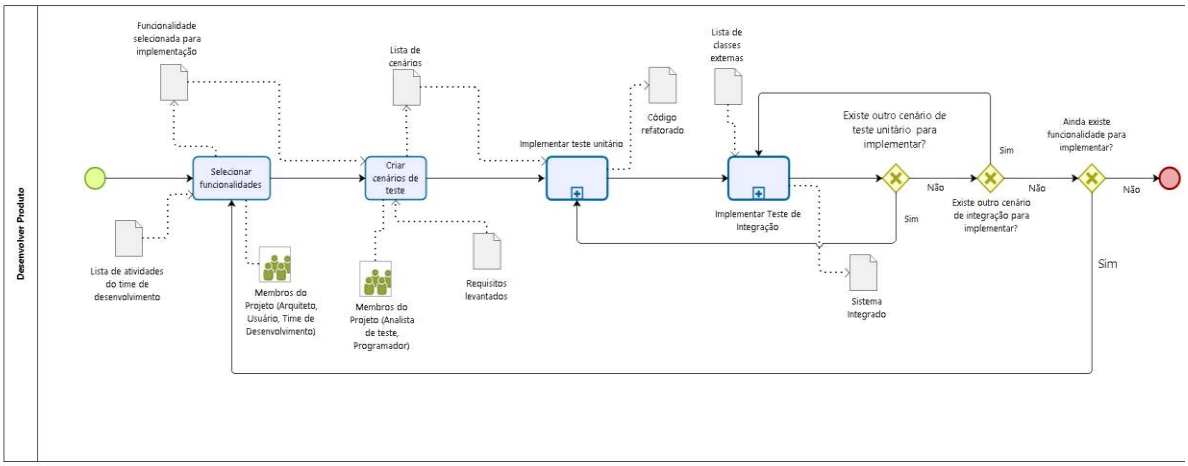

Figura 6 - Parte do processo Desenvolver Produto.
Fonte: Próprio Autores - 2018

O próximo passo é realizar o subprocesso Implementar Teste Unitário. Estes compõem as atividades que guiam a escrita de testes unitários, pode-se observar que a primeira atividade consiste justamente em escrever os testes com base na Lista de Cenários criada anteriormente (ver Figura 7). Dentro do subprocesso observa-se a aplicação do ciclo denominado *red-green-refactor*. Inicialmente, tem-se a atividade de Criar Teste, que consiste em escrever o código do teste unitário. A atividade seguinte, Realizar Teste do Código de Teste, é responsável por realizar os testes com o auxílio de um *framework*, ou seja, ela é executada em pares por representantes do time de desenvolvedores que escrevem o teste mais simples antes de escrever o código de produção e utilizam os métodos do *framework* para realizar as asserções no código. Sendo o teste de caráter inicial, é natural que retorne um resultado negativo, isto é, nesse momento observa-se a fase *red* do ciclo TDD. Logo, é necessário escrever o código de produção para assegurar que o teste vai passar. Esta atividade, que está evidenciada no subprocesso, é executada até que o teste avance para a fase *green*.

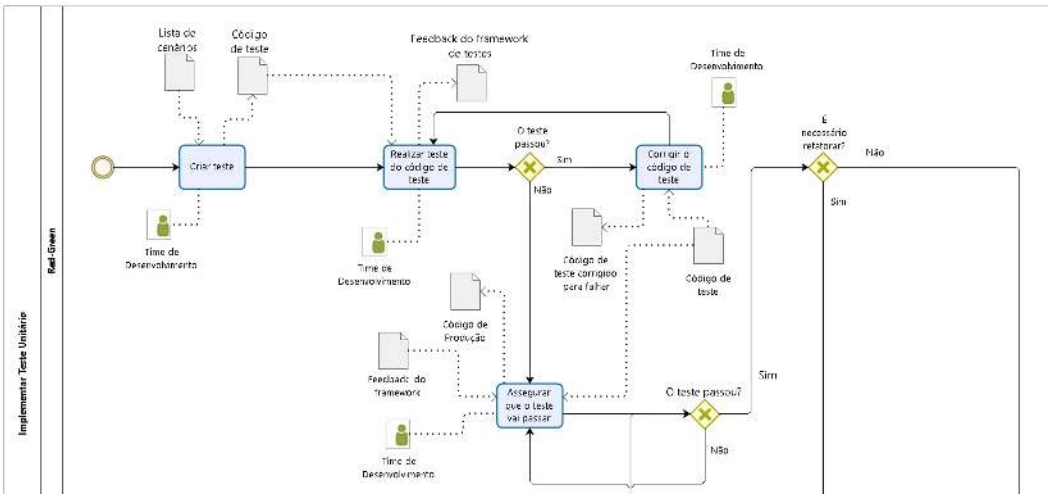

Figura 7 - Fases *red-green* do TDD dentro do subprocesso Implementar Teste Unitário.
Fonte: Próprio Autores - 2018

Após o retorno positivo, é avaliado se o código precisa ser refatorado, então, são aplicadas as boas práticas de refatoração sugeridas pelo método, por exemplo, remover duplicação, escrever o código de maneira legível e com simplicidade, dentre outras. Após cada execução da atividade Refatorar, é necessário executar a atividade Realizar Teste do Código de Produção Refatorado para verificar se o código produzido não regrediu em seu desempenho, isto é, se o que estava funcionando antes passou a apresentar algum tipo de erro no momento atual.

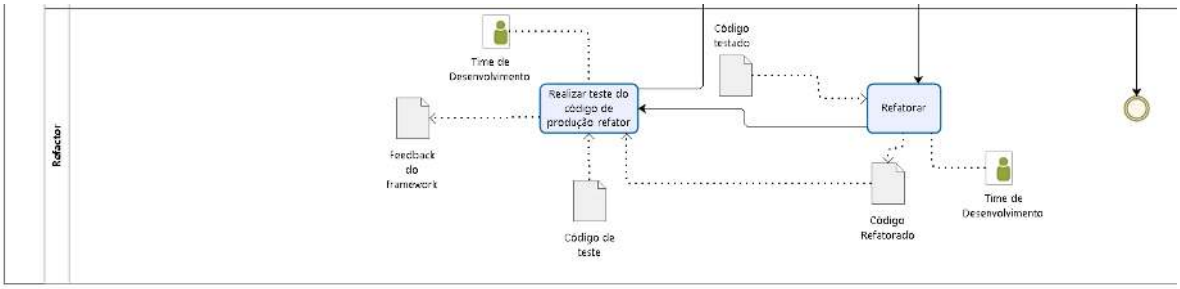

Figura 8 - Fase *refactor* do TDD dentro do subprocesso Implementar Teste Unitário
Fonte: Próprio Autores - 2018

Na Figura 9 tem-se o exemplo prático de um teste escrito em Java, utilizando o *framework* de testes JUnit. A aplicação trata-se de uma pequena parte de um sistema bancário. Na figura, pode-se observar a criação de um cenário de teste através do JUnit, o qual está denotado pelo método *SetUp( )*, onde é instanciado um objeto chamado cliente que pertence a uma classe de mesmo nome. Neste cenário, tem-se a adição de uma conta de um cliente chamado George. Pode-se observar que é criada uma instância da classe Conta e que esta possui três atributos: saldo, agência e número da conta. O método *tearDown( )* é utilizado para destruir os dados do objeto instanciado e liberar recursos no servidor da aplicação.

```
1  package principal;
2  import static org.junit.jupiter.api.Assertions.*;
7
8
9  public class ClienteTest {
10
11     private Cliente cliente;
12
13     @Before
14     public void setUp() throws Exception {
15         cliente = new Cliente("George", new Conta(0.0, 12345, 123456));
16     }
17
18     @After
19     public void tearDown() throws Exception {
20         cliente = null;
21     }
```

Figura 9 - Trecho do código de teste unitário indicando o uso dos métodos do JUnit.
Fonte: Próprio Autores - 2018

Neste cenário de teste é verificado se o valor retornado pelo método *verSaldo( )* é o mesmo que o esperado na conta do cliente George. Portanto, é criado o teste *deveRetornarSaldoAtualDaContaDoCliente( )* e são definidas duas variáveis do tipo

double, saldoAtual e saldoEsperado**,** para realizar a asserção através do método *assertEquals( )*, como pode ser visualizado na Figura 10. É muito importante definir um nome que explique o comportamento esperado pelo teste diante do cenário definido.

```
23      @Test
24      public void deveRetornarSaldoAtualDaContaDoCliente() {
25
26          //executa o metodo de ver saldo
27          double saldoAtual = cliente.verSaldo();
28
29          //define o valor esperado
30          double saldoEsperado = 0.0;
31
32          //testa se o valor recebido é o mesmo do esperado
33          assertEquals(saldoEsperado, saldoAtual);
34      }
35
36  }
37
```

Figura 10 - Trecho do código de teste unitário mostrando a montagem do cenário.
Fonte: Próprio Autores - 2018

Como foi descrito no subprocesso, no primeiro momento o teste ficará vermelho, então faz-se necessário escrever o código de produção. Neste código estão definidas duas classes, Cliente e Conta, que são apresentadas nas Figuras 11 e 12. Pode-se observar que existe uma dependência entre elas, pois na classe Conta é definido o método *getSaldo( )* que é utilizado na classe Cliente.

Todo teste possui três passos: criar um cenário, realizar uma ação e validar o resultado esperado (ANICHE, 2012: 10). Neste exemplo, o cenário montado consiste em verificar se o valor do saldo do cliente é zero, a ação é executada por meio do método *verSaldo( )* e o resultado é validado a partir do retorno fornecido pelo JUnit.

```
1   package principal;
2
3   public class Conta {
4
5       private double saldo;
6       private int numeroConta;
7       private int numeroAgencia;
8
9       public Conta(double saldo, int numeroConta, int numeroAgencia) {
10          this.saldo = saldo;
11          this.numeroConta = numeroConta;
12          this.numeroAgencia = numeroAgencia;
13      }
14
15      public double getSaldo() {
16          return saldo;
17      }
18
19  }
```

Figura 11 -Código da classe Conta escrito em Java.
Fonte: Próprio Autores - 2018

```java
package principal;

public class Cliente {

    private String nome;
    private Conta conta;

    public Cliente(String nome, Conta conta) {
        this.nome = nome;
        this.conta = conta;
    }

    public Conta getConta() {
        return conta;
    }

    public double verSaldo() {
        return conta.getSaldo();
    }

}
```

Figura 12 - Código da classe Cliente escrito em Java.
Fonte: Próprio Autores - 2018

Depois desta atividade, o código é submetido ao teste novamente e ela é executada quantas vezes forem necessárias até que o teste tenha um resultado positivo, isto é, fique verde (ver Figura 13). A partir daí serão realizadas as refatorações, seguidas de testes para cada mudança no código. Nesta etapa existe uma série de boas práticas sugeridas pelo método TDD, porém, detalhar cada uma foge do escopo deste artigo.

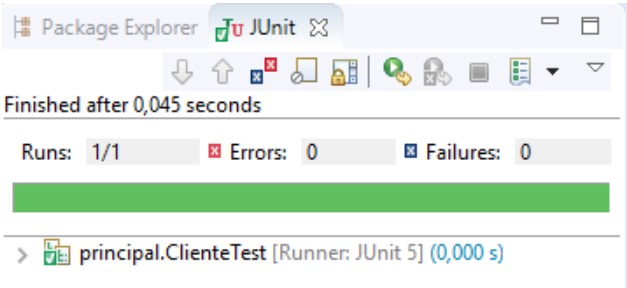

Figura 13 - Retorno do JUnit mostrando a fase *green* do ciclo TDD.
Fonte: Próprio Autores - 2018

4.2.2. Testes de Integração

Os testes de integração asseguram a qualidade e a manutenabilidade do código escrito para servir de interface entre dois pontos da aplicação. São comumente empregados para testar a conexão entre uma classe e um sistema externo. Um exemplo são as classes DAO (*Data Access Object*), que são responsáveis por fazer a conexão da aplicação com bancos de dados.

O subprocesso Implementar Teste de Integração (ver Figura 14) é similar ao de teste unitário, pois inicia com a atividade Criar Cenário de Integração, onde tem como artefato de entrada a Lista de Classes Externas e os produtos gerados são os Cenários de Integração.

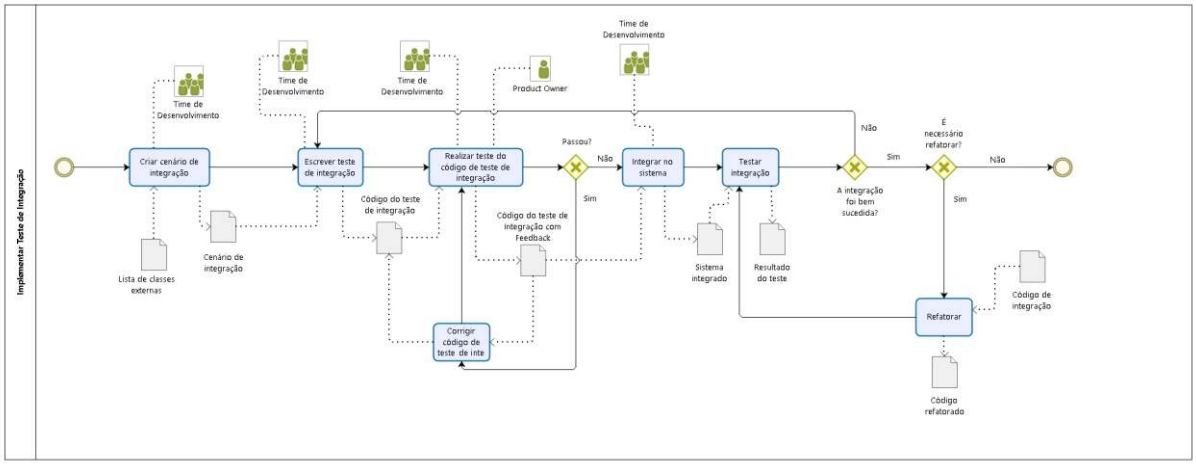

Figura 14 - Modelo do Subprocesso Implementar Teste de Integração.
Fonte: Próprio Autores - 2018

O fluxo do processo segue pelas atividades Escrever Teste de Integração e Realizar Teste do Código de Teste de Integração. Nesta última atividade, é observado o retorno do *framework* e caso o teste tenha sido bem sucedido, parte-se para a integração da funcionalidade no sistema. Caso contrário, o código é corrigido. Após a correção, o código de teste é integrado no sistema e a aplicação é testada. Se a integração for bem sucedida, procede-se para a refatoração e, após o código ser refatorado, a integração é testada novamente. Se não for necessário refatorar o código o subprocesso é finalizado.

Vale ressaltar que no processo de Desenvolver o Produto são utilizadas as práticas de desenvolvimento iterativo e incremental sugeridas pelo método XP. Portanto, como pode ser observado no processo, cada teste unitário escrito pode ser sucedido por um teste de integração. Porém, é importante observar a dinâmica do sistema, isto é, as dependências entre as classes, para evitar exageros na produção dos testes de integração e, desta forma, atrapalhar a fluidez do desenvolvimento do produto.

No fim do subprocesso Desenvolver Produto são verificados se ainda existem cenários de teste unitário para implementar, cenários de integração e funcionalidades. Caso exista algum cenário de teste ou funcionalidade, então o fluxo é retomado para a atividade correspondente a um dos três questionamentos (*gateways*), como visto na Figura 15.

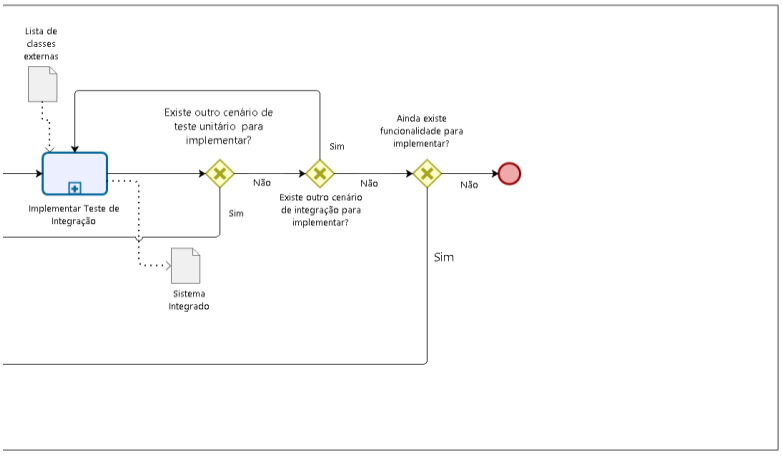

Figura 15 - Fim do subprocesso Desenvolver Produto.
Fonte: Próprio Autores - 2018

Finalizada a etapa de desenvolvimento do produto, retorna-se à reunião diária até que tenha se esgotado os dias necessários para o término da *sprint*. O *Product Owner* verifica quais itens da *sprint* estão prontos, segundo a definição do *Scrum*, e os entrega para a etapa de Revisão da *Sprint*, realizada pelos Membros do Projeto e o Time *Scrum*, onde analisa-se o que foi completo do *Backlog* do Produto e o atualiza.

A seguir, realiza-se a Retrospectiva da *Sprint*, onde o *Scrum Master* e o Time de Desenvolvimento reúnem-se para discutir o desempenho do time e propor melhorias para a próxima *sprint*, encerrando-se assim uma *sprint*.

A partir daqui, retorna-se ao processo macro do *ExScrum*, onde logo após a execução da *sprint*, é feita a verificação do fim da *Release*. Caso a mesma não esteja finalizada, é realizada a execução de uma nova *sprint*, com o resultado obtido da anterior, até que a *Release* seja finalizada. Uma vez finalizada, o *Product Owner* homologa a versão do produto. Caso esteja de acordo com os desejos do *Product Owner*, é realizada a cerimônia de entrega do produto pelos Membros do Projeto. Se ele não estiver pronto, então é necessário retornar para Planejar *Release* e caso seja necessário rever as funcionalidades, retorna-se para Construir Modelo Abrangente. Após a entrega da versão do produto, é verificado se há itens restantes no *Backlog* do produto. Se ainda conter itens, será feito um novo planejamento de *release* para implementá-los, do contrário o desenvolvimento do produto está finalizado. A Figura 16 ilustra essa etapa final.

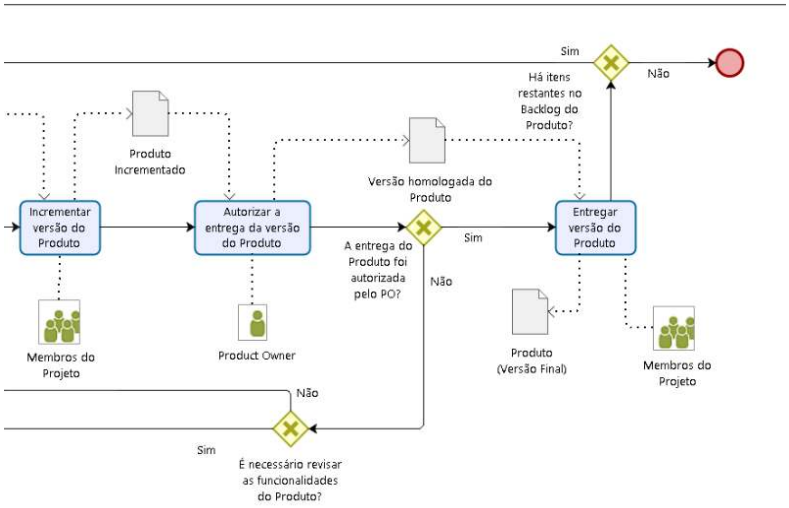

Figura 16 - Etapa Final do Modelo *ExScrum*.
Fonte: Próprio Autores - 2018

5. Avaliação de *ExScrum*

A ferramenta SWOT é usada para fazer análise de cenário ou análise de ambiente utilizado para definir o posicionamento estratégico de uma empresa. O termo SWOT tem origem no inglês e é uma sigla de *strengths* (forças), *weakness* (fraquezas), *opportunities* (oportunidades) e *threats* (ameaças) (SANTOS *et al.*, 2010). A análise SWOT foi desenvolvida na década de 60, na Universidade de Harvard, a partir de uma estrutura resultante de discussões em sala de aula e posteriormente desenvolvido no intuito de ter uma estrutura para trabalhar com definições de estratégias de negócios (TAKAO *et al.*, 2006). Assim, a Tabela 1 apresentar a matriz de SWOT usada para avaliar o processo ExScrum.

Tabela 1 - Matriz SWOT do processo *ExScrum*

Ambiente Interno	**Forças**	**Oportunidades**	**Ambiente Externo**
	● A satisfação do cliente; ● Modelo abrangente; ● Modelo de escopo aberto; ● Melhor gestão em todas as fases do projeto; ● Maior visibilidade ao cliente, durante o andamento do projeto, por se tratar de um processo incremental.	● Avanço das demandas de projetos de software; ● Necessidade de entrega dos produtos demandados dentro do prazo estabelecido.	
	Fraquezas	**Ameaças**	
	● Processo grande com vários fluxogramas; ● Requer maturidade no processo.	● Projeto de escopo fechado.	

Fonte: Próprio Autores - 2018

Diante da análise na Matriz SWOT, percebe-se que o modelo *ExScrum* tem vários benefícios (pontos fortes), como se vê a seguir:

- A satisfação do cliente - uma vez que o processo foca na necessidade do cliente, buscando compreender suas necessidades, valorizando sua cultura organizacional e alinhamento estratégico;
- Modelo de escopo aberto - o que possibilita adequar o projeto às demandas do cliente, fazendo uma nova priorização de acordo com suas necessidades e demandas;
- Melhor gestão em todas as fases do projeto - possibilita ter um maior controle sobre o projeto, podendo identificar os gargalos que possam ser gerados, além de possibilitar um maior controle do emprego dos recursos (humanos e não-humanos);
- Maior visibilidade ao cliente durante o andamento do projeto por se tratar de um processo incremental - ao final de cada incremento o cliente recebe parte do que foi produzido, que foi priorizado anteriormente, podendo determinar qual função é requerida para o próximo *sprint*.

Entretanto o processo *ExScrum* possui como fraqueza, na perspectiva interna, o ponto:

- Processo grande com vários fluxogramas - por ser um modelo robusto, que abrange todas as fases do processo de engenharia de software de forma aprofundada, possui vários fluxogramas que compõem o modelo. Isto exige do usuário o conhecimento nos métodos ágeis, para melhor utilização e compreensão do modelo.

Na perspectiva externa têm-se como oportunidades a crescente demanda de projetos de software. Com o avanço da tecnologia novas necessidades surgem e novas soluções são requeridas, demandando, assim, por novos projetos para satisfazer estas demandas. Neste cenário, o processo *ExScrum* destaca-se por ter seu fluxograma adequado para este cenário de volatilidade e mudanças frequentes no projeto. Além de proporcionar

uma melhor gestão das fases de execução do projeto, dando uma maior transparência para o cliente e melhor gestão por parte do gerente de projeto, possibilitando uma melhor utilização dos recursos e, consequentemente, influenciam diretamente nos prazos estabelecidos de entrega do produto.

Uma ameaça identificada, na perspectiva externa, são os projetos demandados de escopo fechado, pois uma vez que conhecido todos os requisitos e, consequentemente, não tendo necessidade de mudanças no decorrer da execução do projeto, a utilização do processo *ExScrum* não é adequada, por possuir um fluxograma grande e focar nos projetos de escopo aberto, onde não se conhece toda ou parte das necessidades do cliente, e sofre mudanças no decorrer da execução do mesmo.

6. Considerações Finais

No desenvolvimento do processo integrado buscou-se consolidar vários modelos de processo com focos de atuação diferenciados dentro do desenvolvimento de software para que uma equipe de desenvolvedores consiga produzir um produto com maior qualidade, agilidade e facilidade. A qualidade advém do uso de processos amplamente usados no mercado e com resultados constatados na melhoria do processo de desenvolvimento de software. A agilidade é inerente do próprio uso de metodologias ágeis com foco na conclusão das funções mais importantes para o cliente, que consequentemente diminui manutenções onerosas do projeto e produção de funções inúteis ou vagamente usadas.

Para oferecer estes benefícios aos utilizadores desse processo integrado, foram pensadas as uniões de etapas dos diferentes processos de forma harmônica e que contornam as fraquezas que os mesmos possuem quando implementados de forma isolada para executar todas as etapas que um processo de desenvolvimento de software necessita.

Este processo integrado também serve de guia para a produção de novos processos que venham utilizar os mesmos métodos ágeis escolhidos por este trabalho ou outros métodos.

O *ExScrum* é pensando para o desenvolvimento de qualquer software de escopo aberto e que pode ser adaptado conforme a demanda e utilização do time de desenvolvimento que o executa, onde, por exemplo, poderia ser desconsiderada a etapa de utilização do processo do TDD para diminuir o tempo de desenvolvimento do projeto.

Como trabalhos futuros pretende-se implementar o *ExScrum* em pelo menos uma empresa de desenvolvimento de software na região como um estudo de caso para verificar o desempenho do uso deste processo e as restrições e problemas que possam surgir na sua aplicabilidade real.

Referencias Bibliográficas

ANICHE, M. (2012). Test-Driven Development: Teste e Design no Mundo Real. v. 15.2.11. São Paulo: Casa do Código.

AZEVEDO, S. (2008). Porque os Projetos Falham. Revista Mundo PM - Project Management. Disponível em: *http://www.mundopm.com.br/noticia.jsp?id=280*. Acesso em: 02/2018.

BECK, K. (2000). Extreme Programming Explained. Massachusetts: Addison-Wesley.

BECK, K. (2010). TDD Desenvolvimento Guiado por Testes. 1. ed. Porto Alegre: Bookman Editora.

BOURQUE, P.; FAIRLEY, R. E. (2014). Guide to the Software Engineering Body of Knowledge. Version 3.0, IEEE Computer Society.

PRIKLADNICKI, R.; WILLI, R.; MILANI, F. (2014). Métodos Ágeis para Desenvolvimento de Software.1 ed. Porto Alegre: Bookman.

GOOGLE TRENDS (2018). Relatório de pesquisa do google trends. Google Trends. Disponível em: *https://https://trends.google.com.br/trends/explore?cat=174&date=today 5-y&geo=BR&q=scrum,xp,fdd,tdd*. Acesso em: 02/2018.

MILLER, K. W. (2018). Test Driven Development on the Cheap: Text Files and Explicit Scaffolding. Disponível em: *http://www.ccsc.org/northwest/docarchive/2004/augustmailing2004final.doc*. Acesso em: 02/2018.

PRESSMAN, R. S. (2011). Engenharia de Software. 7ª Edição, Editora Amgh.

RETAMAL, A. M. (2011). FDD – Feature-Driven Development: Descrição de Processos. Disponível em: *http://www.heptagon.com.br/files/FDD-Processos.pdf*. Acesso em: 02/2018.

SCHWABER, K.; SUTHERLAND, J. (2013). Guia do Scrumtm. 2013. Disponível em: *http://www.scrumguides.org/*. Acesso em: 02/2018.

SOARES, M. S. (2004a). Comparação entre metodologias Ágeis e tradicionais para o desenvolvimento de software. INFOCOMP, 3(2), 8-13.

SOARES, M. S. (2004b). Metodologias ágeis extreme programming e scrum para o desenvolvimento de software. Revista Eletrônica de Sistemas de Informação ISSN 1677-3071 doi: 10.21529/RESI, 3(1).

SANTOS, M. A.; GREGHI, J. G.; BERMEJO, P. H. S. (2010). Avaliação do Impacto do SCRUM no desenvolvimento de software utilizando a análise SWOT. São Paulo.

TAKAO, E. L.; COPPPINI, N. L.; TOREGEANI, A. F. (2006). Aplicação da Técnica SWOT na Viabilização de um Sistema de apoio para Executivo com Software Livre em uma Indústria de Embalagens Plásticas. CADERNO DE ADMINISTRAÇÃO. v. 14, n.1, p. 9-17.