CENTRO UNIVERSITÁRIO DE ANÁPOLIS
UniEVANGÉLICA
CURSOS SUPERIORES DE COMPUTAÇÃO

METODOLOGIA ÁGIL: PROPOSTA E AVALIAÇÃO DE MÉTODO PARA DESENVOLVIMENTO DE SOFTWARE

WAGNER ALVES GONÇALVES NOGUEIRA

ANÁPOLIS
NOVEMBRO 2016

WAGNER ALVES GONÇALVES NOGUEIRA

METODOLOGIA ÁGIL: PROPOSTA E AVALIAÇÃO DE MÉTODO PARA DESENVOLVIMENTO DE SOFTWARE

Trabalho de Conclusão de Curso I apresentado como requisito parcial para obtenção do grau de Bacharel no curso de Engenharia de Computação do Centro Universitário de Anápolis – UniEVANGÉLICA.

Orientador: Esp. McGill Evaristo Dias.

Co-Orientadora: Msc. Viviane Carla Batista Pocivi.

ANÁPOLIS

NOVEMBRO 2016

TERMO DE APROVAÇÃO

WAGNER ALVES GONÇALVES NOGUEIRA

METODOLOGIA ÁGIL: PROPOSTA E AVALIAÇÃO DE MÉTODO PARA DESENVOLVIMENTO DE SOFTWARE

Trabalho de Conclusão de Curso II apresentado como requisito parcial para obtenção do grau de Bacharel no curso de Engenharia de Computação do Centro Universitário de Anápolis – UniEVANGÉLICA.

_____

Esp. McGill Evaristo Dias

_____

Ma. Viviane Carla Batista Pocivi

_____

Convidado 1

_____

Convidado 2

Anápolis, 04 de Novembro de 2016

AGRADECIMENTOS

Primeiramente a Deus por conceder forças para que eu pudesse chegar até aqui, especialmente à minha mãe que me acompanhou, ajudou e lutou comigo desde o início.

Aos Mestres que passaram e deixaram todo o seu conhecimento em prol do nosso crescimento profissional, e em especial aos meus orientadores McGill Evaristo Dias e Viviane Carla Batista Pocivi.

RESUMO

A intenção desta pesquisa é abordar algumas das diversas metodologias ágeis e alguns dos seus vários processos existentes para desenvolvimento de software que são propostos com a intenção de valorizar o profissional, mudando a cultura de se trabalhar e administrar um projeto utilizando metodologias ágeis, como *Scrum, Extreme Programming (XP) e Feature Driven Development (FDD)*. Com o uso de um processo centralizado, utilizando um conjunto de atividades e práticas voltadas para a solução de problemas encontrados durante todo o ciclo de vida de desenvolvimento de um software ou funcionalidade. A utilização do processo é o caminho mais rápido para obter software de qualidade, priorizando alguns objetivos como a definição de como, onde e quem executará determinadas tarefas e padronização do processo. A junção dos processos de software e engenharia de software traz métodos e metodologias a serem trabalhadas, analisadas e, se viável, implantadas. Quando tratado de métodos e metodologia ágil, a maioria destas tenta atenuar o risco pelo desenvolvimento em curto prazo, com duração de duas a quatro semanas, enfatizando a comunicação dos membros da equipe em tempo real. Os métodos ágeis visam à entrega de soluções de software ao cliente em um prazo mais curto e com maior qualidade. Existem diversas formas de abordar uma metodologia ágil, como por exemplo, quando a empresa adota a metodologia e toda equipe se adapta a ela. Outra forma seria a empresa elaborar um processo híbrido que se adapte à sua rotina de desenvolvimento, reunindo características de metodologias ágeis para a construção de um único processo. Para realizar este projeto de pesquisa, utilizará de estudo de caso, feito na empresa Infopharma, levantando os costumes, processos e culturas que hoje são aplicados no desenvolvimento de softwares dentro da empresa. O objetivo do projeto é, propor um processo hibrido de desenvolvimento de software, utilizando métodos e técnicas ágeis. Uma análise comparativa será realizada entre as metodologias, definindo quais são as características que irão melhor se adequar à realidade da empresa. E por fim, implantar gradativamente, e analisar os resultados obtidos.

PALAVRAS-CHAVES: Metodologias Ágeis. Processo Centralizado. Padronização do Processo. Processo Híbrido.

ABSTRACT

The intention of this research is to address some of the various agile methodologies and some of its various existing processes for software development that are proposed with the intention of enhancing the professional, changing the culture of working and managing a project using agile methodologies like Scrum, Extreme Programming (XP) and Feature Driven Development (FDD). With the use of a centralized procedure, using a set of focused activities and practices for troubleshooting found throughout the development life cycle of a software or functionality. The use of the process is the fastest way to get quality software, prioritizing some goals as the definition of how, where and who will perform certain tasks and process standardization. The combination of software processes and software engineering provides methods and methodologies to be worked on, analyzed and, if feasible, implemented. When treated methods and agile, most of these attempts to mitigate the risk by short-term development, lasting two to four weeks, emphasizing the communication of team members in real time. Agile methods are aimed at delivery of customer software solutions in a shorter time and with higher quality. There are several ways to approach an agile methodology, such as when the company adopts the methodology and the whole team adapts to it. Another way would be the company develop a hybrid process that suits your routine development, bringing together features of agile methodologies for the construction of a single process. To carry out this research project, use the case study, the company Infopharma, raising the customs, processes and cultures that are applied today no software development within the company. The purpose of the project is to propose a software development process using agile methods and techniques. A comparative analysis should be performed on methodologies, defining which are as characteristics that adequately improve the reality of the company. And finally, gradually deploy, and analyze the results obtained.

KEYWORDS: Agile Methodologies. Centralized process. Standardization process. Hybrid Process.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

XP	Extreme Programming
FDD	Feature Driven Domain
ERP	Planejamento dos Recursos da Empresa
UML	Linguagem de Modelagem Unificada
FBS	Feature Breakdown Structure
CRC	Class Responsability Card
SVN	Sistema de Controle de Versão
TI	Tecnologia da Informação

SUMÁRIO

1. INTRODUÇÃO

Desde o início dos anos 2000, o desenvolvimento ágil nas corporações vem ganhando espaço e mercado. Em muitas circunstâncias pode gerar melhores resultados para as corporações que adotam algumas de suas metodologias.

A pesquisa será feita mediante um estudo de caso feita na empresa Infopharma. A Infopharma está no mercado há cinco anos oferecendo soluções de software e documentação para o setor farmacêutico. Atualmente, a equipe tem se dedicado ao desenvolvimento do projeto principal, no qual se encontra em crescente evolução.

No entanto, a equipe de desenvolvimento é pequena, não há padronização na utilização de métodos e processos definidos para o desenvolvimento do projeto, resultando em índices não satisfatórios, havendo falhas de comunicação e mal entendimento do escopo, onde a regra de negócio está centralizada ao gestor da empresa.

Mediante a este cenário será proposto um processo de desenvolvimento de software baseado em metodologias ágeis com a intenção de atenuar estas falhas dentro do ambiente de desenvolvimento. O intuito será de trazer para o processo proposto o melhor de cada uma das metodologias, mesclando atividades de engenharia de software, gerência de projetos e desenvolvimento de software, onde o resultado esperado é um processo que não gere um impacto de adaptação e tempo para equipe, otimizando cada etapa de desenvolvimento, sendo um processo leve que atenderá as mudanças constantes que poderão surgir no decorrer do trabalho.

As metodologias escolhidas para compor o processo será *Scrum, Extreme Programming (XP) e Feature Driven Development (FDD),* onde será feito um trabalho de análise dessas metodologias em cada etapa do fluxo de desenvolvimento do projeto. Dessa forma, espera-se obter características de cada metodologia e então analisar e comparar o que cada uma tem de melhor das demais, onde de uma característica, ou até mesmo unindo traços de duas características de métodos diferentes, poderá ser obtida uma atividade que resulte de forma positiva. Cada uma dessas características escolhidas será uma atividade que irá compor o processo de desenvolvimento que será proposto.

Cada etapa do processo será validada e testada com a gestão da Infopharma para certificar que aquela atividade realmente irá gerar resultados positivos e relevantes facilitando o desenvolvimento e melhorando a comunicação, desde o planejamento do projeto até o final das pequenas partes funcionais.

A implantação do processo será feita gradativamente, aplicando, inicialmente, a uma funcionalidade em que o processo precisará ser capaz de facilitar a visão do problema a ser desenvolvido, planejado e gerenciado da atividade e construção. Visto de forma que enfatize o planejamento e a previsão de soluções para cada etapa do desenvolvimento resolvendo cada um dos seus problemas.

Para realizar esta pesquisa serão utilizados os conceitos dos principais autores como Sommerville e Pressman, voltados para a área de engenharia de software, Pham e Cohn que tratam de gerenciamento de projeto com Scrum e Highsmith autor do livro Gerenciamento ágil de projeto.

As metodologias que apresentam características que suprem a necessidade de desenvolvimento, planejamento, administração e manutenção de software da empresa são: *XP*, *Scrum e FDD*.

"Os métodos ágeis visam à entrega de soluções de software ao cliente em um prazo mais curto e com maior qualidade. Existem diversas formas de abordar uma metodologia ágil, uma delas é a empresa adotar a metodologia e toda equipe se adaptar a ela. Outra forma seria a empresa elaborar um processo híbrido que se adapte à sua rotina de desenvolvimento reunindo características de metodologias ágeis para construção de um único processo. " (HIGHSMITH, 2011)

Roger Pressman (2006) ainda afirma que o diferencial das metodologias ágeis das metodologias tradicionais são o enfoque e os valores, ou seja, para uma metodologia ser considerada ágil deve enfatizar pessoas ao invés de processos e algoritmos, sendo assim nossa prioridade é satisfazer o cliente através da entrega contínua e adiantada de software com valor agregado, disseminar conhecimento entre a equipe além de melhorar a qualidade da codificação dos sistemas.

Este trabalho se propõe a desenvolver um processo híbrido de desenvolvimento de software, utilizando metodologias ágeis para empresa de desenvolvimento de pequeno porte. Uma análise comparativa será realizada entre as metodologias, definindo as características que melhor se adequam à realidade da empresa Infopharma. E por fim, implantar e analisar a metodologia proposta.

Como resultado obtido, foi gerado uma imagem representando o fluxo do processo de desenvolvimento, ilustrando as etapas que compõe o projeto. Cada etapa do processo é composta por uma ou mais características, que contém nas metodologias utilizadas para o estudo. O processo foi implantado em funcionalidades piloto, em cada funcionalidade foi aplicado o processo, e feito uma análise baseando nos feedbacks dos envolvidos. Após cada

análise, os pontos levantados que tiveram uma repercussão negativa, foram tratados e evoluídos. Como apresentado, na segunda funcionalidade implantada, o processo já obteve resultados mais satisfatórios.

2. REFERENCIAL TEÓRICO

2.1. VISÃO GERAL

Segundo Pressman (2011) Processo foi definido como um conjunto de atividades de trabalho, ações e tarefas quando algum artefato de software deve ser criado. Onde cada atividade aloca-se dentro de uma metodologia ou modelo que determina seu relacionamento entre elas. Tornando uma forma eficiente para que o software evolua respondendo às necessidades do cliente e às suas mudanças, sempre incluindo o feedback.

Segundo HIGHSMITH (2012) os processos devem ser adaptados conforme as necessidades do ambiente. As formas como as pessoas fazem suas atividades e como se comportam é potencializada para criar novos produtos.

Para que ocorra essa evolução é necessário utilizar um conjunto coerente e coordenado de métodos, fornecendo um roteiro, um processo dinâmico e interativo. O objetivo de utilizar metodologia é definir de forma clara todos os papéis e responsabilidades para todos os envolvidos no projeto, com isso possibilita o gerenciamento do projeto para que não haja desvios de planejamentos de custos e prazos, ser produzido de forma eficiente e com qualidade.

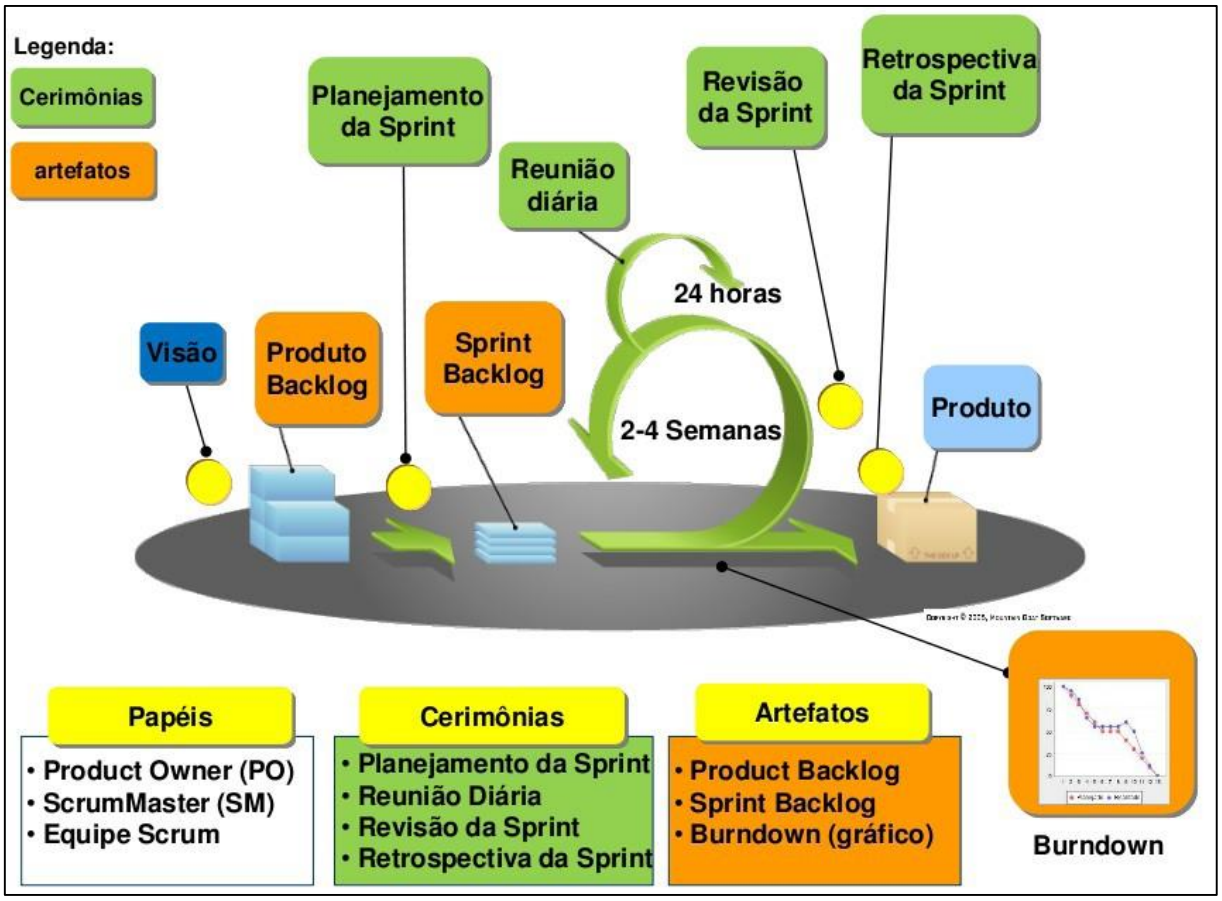
O uso da metodologia não deve limitar a criatividade do profissional, mas que seja um instrumento que determine um planejamento, unindo e coordenando as áreas envolvidas. Assim, ao se considerar uma proposta para formalização de um processo ou método de desenvolvimento de software, sem deixar de lado estímulo por diversas abordagens. Considerando metodologias ágeis, não é raro que uma organização use Scrum, programação extrema (XP) e ainda o Desenvolvimento Orientado a Funcionalidade (FDD); Onde um dos pressupostos do desenvolvimento ágil é a adaptação a diferentes situações.

2.2. METODOLOGIA SCRUM

O Scrum é um framework para gestão e planejamento de projeto, pensado e criado para atingir pontos críticos na gerência de projetos reduzindo dificuldades como falta de planejamento, mudança constante de requisitos, escopo mal definido, falta de participação do cliente e falhas nas comunicações. Baseado em entregas rápidas, contínuas e frequentes de softwares funcionais. Cada um tem o seu papel e responsabilidades definidas, o time é composto por três papeis: Scrum Master, Product Owner e Scrum Team.

A figura 1 apresenta a estrutura do framework Scrum informando as cerimônias que acontecem a cada etapa do desenvolvimento e cada artefato que deverá ser produzido em cada etapa. Logo abaixo ainda neste capítulo será detalhado o que é trabalhado em cada etapa do Scrum.

Figura 1 – Fluxo de Processo do Scrum



Fonte: http://www.desenvolvimentoagil.com.br

Iniciaremos falando sobre os papéis que compõe o time Scrum, o Scrum Master em princípio pode ser qualquer pessoa da equipe, mas geralmente é exercido por um gerente de projeto ou um líder técnico. Atua como um facilitador para equipe, facilitando a resolução de obstáculos e lidando com quaisquer problemas que surjam protegendo a equipe de perturbações externas assegurando que ela não se comprometa excessivamente com relação àquilo que é capaz de realizar durante um Sprint.

O Product Owner é o dono do projeto, fornece todo o conhecimento do negócio em forma de requisitos para a equipe. O Product Owner que faz a relação entre a empresa e os clientes, sendo visto como o ponto de contato para o esclarecimento das dúvidas da equipe sobre os requisitos do produto.

Team Scrum é o time de desenvolvimento composta por engenheiros e programadores, sua responsabilidade é pelo desenvolvimento do projeto, ela também deve ter a autonomia para tomar decisões sobre como executar o seu trabalho. Deve ser uma equipe auto organizada, multidisciplinar e composta por um número de 3 a 9 pessoas para evitar dificuldade de comunicação e diminuição da produtividade.

Sobre as cerimonias que acontecem durante o processo do *Scrum* iniciarei falando sobre a visão, onde o responsável se reúne junto ao cliente para entender suas necessidades e então levantar os requisitos para o desenvolvimento do projeto. Será feita uma lista contendo todas as funcionalidades que foram capazes de abstrair que irá compor o produto, não é preciso estar completo visto que, com o decorrer do desenvolvimento novas expectativas vão surgindo. O resultado esperado para esse momento de reunião com o cliente é criar o *Product Backlog* contendo as funcionalidades desejadas para o produto.

Planejamento da *Sprint* consiste em reunião onde estão presentes o *Product Owner, Scrum Master,* todo o time de desenvolvimento e uma equipe representando o cliente. O objetivo é descrever funcionalidades que representam maior prioridade para a equipe, e quebrar essas funcionalidades em tarefas técnicas. Não é necessário descrever todos os itens que estão no *Product de Backlog.*

Sprint Backlog momento onde o time se compromete a fazer uma *Sprint* extraindo os itens do *Product Backlog* com base nas prioridades definidas pelo *Product Owner* considerando o tempo e complexidade da funcionalidade. Neste momento é determinado quantos itens serão trazidos pela equipe que irão compor a *Sprint,* essa definição depende do relacionamento dos itens entre si, caso houver um item que não se relaciona pode-se definir uma *Sprint* apenas com este item.

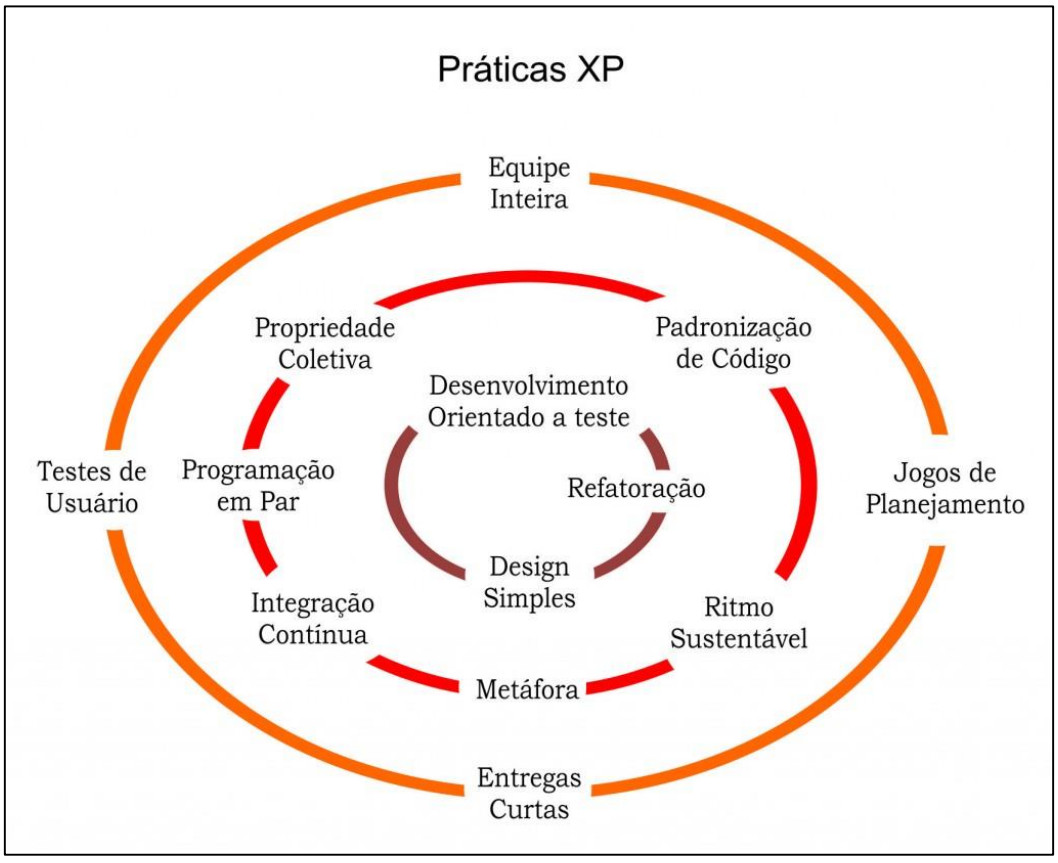
Ao final de cada *Sprint* ocorre o momento chamado de *Retrospective*, que serve para identificar o que deu certo, o que pode ser melhorado e quais ações a equipe deve tomar para que essa melhoria seja efetiva. Acontece também a revisão mostrando o que foi alcançado durante o desenvolvimento da *Sprint* avaliando o projeto em relação à *Sprint*.

2.3. METODOLOGIA XP

XP é uma metodologia ágil orientada explicitamente a pessoas e vai contra o senso comum do gerenciamento de que as pessoas são peças intercambiáveis do processo, conceituando que os profissionais envolvidos podem ter sucesso com a simples adoção e depois com a obediência a uma coleção de princípios e práticas. Esses princípios existem para servir de ponte entre valores e práticas guiando a um domínio específico.

Segundo Pressman (2006, p.88), o XP prefere uma abordagem orientada a objetos envolvendo quatro atividades metodológicas: Planejamento, projeto, codificação e testes. Figura 2 apresenta as práticas do XP durante o desenvolvimento podendo perceber

Figura 2: Práticas XP



Fonte: www.devmedia.com.br

Ah uma grande confiança na sinergia entre as práticas, as características têm pontos fortes assim como também tem seus pontos fracos, que são superados pelos pontos fortes das outras. Abaixo citarei brevemente cada prática proposta pelo XP.

Equipe Inteira: Onde cada pessoa que participa do projeto tem uma habilidade necessária para o desenvolvimento do projeto, o objetivo é que todos participem do desenvolvimento do projeto.

Jogo de Planejamento: Onde os desenvolvedores por meio de reunião priorizam as funcionalidades, o desenvolvimento é feito em iterações semanais. Os clientes participam dessa reunião, são eles que identificam as prioridades para que possam ser estimadas. E no final de cada semana o cliente recebe novas funcionalidades prontas para produção, totalmente testadas.

Entregas Curtas: As versões produzidas chegam a ser menores que as produzidas por outras metodologias incrementais como o *Scrum.* As pequenas versões funcionais auxiliam no processo de aceitação por parte do cliente, que dá a liberdade de testar parte do sistema que está comprando.

Testes de usuário: É um roteiro de teste produzido pelos testadores, analistas e clientes para aceitar e validar um requisito do sistema.

Padronização de código: São regras estabelecidas pela equipe de desenvolvimento para programar e todos devem seguir essas regras, de forma que o código parece que foi feito por uma só pessoa.

Ritmo Sustentável: É a busca da qualidade de vida no trabalho, sem realizar horas extras exceto quando trouxer produtividade para execução do projeto, trabalhando apenas 40 horas semanais. Buscando a motivação dos membros com um ritmo de trabalho bem energizado, mas para isto o ambiente de trabalho e a motivação da equipe devem estar sempre em harmonia.

Metáfora: É onde o cliente e equipe de desenvolvimento procuram conversar na mesma língua para que ambos tenham o mesmo entendimento sobre o projeto. Um conceito de rápido para um cliente de contabilidade é diferente para um programador experiente em controlar comunicação em tempo real.

Integração Contínua: Sempre atualizar a versão no sistema atual com a nova funcionalidade após pronta, não é necessário esperar uma semana para integrar a funcionalidade ao projeto, essa integração de forma contínua permite saber o status do real desenvolvimento. Integrando de forma contínua reduz a possibilidade de conflitos e a possibilidade de erros no código fonte.

Programação em Pares: É a programação em dupla utilizando o mesmo computador, geralmente formada por um programador experiente servindo como instrutor e outro iniciante na linguagem. Como ambos utilizam o mesmo computador o novato fica a frente de fazer a codificação, e instrutor auxiliando no desenvolvimento ajudando a desenvolver suas habilidades.

Propriedade Coletiva: O código fonte não tem dono e qualquer um pode editar o que quiser sem precisar pedir permissão para fazer alteração, o objetivo é fazer com que toda equipe conhecer todas as partes do sistema.

Desenvolvimento Orientado a Testes: Para cada funcionalidade crie os testes unitários e façam com que eles funcionem. Os testes unitários são essenciais para que seja mantida a qualidade do projeto.

Refatoração: É um processo de melhoria contínua no código, melhorando a clareza e leitura do código dividindo em métodos mais consistentes e de maior reaproveitamento em outras partes do código, evitando a duplicidade de código fonte.

Design Simples: Simplicidade é um dos princípios do XP, é fazer um código simples e exato somente para que a funcionalidade seja implementada. Um exemplo é criar um código para o cliente acessar o sistema com o usuário “teste” e senha “123” sem se preocupar com ferramentas ou frameworks de autorização e autenticação.

O tamanho da equipe que o autor aborda é entre pequenas e médias, onde a mudança aos requisitos é constante, utilizando uma abordagem totalmente incremental encorajando assim a comunicação entre as pessoas.

O objetivo do XP é dar agilidade ao desenvolvimento garantindo a satisfação do cliente, insistindo em fazer com que o cliente trabalhe diretamente no projeto. A priori no cliente dá a ele o poder de responder perguntas e tomar decisões relativas à prioridade de recursos, riscos e assim por diante.

Os papéis no XP não são fixos fazendo assim que cada contribua com o melhor que tem a oferecer para que a equipe tenha sucesso. É visto no XP que papéis fixos são úteis para se aprender novos hábitos, mas a partir do momento que há uma relação respeitosa estabelecida entre os membros da equipe os papéis fixos já podem ser interferir no objetivo de fazer com que cada dê o melhor de si.

No XP os papéis são mais abrangentes do que na metodologia Scrum, eles são: Analista de testes, arquitetos, designers, executivos, gerentes de projeto, gerentes de produto, programadores, recursos humanos, redatores e usuários.

O XP faz uso de boas práticas dando liberdade para cada modelar sua própria forma de trabalho com o XP. É de extrema importância que se entenda bem a essência do XP e principalmente que se tenha disciplina e criatividade dessa forma toda poderão usar e ter benefícios através do XP.

Essas boas práticas falando sucintamente visa o cliente sempre disponível para colaborar em dúvidas, alterações... dando um dinamismo ao projeto. Integração contínua onde os módulos são integrados e testados diversas vezes por dia. Simplicidade do projeto onde o código está a qualquer momento escrito na forma simples e clara conforme o padrão definido pela equipe, podendo ou não sofrer melhorias constantes de refatoração levando em consideração requisitos como prazo e custos. Programação em pares, no XP diz que deve ser feito com duas pessoas trabalhando no mesmo código com o mesmo teclado. É feito rodízio entre essas duplas com o objetivo de uniformizar os códigos assim a equipe por um todo é responsável por cada arquivo, segundo a Microsoft soma forças para a implementação do projeto.

2.4. METODOLOGIA FDD

FDD é uma metodologia ágil para gerenciar e desenvolver softwares com o lema de resultados frequentes, tangíveis e funcionais.

Com base na lista de funcionalidades deve-se planejar de forma incremental, após o planejamento é feito o detalhamento, é de extrema importância que o desenho esteja de acordo com o cliente deseja, então é mantido contato constante com a equipe de clientes.

Esta documentação é necessária para o bom desenvolvimento, documentando apenas o que irá ser utilizado de forma que não tome muito tempo. Após documentar inicia-se a fase do desenvolvimento e testes, sendo de forma incremental e testando do início ao fim do processo com integração de cada incremento continuamente. Lembrando que no FDD o código já é propriedade de quem o desenvolveu, diferindo do XP onde o código é comunitário.

A intenção da entrega contínua é a criação de um ambiente separado, onde as modificações individuais são unificadas ao projeto, onde se executa os testes, gera a documentação. Cada entrega é vista como um pedacinho, onde cada um desses é submetido a todas as fases do ciclo de vida do software.

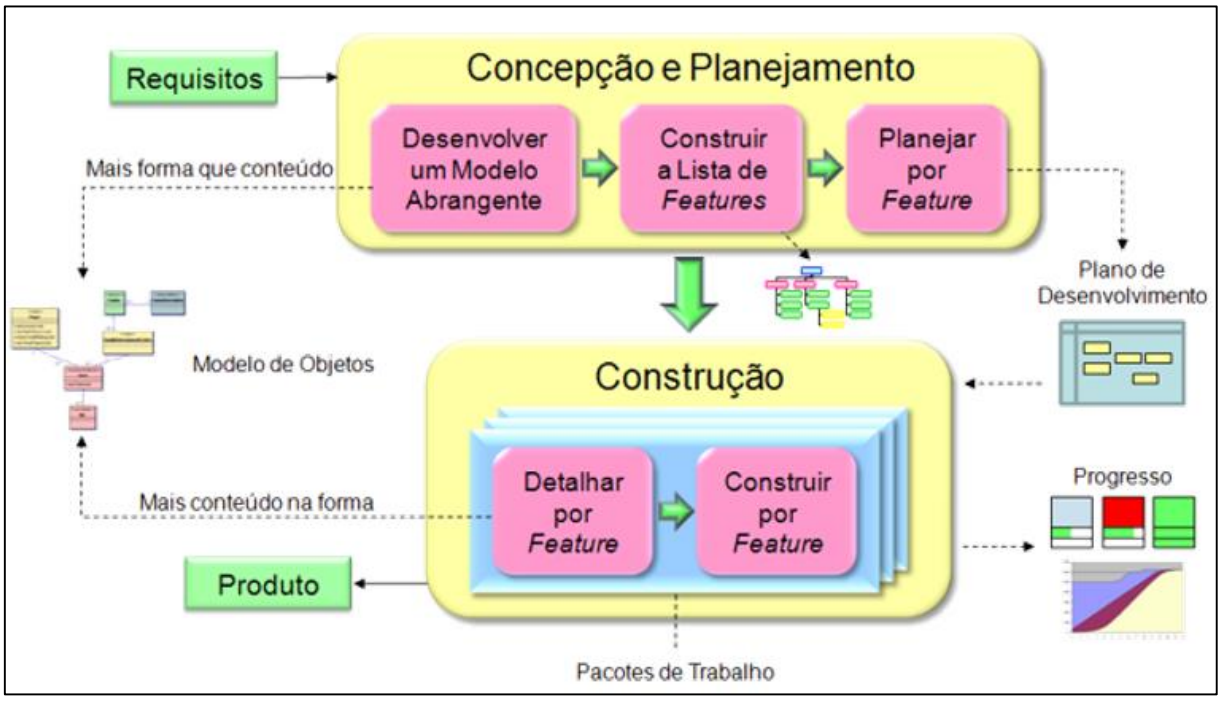
FDD combina as melhores práticas de gerenciamento de projeto, o seu forte é mesmo na Engenharia de software orientada a objetos. Seus princípios e práticas proporciona equilíbrio entre as metodologias mais tradicionais até as mais extremas, dessa forma propõe uma transição

mais suave para organizações com âmbito conservador, obtendo retomada de responsabilidades para as organizações que se desiludiram com as propostas mais radicais.

Como outras metodologias o FDD é uma metodologia bastante objetiva possuindo apenas duas fases que duram tipicamente entre uma a duas semanas. As fases são concepção & planejamento onde se pensa no modelo cria uma lista de características e planejar através delas, essa fase é feita apenas uma vez e construção onde já se desenvolve de forma iterativa e incremental.

Figura 3 mostra as duas fases proposta pelo FDD e os artefatos que devem ser criados em cada fase. Logo abaixo será descrito cada fase e seus artefatos:

Figura 3: Fases e etapas do FDD



Fonte: www.heptagon.com.br

O FDD conta também com cinco processos que são integrados e bem definidos, onde contém o desenvolvimento dos requisitos, modelagem lógica, decomposição do modelo de domínio, estimativas, prioridades de valor/cliente, testes, desenvolvimento e resultado. Irei citar brevemente os processos logo abaixo:

Desenvolver um modelo Abrangente – Neste processo o objetivo é gerar resultado que possa guiar a equipe durante os ciclos de construção. Neste processo ocorre o desenvolvimento dos requisitos, é feita uma análise orientada a objetos, modelagem lógica de dados.

Construir uma Lista de Funcionalidades: Aqui é onde se constrói o Product Backlog citado no Scrum, ou chamado também como lista de espera do produto criando uma hierarquia de funcionalidades que representa o produto. É feita uma decomposição funcional do modelo de domínio em camadas de áreas de negócio, atividades e passos automatizados da atividade (funcionalidade).

Planejar por Funcionalidade: Aqui se cria um roteiro de desenvolvimento com pacotes na sequência definida no processo anterior para a construção. Levando em consideração a prioridade e também o valor para o negócio/cliente, abrangendo estimativas de complexidade e dependência das funcionalidades.

Detalhar por Funcionalidade: Aqui monta o esqueleto de código dos modelos de domínios para serem preenchidos. Já dentro da construção a equipe detalha os requisitos e outros artefatos para a codificação da atividade, incluindo os testes.

Construir por Funcionalidade: Enfim a etapa onde cada esqueleto montado anteriormente é preenchido, testado e inspecionado oferecendo um release pronto para ser usada pelo usuário, o código integrado ao repositório principal do projeto.

FDD possui papéis bem definidos e os mesmos são divididos em três classes que são: Principais, apoio e adicionais. Na classe de papéis principais existe o gerente de projeto, arquiteto, gerente de desenvolvimento, programador chefe, proprietário de classe e especialista de domínio. No apoio fica o gerente de domínio, versão, especialista na linguagem, coordenador de construção, ferramenteiro e administrador do sistema. Em papéis adicionais fica o testador, desenvolvedores e escritor técnico.

3. ANÁLISE DAS METODOLOGIAS PROPOSTAS

3.1. VANTAGENS DE UTILIZAR O SCRUM

Risco: Reduzir o nível do risco ou a incerteza a zero, ou para menor possível utilizando um frequente ciclo de inspeção e adaptação.

Ciclo de vida: Ciclo de vida mais enxuto, otimizando o tempo produtivo.

Adaptação: Gestão adaptável, onde as mudanças e as alterações são as únicas certezas.

Comprometimento da equipe: Com a definição de prazos para cada Sprint definida a equipe participa ativamente da definição das atividades e cronogramas, por isso o grau de comprometimento é maior assim como a motivação em atingir a meta que é de finalizar cada Sprint no prazo que está proposto. Equipe auto motivada e disciplinada, mantendo o orgulho das pessoas.

Visão de projeto: Como os projetos passam a ter um backlog que possibilita o acesso a qualquer integrante do time, os projetos passam a ter uma visualização melhor do escopo dentro da organização. Sem os backlogs de produto os projetos ficavam ocultos e só os gerentes de projetos sabiam sobre o negócio do projeto, eram visualizados de forma integral.

Tempo de entrega: A velocidade de desenvolvimento está ligada a alguns fatores como diminuição de bugs e maior qualidade. Devido a maior interação da equipe durante o processo de construção do produto e pelo fato do método priorizar o software funcionando.

Papeis bem definidos: Todos tem o conhecimento sobre as suas responsabilidades.

Time pequeno: Como o time de desenvolvimento é pequeno onde o número máximo apontado por Pham é nove, então a comunicação e o espírito colaborativo é mais eficiente.

Gerenciamento: É um processo ágil e flexível, torna-se melhor a adaptação as mudanças que ocorrem durante o projeto buscando minimizar os riscos e maximizar a qualidade.

Redução de bugs: A equipe se compromete a entregar aquilo que ela consegue no prazo de uma Sprint, o prazo deixa de ser o vilão, prioriza a qualidade que passa a ser o centro das atenções e a funcionalidade proposta na Sprint, consequentemente reduz o retrabalho da equipe no mesmo artefato.

Agregar valor: A funcionalidade que agrega mais valor ao negócio do cliente deve ser entregue primeiro, e não o que seja mais fácil primeiro. Geralmente o cliente não tem definido o que é mais importante no projeto, por isso o Scrum tem essa flexibilidade então pode ser alterada essa definição da sequência de atividades, basta ter os requisitos.

3.2. DESVANTAGENS DE UTILIZAR SCRUM

Engenharia de software: Como o Scrum é voltado para o planejamento e gerenciamento de projeto há ausência de práticas de engenharia e desenvolvimento de software. Por este motivo necessita da associação de outra metodologia que contenha essa característica, por exemplo, o XP.

Cultura: É difícil de ser implementada, principalmente devida a resistência de mudanças culturais, pois para ter sucesso deve ser seguido regra por regra.

Papéis indefinidos: Entraria em contradição citar como vantagem definição de papéis logo acima, porém podemos dividir está visão em duas.

Os papeis são bem definidos somente na visão do Scrum onde existem só quatro pessoas: O cliente, o Product Owner, o Scrum Master e o Time. Mas onde entraria o analista

de negócio, o analista de sistema, o testador? Dá a entender que o Time é quem deve modelar isso nem sempre acontece, no time deveria conter também o analista de negócio? Se o time tiver as competências necessárias, funciona, caso contrário o resultado não será assim tão favorável.

Nível do grupo: O grupo deve ser de alto nível, todos os integrantes do grupo, pois são poucas pessoas trabalhando, onde entra o que citei no tópico acima. Por esse motivo ninguém sabe tudo o que precisa ser feito nos primeiros estágios de um projeto. Reconhecer que os indivíduos e times não podem saber tudo enquanto planejam, não importa em qual nível operam.

Tempo investido: Segundo um artigo postado no site da *cio.com.br,* há o risco de investir muito tempo em sessões de Scrum. Um bom alvo para comunicação é de 5% a 10% do tempo diário. Um projeto sofre quando a comunicação se torna menos de 5% do tempo, e os benefícios diminuem significativamente entre 9% e 10%.

Segundo Kai Gilb o Scrum é praticado como uma grande lista corporativa de coisas para fazer, reduzindo as tarefas sem a necessidade de desenvolvedores com habilidades criativas ditando a eles o que fazer.

Kai cita também que exige soluções feitas por amadores que seriam os clientes, onde essas soluções deveriam serem feitas por arquitetos. Onde o conhecimento de como criar um produto competitivo é pouco ou nenhum.

3.3. QUADRO ETAPAS DO SCRUM

	SCRUM
ANÁLISE	• **Itens Product Backlog (PBI):** São as histórias de usuário, para ter um bom PBI primeiro deve-se identificar os objetivos e segundo passo coletar os requisitos. Devem ser feitos pelo Product Owner para descrevê-los para a equipe e então determinar quais itens farão parte da Sprint. • **Reunião de estimativa:** Prioridade ao backlog do produto, itens relevantes estimados, nesta etapa se define o objetivo da Sprint inicial.

PROJETO	As ferramentas do Scrum focam principalmente em automatizar o Kanban e suas interações. Abaixo citarei alguns relevantes: • **ScrumHalf:** Facilita a colaboração do time distribuído, fornece gráficos como *burn-down* automaticamente e relatórios, interação com *Twitter, Dropbox*, possuí um chat, gráfico de *Gant* para acompanhar o planejamento feito pela própria ferramenta e bem fácil a manutenção. • **PangoScrum:** Auxilia nas atividades principais do Scrum como *product backlog*, Sprint, agendamento e planejamento de sprints. Particularmente achei pouco burocrático o uso, e não fornece uma usabilidade tão favorável ao usuário, e não disponibiliza painel de tarefas. • **Trello:** Não propriamente do Scrum, somente automatiza o quadro Kanban, mas a usabilidade é o destaque da ferramenta, muito simples para sincronizar quadros, facilidade de uso da ferramenta é alta, muito versátil, e sua aplicação pode ser em variados fins.
CODIFICANDO	Definido Daily Scrums, que são as reuniões diárias que tem o objetivo de disseminar conhecimento de cada pessoa sobre o que foi feito no dia anterior, identificar os impedimentos e priorizar o trabalho a ser realizado no dia. Todos os membros deverão participar outras pessoas externas também poderão participar, porém poderão apenas ouvir. Os impedimentos levantados deverão ser resolvidos pelo Scrum Master o mais rápido possível.
TESTES	Dividida em dois grupos: Planejamento e execução. Planejamento realiza as tarefas como a definição de casos e procedimentos de testes. Execução: Executa os testes planejados e reporta as falhas ocorridas durante os testes.

Fonte: PHAM, Andrew.

3.4. VANTAGENS DE UTILIZAR XP

Cliente: O cliente sempre por perto resultando em um produto final bem próximo do que ele esperava. O Cliente decide prioridade no que deve fazer primeiro e o que pode esperar.

Programação em Par: A programação em duplas reduz o número de erros e aumenta a legibilidade do código, facilitando em manutenções.

Diminuição de bugs: Devido aos testes unitários e feito várias vezes por dia os erros são encontrados mais cedo.

Regra de negócio: Foca no negócio e é capaz de suportar à mudanças contínuas do mercado.

Equipe: Trabalho em conjunto entre equipe dos clientes e desenvolvedores proporcionando um trabalho em conjunto.

Adaptação: As histórias de usuários não são estáticas, pode-se acrescentar ou mesmo remover outras.

Feedback: Quanto mais rápido o feedback maior o aprendizado produzido por ele.

Simplicidade: Não construir complexidade desnecessária, desenvolver a solução mais simples que possa funcionar.

Mudança Incremental: Uma série de mudanças pequenas para resolver grandes problemas.

Documentação: O código é a documentação mais atualizada, descarte tudo aquilo que não irá utilizar mais ou que seja desnecessário.

3.5. DESVANTAGENS DE UTILIZAR XP

Produtividade: Perda de produtividade adotando programação em duplas, reduzindo de aproximadamente 50% do tempo segundo Medrado.

Maturidade: Exige maturidade e vontade de melhorar os processos de desenvolvimento, principalmente o apoio da alta direção e dos participantes, onde a equipe de clientes deve saber até onde podem ir e quais são seus direitos.

Requisitos instáveis: Os requisitos se modificam rapidamente a medida que os releases estão sendo implementadas e os clientes estão presentes acompanhando o desenvolvimento surge novas expectativas sobre a funcionalidade.

Disponibilidade: Requer reuniões constantes com cliente e por este motivo o cliente perde o foco das funcionalidades e passa a querer que a equipe foque em detalhes irrelevantes principalmente os de layouts.

Cultura: Requer muita mudança cultural para que se utilize o XP atendendo seus valores e princípios.

Equipe: Equipe deve haver no máximo 12 integrantes, mas o XP pode ser utilizado de forma escalável aumentando assim a comunicação entre a equipe.

Espaço físico: Como requer que todos estejam envolvidos, necessita de um espaço físico onde deixe todos os envolvidos próximos.

Atualização de versão: Pelo fato de lançar versões constantemente ao fim de cada release, é importante manter o cliente atualizado em relação as versões para que o mesmo não solicite alteração em uma versão que já está desatualizada.

3.6. QUADRO DE ETAPAS DO XP

	XP
ANÁLISE	• **Cartão de visão:** Feito pelos clientes ou stakeholders envolvidos com no máximo 25 palavras declarando a finalidade da função ou projeto, considerando pontos que implicam durante o ciclo de vida do projeto como riscos, tecnologia envolvida, equipe... • **Histórias de usuário:** Feito pelo cliente com menor quantidade de informações necessárias a fim de definir um caminho através do sistema, escrito em cartões começando pelo objetivo, onde cada cartão representar uma etapa do objetivo. As histórias de usuário são utilizadas durante todo o projeto. • Planejamento de releases e pequenas liberações, dividindo em ciclos, calculando a velocidade do projeto e reuniões em pé.
PROJETO	• **Simplicidade:** Preza por não adicionar nada fora do tempo, seja funcionalidade, documentações... • **Metáfora:** São descrições sem utilizar termos técnicos com o objetivo de guiar o desenvolvimento com maior transparência possível para o cliente. • **Cartões CRC:** Classe-Responsabilidade-Cartão que serve como mecanismo eficaz para pensar sobre o software em um contexto orientado a objetos, identificando e organizando as classes. CRC é o único artefato de projeto produzido no XP.
CODIFICANDO	• **Cliente sempre disponível:** A equipe de clientes na mesma sala dos desenvolvedores participando de cada iteração. • **Programação em pares:** Dupla utilizando a mesma máquina e trabalhando no mesmo código. • **Padrões:** Codificar de acordo com padrões acordados entre a equipe.

	• **Testes Unitários:** Antes de desenvolver cada etapa que envolve a iteração, desenvolver teste unitário primeiro. • **Integração:** Ao término de cada unidade ou etapa integrá-la ao projeto com frequência, permite que a equipe esteja em contato constante com o cliente. • **Código aberto:** Todos tem o direito de alterar partes do código do outro parceiro, o código é de propriedade coletiva.
TESTES	• **Unitários:** Todo o código deve ter os testes unitários, onde cada unidade deve ser testada antes de ser liberada. • **Aceitação:** É regra testes de aceitação com frequência.

Fonte: SBROCCO, Teixeira; MACEDO, Paulo.

3.7. VANTAGENS DE UTILIZAR FDD

FDD é recomendado para qualquer tipo de desenvolvimento onde o foco é características que fornecem valor ao cliente, onde a prioridade quem dá é o cliente. Os pequenos blocos de funcionalidades priorizadas pelo cliente prontos a cada duas semanas ou menos.

Por haver um planejamento mais detalhado os requisitos são mais formais e o guia construído na fase de concepção e planejamento faz-se uso até o final sem alterações relevantes. Isso faz com que forneça uma forma de saber se os planos e a estimativas são sólidos dentro dos primeiros 10% de construção do projeto.

Monitoramento em termos de negócio bem detalhado dentro do projeto, com resumos de alto nível para os Stakeholders, utilizando de rastreabilidade e relatórios.

3.8. DESVANTAGENS DE UTILIZAR FDD

A Universidade Federal Rural de Pernambuco – Unidade de Serra Talhada, publicou um artigo onde apontava algumas das desvantagens sobre a metodologia. Onde havia questionamentos sobre a eficácia e aplicabilidade do FDD, devido a formalização dos requisitos, a geração de UML e documentações. Contestação sobre o tamanho mínimo de um time, o que não se enquadra para empresas de desenvolvimento de pequeno porte, pois geralmente um time com quatro pessoas realiza as atividades com clareza, e alta produtividade, houve também questionamento sobre como manter

a metodologia no ambiente, para não entrar em desuso devido à grande demanda e o tempo de entrega.

3.9. QUADRO DE ETAPAS DO FDD

	FDD
ANÁLISE	• **Histórias de usuário:** Para quebrar as funcionalidades, quando necessário. Não obrigatório em todos os projetos, à partir das histórias se alimenta o FBS (*Feature Breakdown Structure).* • **Feature Breakdown Structure:** Decompor todo o modelo de domínio de forma funcional separando nas camadas de negócio, atividades e funcionalidades criando o roteiro de prioridades a ser desenvolvidas, o modelo de nomeação deve ser <***ação***><***resultado***><***objeto***>. • **Resultado:** Diagramas de classes focando no modelo de domínio e seus relacionamentos. Métodos e atributos identificados em cada classe, diagrama de sequência se houver, comentários pelo qual motivo foi escolhido daquela forma e quais alternativas foram analisadas para chegar no resultado.
PROJETO	• **Utilizar cores na UML:** Cores foram sugeridas por um dos criadores Peter Coad, Eric e Jeff de Luca. Onde adotaram quatro cores, são utilizados mais no diagrama de entidades e de classe. Onde o rosa representa um objeto que armazena informações temporárias. O amarelo representa um papel de algum envolvido, seja, por exemplo, o acesso ao sistema feito pelo administrador. O azul representa uma descrição, e o verde é algo tangível, unicamente identificável. • **Análise orientada a objetos:** Desenvolver requisitos elaborando casos de uso, modelagem de dados utilizando diagrama de classes e sequência, e classe de entidades para modelar comportamentos e informações que devem ser armazenados.

	• **Critérios:** Deve obter uma lista de áreas de negócio, pra cada uma dessas áreas uma lista de atividades de negócio e para cada atividade uma funcionalidade que satisfaça o passo. • **Modelar funcionalidade através da Mind Map Modeling (M3):** A modelagem pode ser feita utilizando M3 com UML. M3 é como se fosse um mapa mental do seu modelo voltado para gerir as informações, funciona como uma ferramenta de brainstorming facilitando a memorização e aprendizado. • **Verificar:** Realizar um auto avaliação e dos membros ativos que participaram na modelagem, pedindo aos especialistas confirmarem ou esclarecer questões que afetam a lista de atividades.
CODIFICANDO	• **Implementação:** Os indivíduos responsáveis por cada classe implementarem os itens necessários para satisfazer aos requisitos. • **Inspecionar:** É realizada uma inspeção no código com membros da equipe de funcionalidades, é feita antes ou após o teste de unidade. • **Versão:** Após a inspeção de código com sucesso e faz a integração da atividade ou unidade para o projeto inteiro.
TESTES	• **TDD:** Utiliza para manter a qualidade do software. • **Teste unitário:** Os proprietários das classes testam seus códigos para certificar que todos os requisitos de suas classes foram atendidos. O líder determina quais testes é necessário, isso é, quando há classes externas envolvidas.

Fonte: PRESSMAN, Roger S.

4. DEFININDO O PROCESSO

4.1. AMBIENTE DE DESENVOLVIMENTO

Antes de definir um processo de software e atribuir responsabilidades, é necessário configurar o ambiente de desenvolvimento, de centralizar as atividades realizadas, apresentando os processos, gerenciamento das mudanças realizadas no projeto, afim de acompanhar a evolução dos requisitos de perto por todos da equipe. Com o objetivo de alcançar benefícios, reduzindo o impacto sobre a qualidade do projeto, através do uso de ferramentas especializadas para gestão de mudanças.

Algumas ferramentas foram adotadas para o processo de gerenciamento de software, com o intuito de centralizar o processo de software, tanto suas documentações geradas, requisitos e até o código fonte. Foi criado um repositório, em servidor web utilizando a ferramenta do GIT e Tortoise Git para controle de versão. Foi criado diretórios para cada tipo de projeto, cada artefato que compõe o processo e o projeto, e adicionados ao controle de versão.

Foi adotado um ciclo de vida para os requisitos, tarefas e atividades que deveriam ser realizadas no projeto por alguma equipe ou membro. A primeira definição foi a criação de listas de correções, lista de erros e listas de tarefas. Essa estratégia abordada pela equipe surtiu com efeito muito positivo, porque um dos principais gargalos era, o que está sendo feito agora, por quem está sem feito e qual a situação da tarefa? Então, cada desenvolver tinha sua lista de tarefas com o seu nome, número de sequência, mês e ano em que foi iniciada **<Tarefas> <Nome do desenvolvedor> <00> <ano-mês>.** Um exemplo seria, Tarefas Jose 001 2016-08, como é mostrado na figura 4.
A lista era gerada pelo líder de projeto, que atribuía ao desenvolvedor suas atividades a serem realizadas, assim que determinada atividade era concluída, o desenvolver colocaria a data de conclusão e partiria para a próxima. Ao final da lista deveria renomear a lista utilizando o controlador de versão inserindo o nome **Finalizada.**

Essa abordagem foi adotada para todas as documentações do projeto, separadas por raízes dentro do diretório do SVN. Então era necessário que, no início do dia todos os integrantes fizessem as atualizações de seus diretórios, para que pudesse integrar as modificações realizadas durante o dia anterior pelos outros membros.

Outra ferramenta que foi adotada, foi o Redmine, esta ferramenta teve um nível médio de aceitação entre os integrantes. Alguns integrantes acharam que a ferramenta burocratizou o processo de desenvolvimento, pois estava realizando tarefas que contribuiria para perca de produtividade.

Até a fase de implantação, e realização das primeiras documentações técnicas do projeto, foi um período árduo. Era o momento de todos contribuir com um pouco do que sabia sobre cada funcionalidade existente no projeto e começar a escrever no Redmine. Durante um período de noventa dias, todos se empenharam parte do tempo para atualizar a documentação do projeto na plataforma. A partir desse período a medida que nova funcionalidade era desenvolvida, posteriormente era redigido a documentação na plataforma, de forma técnica, resumida e eficiente.

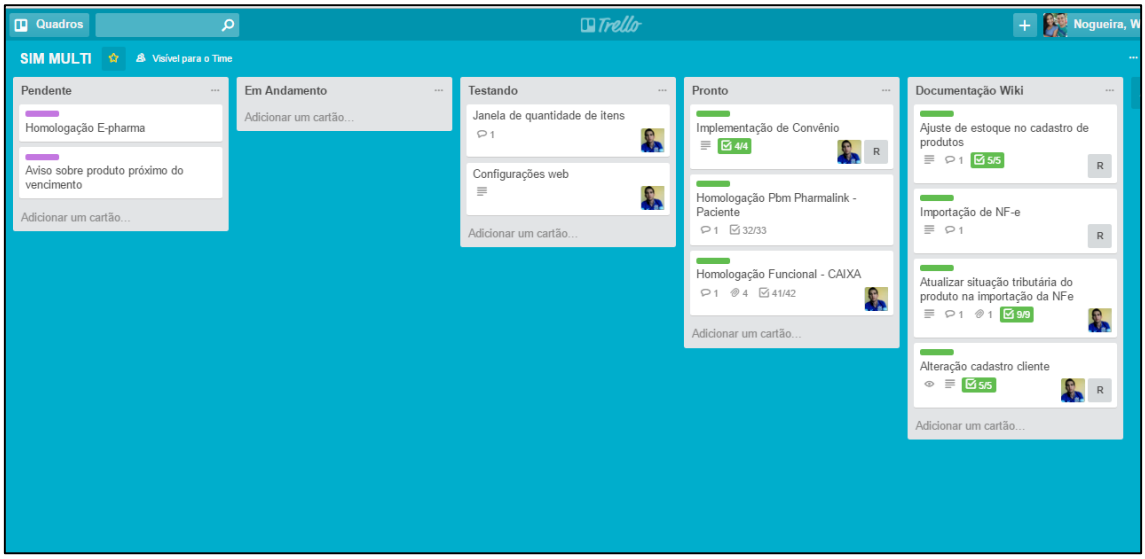
Hoje, com o uso da documentação na plataforma, os arquivos de requisitos, continuaram a serem desenvolvidos em documentos word. O documento word se mostrou mais preciso e manteve a produtividade por parte de quem os produzia. Cada arquivo continha requisitos de uma determinada funcionalidade que iria compor o projeto de software.

Os membros que tiveram um choque de cultura com o uso do Redmine, perceberam que, mesmo ainda com pouca informação e documentação sobre as funcionalidades é altamente produtivo no momento do desenvolvimento. Alguns dos feedbacks recebidos foi que, com o uso da documentação a produtividade reduz em torno de 80% a 90% do tempo que seria destinado ao estudo e entendimento da mesma.

Outra ferramenta utilizada foi o Trello, baseada no quadro Kanban do Scrum, a ferramenta permite gerenciar projetos em listas, que pode ser ajustada de acordo com as tarefas realizadas dos usuários. No quadro de tarefas, cada cartão representava uma estória de usuário e estava ordenada de acordo com as prioridades. As cores representam o status da funcionalidade, eram criadas checklist de atividades a serem realizadas dentro do cartão, para exibir o andamento da atividade.

A ferramenta foi depreciada com o uso das documentações no word, apesar de ser bastante intuitiva, limitava o uso para usuários com menor experiência. O ponto que levou a ferramenta ao desuso, era a criação de cartões onde necessitaria de imagens apontando erros de serem anexadas. Foi encontrado essa dificuldade no momento de listar cartões que representavam atividades a serem corrigidos, que era preciso anexar a imagem do sistema que apresentava o erro. Os cartões se apresentavam com bastante informação, e não permitindo a leitura de forma clara e ágil. Então, o uso da ferramenta foi interrompido pois a facilidade de uso não era tão alta para usuários que tivessem menos interação com o mundo técnico de TI.

Figura 4 – Ferramenta Trello.



Fonte: trello.com

4.2. DEFININDO UM PROCESSO

Definir um processo utilizando os três métodos ágeis, filtrando o melhor que cada um pode oferecer, nos permite criar um ciclo de vida mais robusto para o desenvolvimento de software baseado nas melhores práticas ágeis.

Se analisarmos o XP vemos que ele dá conta de todo o ciclo de desenvolvimento sem a necessidade do Scrum e FDD, porém para ser verdade necessitária de alta maturidade da equipe e bastante experiência nas práticas do XP.

Como o Scrum é um framework, para gerenciamento de projetos é uma prática ideal. Utilizar outras metodologias como FDD para as práticas de engenharia de software.

O primeiro item do Scrum é criar um Product Backlog, a fase de concepção e planejamento do FDD pode criar o nosso Backlog de produto.

A solução é então criar modelo interativo e incremental onde o planejamento, modelagem, desenvolvimento e testes acontecem em todas as etapas de cada Sprint resultando em um release funcional para o cliente. Para isso é necessária uma exploração evolutiva com feedbacks constantes, criando condições para que eles aconteçam de forma dinâmica, onde a adaptação às mudanças deverá ser amadurecida durante todo o ciclo de vida da equipe, que tem por desafio auto se organizarem para que reduza a multitarefa. O processo deve ser simples, ter apenas o que será utilizado. Na prática, é resultado de uma grande mudança de cultura.

4.3. REQUISITOS – VISÃO DO PROJETO

Começar o processo pelo levantamento dos requisitos, onde no Scrum seria a etapa chamada visão do projeto. Para esta etapa, foi definido a utilização da técnica do XP Cartão de visão onde será formado um conjunto de estórias, e delas extrair os itens de backlog. Deverá ser feito preferencialmente em um local fora do escritório, deve conter os líderes de equipe que participarão de alguma forma do desenvolvimento do projeto, inclusive time de desenvolvimento.

É de grande importância às presenças dos principais líderes que compõe a equipe, pois adquire o entendimento do escopo facilitando o desenvolvimento, evitando assim pausas geradas a partir de dúvidas sobre o processo.

As estórias de usuários serão feitas em cartões, conterão informações descrevendo de forma clara e objetiva, inicia-se pelo objetivo que deverá ser atingido e, ao final deverá ser atribuído a cada cartão sua prioridade, representado em forma de números. Cada cartão representará uma funcionalidade, ou até mesmo, subpartes da funcionalidade.

Segundo COHN, deve estender o tempo necessário, para que toda a equipe tenha o entendimento de cada funcionalidade que deverá conter no projeto. Nada impede que os cartões sofram alterações, ou até mesmo sejam excluídos da pilha de cartões.

Highsmith cita o modo como se comportam, de antecipar buscando compreender os limites e minimizar os erros o quanto for possível. Nesta etapa, cada cartão deve ser refinado e discutido por todos os presentes, cada cartão pode ser reescrito a qualquer momento, à medida que a equipe for adquirindo novo conhecimento e, consequentemente novas expectativas. Os membros da equipe irão atribuir custo a cada uma delas, com base no tempo que será necessário para o desenvolvimento.

Os cartões serão ordenados de acordo com as prioridades, apontadas pela equipe de cliente. Os cartões serão utilizados durante todo o ciclo de desenvolvimento do projeto, é de grande importância que os cartões do topo da pilha, estejam representando a funcionalidade na qual se agrega mais valor ao cliente.

4.4. DESENVOLVER UM MODELO – VISÃO DO PRODUTO

Os cartões serão vistos como requisitos funcionais do projeto, o planejamento será feito a partir do primeiro cartão. Para aperfeiçoar a ideia abrangente capturada nas estórias de usuário, utilizará os conceitos de engenharia de software praticados na metodologia FDD.

A fase de concepção e planejamento, trata os requisitos desenvolvendo um modelo abrangente, com análise orientada a objetos, com objetivo de identificar os conceitos mais

importantes do sistema, abstraindo um modelo que sirva de referência para toda a análise que virá pela frente. PRESSMAN adverte que, essa análise não pode ser vista como verdades absolutas que não devem ser alteradas. A equipe deve estar preparada para as mudanças que venham a ocorrer.

Criar os fluxos de processo, para que facilite a visualização de todo o processo que o sistema/funcionalidade irá transitar até atingir o seu objetivo, o fluxo do processo pode ser representado pelo diagrama de sequência, porém fica a critério da equipe. De forma que facilite o entendimento para ambas as equipes, de projeto e clientes.

Então, neste ponto, todos já devem ter o entendimento do fluxo de cada atividade junto ao cliente. Fica a critério da equipe e da maturidade no processo, para decidir por começar pelo diagrama de classe, onde o refinamento das responsabilidades, atributos e operações envolvidas. É comum esse refinamento nos levar a descoberta de novas classes relacionadas, com atributos e suas operações.

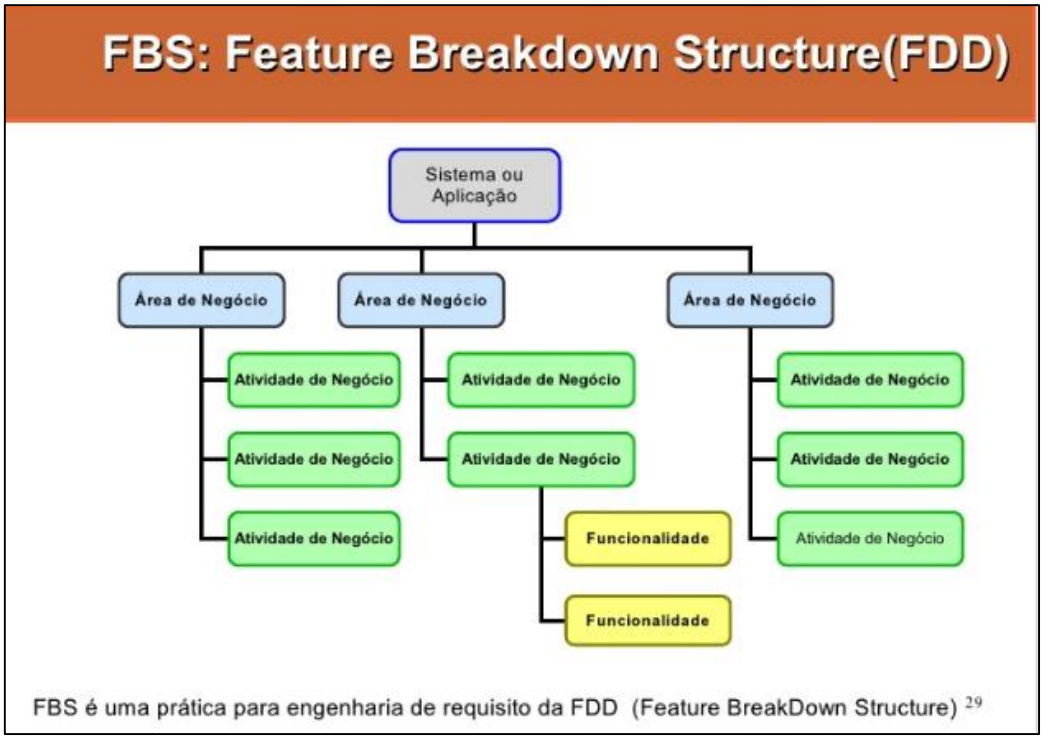
Não se enxerga a criação de diagramas apenas como documentação, mas há alguns ganhos que obtêm com ela, como a facilidade de reutilizar, entender e manter o software. A vantagem, é poder pensar em uma solução sistêmica focada em responsabilidades antes de começar a desenvolvê-la, ter uma visão contemplando os problemas que querem resolver traçando estratégias isoladas ou abrangentes, identificando a responsabilidade sobre o conceito determinado.

4.5. CONSTRUIR LISTAS DE FUNCIONALIDADES – VISÃO DO PROCESSO

Próximo passo é definir uma lista de funcionalidades decompondo todo o domínio, no Scrum essa etapa é definida como Product Backlog, esse trabalho pode ser feito em reunião com gerente de projetos e desenvolvedores-líderes.

Antes de utilizar qualquer mecanismo ou ferramenta que pudesse auxiliar nesta etapa, em reunião, o objetivo da equipe era identificar e decompor o conjunto de funcionalidades. A decomposição funcional é definida a partir da divisão do domínio, em áreas que englobam regras de negócio, que por sua vez também são decompostas em formato de árvore, onde cada ramo é uma funcionalidade pertencente à regra. Um mecanismo adotado para gerar a árvore foi o FBS, que apresentou uma alta abstração do domínio e entendimento de cada serviço e funcionalidade. Cada funcionalidade pode representar uma classe do projeto, se assim for, realiza o diagrama de classe dessa atividade e de suas dependências.

Figura 5 - Estrutura Analítica do FBS.



Referência: http://www.heptagon.com.br

Criar estrutura analítica de funcionalidades utilizando FBS é uma prática de engenharia de requisitos, que é citada na metodologia FDD. Mitigando o domínio nas camadas de áreas de negócio, atividades de negócio e funcionalidades granulares onde não levará mais do que duas semanas para ser completada.

Neste momento as funcionalidades se encontram granulares, a ponto de não levar mais do que duas semanas para ser concluída. Se a atividade parecer maior do que duas semanas, então deve ser quebrada em atividades menores, que então se tornam funcionalidades.

4.6. PLANEJAMENTO DAS FUNCIONALIDADES

Para o planejamento foi mesclado técnicas de engenharia de software proposta pela metodologia FDD, e técnicas XP para obter dinamismo. É feito uma reunião no início de cada semana, para priorizar as tarefas, onde identifica o roteiro de tarefas que são prioridades.

Todos os alicerces citados na metodologia são altamente explorados, principalmente a comunicação, todo ponto de vista é ouvido e discutido, a princípio não era uma prática da equipe conversar muito, expor as ideias, mas com o hábito e a pratica a comunicação foi sendo mais efetiva e assertiva. É essencial para que não haja lacunas em processos e procedimentos a serem tomados entre a equipe, reduzindo os problemas.

É considerado a sequência de desenvolvimento, o engajamento das atividades aos desenvolvedores, as classes principais e para quais desenvolvedores. As atividades obrigatórias desta etapa são: Determinar uma sequência de desenvolvimento, atribuir as atividades aos programadores.

A equipe deve atribuir uma data para término de cada atividade, baseando-se nas dependências entre as funcionalidades, distribuição da carga de trabalho entre os programadores, complexidade, priorizar atividades de alto risco.

Ao final realiza-se uma auto-avaliação dos envolvidos em relação a participação ativa de cada participante da equipe. E então, é definido a lista de tarefas e seus respectivos desenvolvedores.

4.7. DETALHAMENTO POR FUNCIONALIDADE

A funcionalidade a ser desenvolvida, refina-se todos os requisitos, documentações disponíveis, esboços de interfaces e qualquer outra fonte que ofereça suporte. É uma atividade de critério opcional, na maior parte dos casos, este momento deixa de ser realizado, dependendo da complexidade da funcionalidade e o quão a equipe conseguiu internalizar o problema.

O líder de equipe então prioriza as atividades, atribuindo as listas de tarefas de cada integrante responsável por sua tarefa, e assim submetendo ao sistema de controle de versão para ser compartilhado entre os membros da equipe. Diferente de como é abordado na metodologia FDD, as classes não têm dono. O código fonte é aberto a todos da equipe, assim permite a interação e comunicação com outros membros em necessidades de integrar força de trabalho.

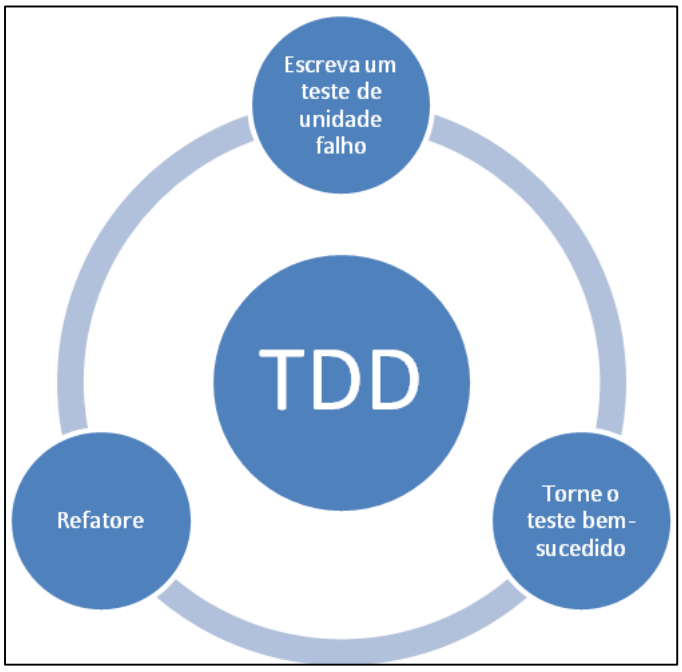
4.8. DESENVOLVIMENTO

Para o desenvolvimento algumas práticas do XP e também do FDD foram mescladas para realização do trabalho. A primeira premissa vem do XP, que é o desenvolvimento guiado a testes, utilizando o conceito TDD.

Para o uso do TDD foi definido um ciclo de desenvolvimento para garantir a qualidade e também a produtividade. Na prática, aplicando o TDD não é simplesmente criar um teste para a funcionalidade, e sim é necessário criar um projeto a parte exclusivo para os testes automatizados. Dessa forma, a estrutura de classes do projeto fica bem organizada e fácil de manter.

O conceito para o ciclo de vida do TDD é bem claro: Teste não passando, adiciona funcionalidade nova, teste passando e próximo. Porém essa prática não se encaixou perfeitamente dentro do projeto.

Figura 6 - Ciclo tradicional do TDD.



Fonte: http://tdd.caelum.com.br

O motivo de já existirem funcionalidades prontas, então foi trabalhar o TDD de forma reversa, terminologia adotada entre a equipe. Que consistia em criar testes, diversas variações de testes para o mesmo requisito. Outro motivo foi pensando na reutilização de código, e como haveria diversos testes para o mesmo requisito, foi criado os Mocks, que consistia em criar e simular um ambiente para determinado funcionalidade. Então foi criado um novo projeto, dentro da mesma solução destinado somente a testes, como mostra na figura 5.

Um exemplo simples de um teste unitário fictício: Funcionalidade de ler nota fiscal, então criou-se um mock que montava todo o ambiente necessário na base de dados para possibilitar a simulação real de uma nota fiscal. Sabendo que, teria que utilizar o serviço que a própria aplicação utiliza em ambiente de produção. Então criou-se uma classe destinada somente a aquele tipo de teste, e nela foram criadas suas variações.

Então para requisitos que ainda não existia mocks preparados, é criado primeiro o serviço, quanto os métodos na camada de acesso a dados que farão a consulta, o serviço está funcionando conforme é especificado em documentação.

A classe que monta o ambiente para ser testado dentro de um teste, é denominada Mock. Como os mocks faz uso dos serviços que compõe a funcionalidade, então é necessário que o serviço já esteja pronto e disponível para ser consumido pelo mock, e consequentemente ser consumido pelos testes.

Algumas práticas XP não foram adotadas, uma delas foi a programação em pares, devido ao tamanho da equipe por ser pequena, então enquanto foi definido que é mais produtivo cada desenvolver focar na resolução de cada atividade. E o pilar é priorizar três fatores que apontamos como pilares para o bom desenvolvimento a comunicação, priorização e entrosamento.

Quanto menos burocratizar o processo de comunicação, melhor será a produtividade e trará resultados positivos e que agregaram valor à equipe e ao projeto.

A princípio, como o projeto ainda não tem um cliente especifico, o diretor da empresa faz o papel do cliente, pois ele é o maior interessado, e é dele que parte todas as regras de negócio e requisitos. Então, ele está sempre presente na empresa, bem entrosado com toda a equipe e sem burocracia, pode ser chamado a qualquer hora de forma rápida e ágil.

As reuniões diárias do Scrum, está sendo feito semanalmente, ao iniciar do expediente na segunda feira, na reunião é feito um debate do que cada um fez na semana, como foi o desenvolvimento na determinada tarefa, qual a situação da atividade, recebe e passa feedbacks para propor melhorias, é definido metas a serem atingidas durante o decorrer da semana. As reuniões são feitas em pé, e duram cerca de 15 a 20 minutos.

Pequenos releases é uma característica do XP, é feito durante o decorrer da semana, ao final de cada atividade, contando com os casos de testes desenvolvidos. Hoje já existe o projeto, e constantemente está sendo desenvolvido novas funcionalidade e correções para atender e se adequar às mudanças de requisitos. Então toda alteração feita, é utilizada o controlador de versão, e é criado um ramo para sua máquina, onde é feito os commits de forma local, assim que toda a alteração for feita e testada manualmente, e criado os casos de testes ou ajustados os já existentes, é feito uma união do seu projeto com o projeto atual em produção, assim disponibiliza para os outros desenvolvedores a nova funcionalidade ou correção.

Integração contínua, é uma característica do XP, após a implementação dos testes automatizados, a IDE Visual Studio, permite fazer o build da aplicação de forma automática. Então toda a versão de produção é publicada de forma automática, para isso é necessário que todos os testes automatizados estejam rodando.

A integração contínua do XP aborda a integração de códigos fontes diversas vezes por dia, para atender essa característica, no controle de versão criar o chamado de branch para cada módulo a ser desenvolvido, e logo após testado integrar com o diretório raiz, onde fica a master do código. Assim, mantém a integridade do código fonte em produção, e constantemente durante o dia vai fazendo os commits e após tudo testado, faz o merge com a raiz.

5. APLICAÇÃO REAL DO PROCESSO

O processo foi testado em duas funcionalidades pilotos, que serviram como base para a implantação, e adaptação e amadurecimento à nova cultura de desenvolvimento de software. Informações gerais sobre as funcionalidades piloto.

Características	**Funcionalidade 1**	**Funcionalidade 2**
Nº de integrantes	4 desenvolvedores e 1 líder de projeto	7 desenvolvedores, 1 líder projeto e 1 membro equipe de cliente
Duração do Projeto	8 meses	4 meses
Entregas Sprints	2-3 semanas	2-3 semanas
Objetivo	Automatização de Paf-ECF.	Automatização de convênio Farmácia Popular.

5.1. FUNCIONALIDADE PILOTO I

5.1.1. DESVANTAGENS APONTADAS NO PROCESSO

- Resistência em gerar estórias de usuários; os cartões não continham todas as informações necessárias que retratava a real necessidade. O processo foi repetido diversas vezes para chegar no ponto adequado.
- Nas reuniões de planejamento, foi levantado requisitos que não havia sido citado nos cartões de história. Então foi necessário, alterar o conteúdo de cartões.
- Resistência ao gerar documentações, houve controvérsias em relação a gerar o diagrama de classes e modelo. O motivo foi a redução na produtividade, e os diagramas sofreram modificações ao final da funcionalidade.
- A equipe não havia experiência com TDD, então para construir os testes que iriam abranger todas as possíveis variações de casos para essa funcionalidade tomou mais tempo do que havia planejado.
- Resistencia em gerar documentação técnica da funcionalidade na ferramenta Redmine.
- Resistência ao detalhar a funcionalidade e seus objetos, o motivo dado foi, a funcionalidade já estava bem internalizada entre a equipe e não havia necessidade. Essa fase não foi considerada como importante.
- Criação da lista de funcionalidades utilizando o conceito FBS, houve dificuldade entre as partes para separar atividade de negócio e área de negócio, dificuldade em identificar o que seria a real funcionalidade dentro de árvore.
- Criação de casos de usos que não foram utilizados como apoio à documentação.
- Funcionalidade gerou bugs relacionado a regra de negócio da atividade.

5.1.2. VANTAGENS APONTADAS NO PROCESSO

- Maior entendimento e clareza do problema para toda equipe;
- Requisitos disponíveis nas documentações e também no Redmine, foi considerado como aumento de produtividade e visto como recurso muito importante e essencial.
- Houve uma melhora na comunicação, algumas barreiras foram quebradas. Como não havia esse costume, as reuniões não eram tão interativas e dinâmicas.
- Foi possível mensurar um prazo mais próximo da realidade para o desenvolvimento da funcionalidade. Porém, o prazo foi estourado ultrapassando 48 h do previsto.
- Tarefas organizadas em listas obteve uma visão mais ampla em que se encontra a equipe com o que havia sido planejado.
- Boa aceitação da ferramenta Trello, pela facilidade de uso e interatividade.

5.2. FUNCIONALIDADE PILOTO II

Para o desenvolvimento da funcionalidade piloto II, as desvantagens apontadas na funcionalidade I foram tratadas.

5.2.1. DESVANTAGENS APONTADAS NO PROCESSO

- Foram necessárias mais de uma reunião, para levantar as histórias de usuário.
- Apontou como redundante as informações das listas de tarefas de cada desenvolvedor, com as informações presentes no Trello. Apontou que, por maior familiaridade com o documento Word, foi definido que o uso da ferramenta Trello não seria mais utilizada.
- Na criação de diagramas, foram criados somente diagramas de classe e de entidade relacional. O Diagrama de caso de usos não foi produzido, o que gerou conflito entre membros da equipe, defensores dos casos de usos.

5.2.2. VANTAGENS APONTADAS NO PROCESSO

- Criação das estórias de usuários abrangeu todos os requisitos da funcionalidade.
- As reuniões de planejamento das funcionalidades foram realizadas, houve maior comunicação e entrosamento da equipe, os membros da equipe mostraram-se mais confiantes e encorajados.
- Ao detalhar a funcionalidade a equipe obteve mais facilidade para criar a árvore de funcionalidade.
- A funcionalidade não gerou bugs originados de regra de negócio, houve correções, porém, correções feitas em interface.

• Os testes automatizados foram criados com mais facilidade, e a equipe se sentiu mais confiante e segura com os testes, foram vistos como muito importante manter todos os testes atualizados e rodando.

• Criação dos diagramas de classe foram vistos como artefatos importante e relevantes para documentação.

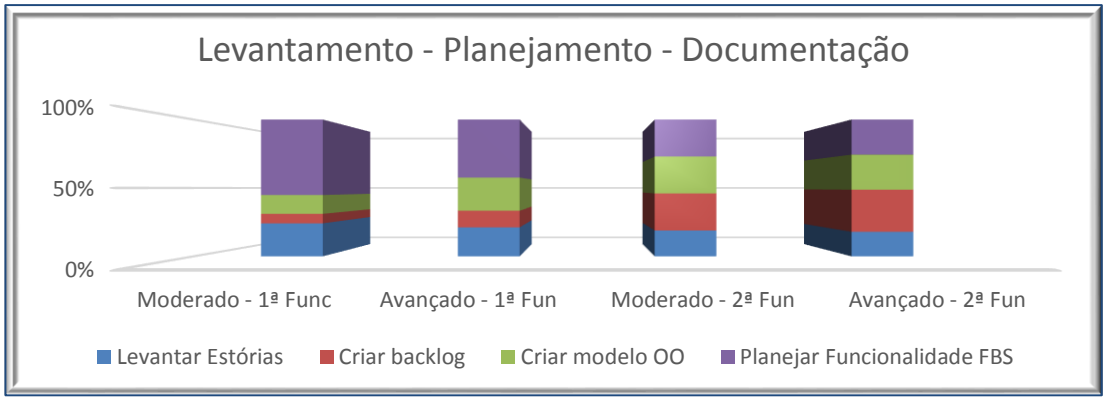
• Estimativa de prazo para conclusão da funcionalidade, foi assertiva. A equipe finalizou a funcionalidade no último dia da segunda semana. Foi apontado a documentação descrita na lista de funcionalidade como artefato de alto valor, que contribuiu para o apoio no decorrer do desenvolvimento.

• Documentação criada no Redmine não vista como redução na produtividade, e sim ganho de produtividade.

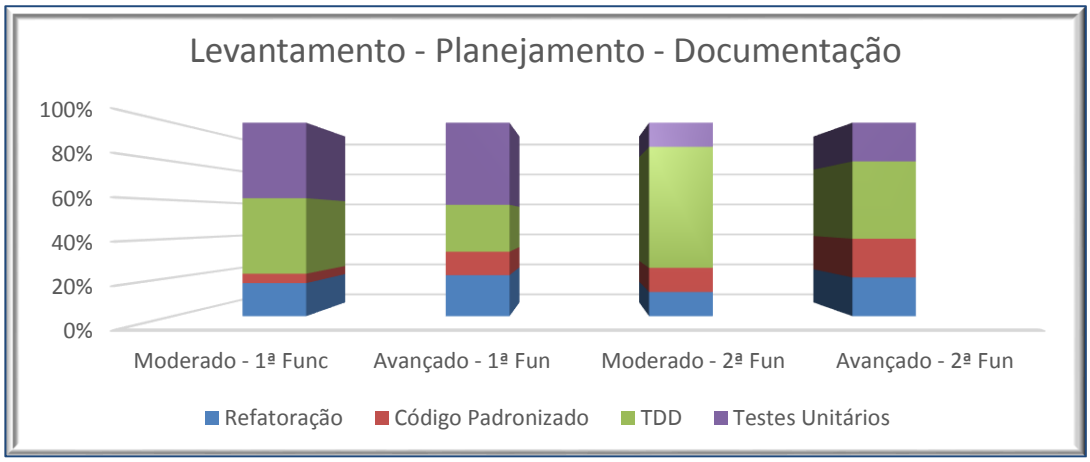
• Por não ter produzido os casos de uso, foi verificado que os mesmos não eram tão relevantes quando tinham em mãos as funcionalidades bem detalhadas no FBS e também o diagrama de classes. Neste momento ficou definido que não seria mais produzido os casos de usos.

5.3. APRENDIZADO DA EQUIPE

Para analisar o ganho em aprendizado, foi levantado o grau de dificuldade dos participantes do processo, em aprender a aplicar as técnicas organizacionais propostas no processo. Essa primeira análise, mostra o entendimento sobre as técnicas de levantamento de requisitos, até a parte de planejamento por funcionalidades. A equipe de planejamento, sentiu mais dificuldades nas técnicas de engenharia de software, e nas características da metodologia FDD. Para se orientar no gráfico, o índice moderado simboliza que a equipe não teve dificuldades que impedisse a equipe de não realizar a tarefa. No índice avançado, foram as atividades que foram necessárias mais horas estudando como adotar a técnica e definindo boas práticas.



Analise o desempenho da equipe técnica, aplicando as técnicas propostas pela metodologia XP.



Resultados obtido nos aspectos organizacionais, entre a equipe técnica, cliente, equipe de planejamento e testes:

Atributos	**Funcionalidade 1 (média)**	**Funcionalidade 2 (média)**
Comunicação	94%	98%
Entrosamento	60%	96%
Confiança	80%	100%
Satisfação (cliente)	97%	100%

Resultado obtido na qualidade do código-fonte e padrões de projeto:

Atributos	**Funcionalidade 1 (média)**	**Funcionalidade 2 (média)**
Testes Automatizados	70%	96%
Padrões de projeto	93%	99%
Código-fonte	92%	98%
Novas Tecnologias	90%	92%

Aprendizado da equipe para o estudo de novas tecnologias, conceitos e padrões ficaram mais constantes a partir da funcionalidade 2, pois vendo a importância do uso a motivação em fazer melhor foi sendo gradativa por parte da equipe, em busca de melhorar os seus processos.

6. RESULTADO APÓS USO DO PROCESSO

O intervalo de tempo entre o início da funcionalidade 1 e o final da funcionalidade 2 foram cerca de um ano. Este processo, não foi proposto dessa forma, durante esse tempo todo

o processo foi remodelado, testado, muitas características relevantes foram deixadas de lado, e outras abraçadas profundamente.

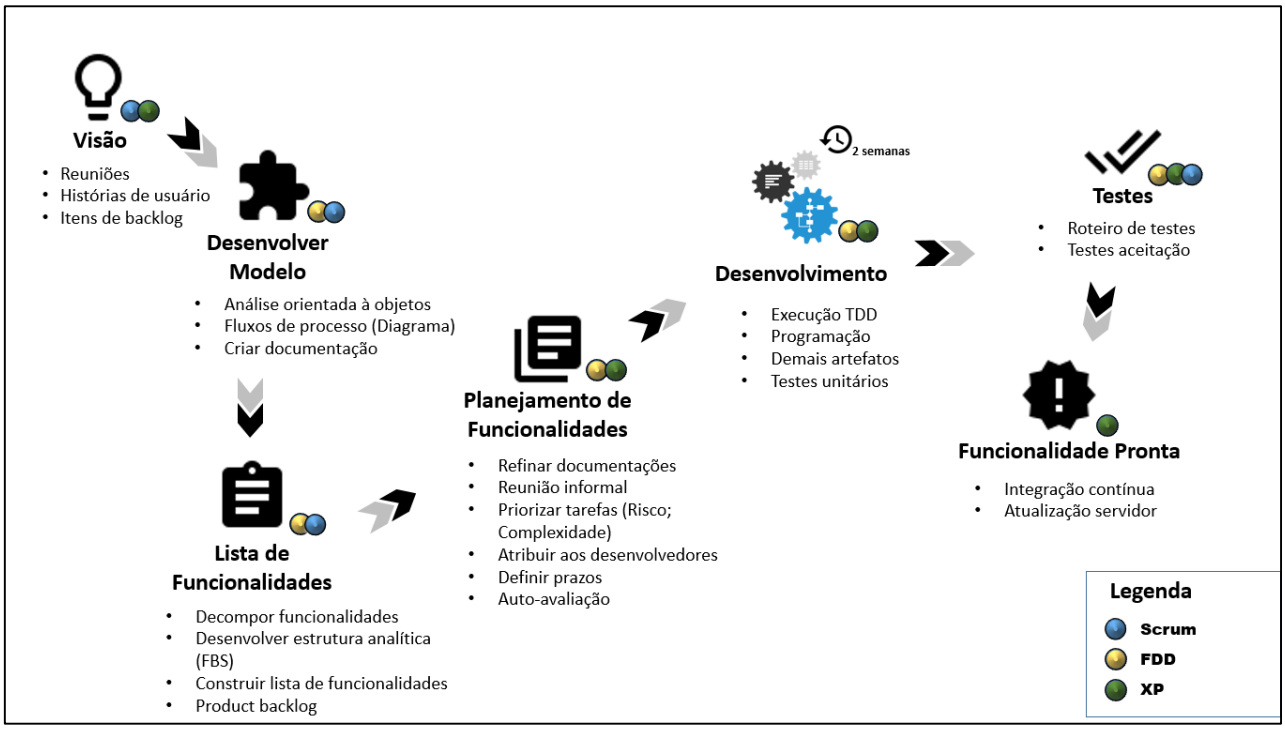
Colocar em prática este processo, foi apontado como uma das principais atividades a ser realizada, pois todos sabiam o quanto era importante e o quanto era difícil de aplicar. É claro que, há várias outras características que podem ser bastantes relevante, e com muito poder de gestão que não está sendo citada no processo.

O maior desafio foi as pessoas envolvidas, algumas viam o processo como uma boa alternativa para melhorar a produtividade da empresa e enriquecer o desenvolvimento. Por outro lado, outros viam como barreiras, que iriam impedir de realizar suas atividades, que iriam reduzir sua capacidade de gerar features novas.

Durante esse tempo, entrou e saiu ferramentas, testou características que a princípio seriam adotadas. Uma dessas características foi a famosa reunião diária em pé do Scrum, para o nosso time não houve tanto retorno dessa prática, pois com as listas de tarefas a disposição da equipe, todos sabiam claramente o que estava sendo feito. Então as reuniões foram substituídas para um período semanal, onde era discutido o objetivo a ser alcançado na semana, o que havia de ser melhorado, feedback em geral da equipe, com tempo de duração de 15 a 20 minutos.

Como resultado foi criado o fluxograma do processo obtido como resultado deste trabalho. Na representação abaixo, em cada etapa do desenvolvimento mostra conforme a legenda qual metodologia, e qual característica foi utilizada. As características utilizadas, não estão sendo utilizadas puramente no processo, sofreram adaptações para se adequarem ao cenário.

Figura 7 – Processo de Desenvolvimento de Software Ágil Wagner.



7. CONCLUSÃO

A adoção de métodos ágeis é uma alternativa encontrada para acompanhar o ritmo de desenvolvimento, trazendo para a equipe maior entrosamento, mais confiança ao executar planejamentos, e elevando a confiança do usuário final ao extremo. Neste trabalho foram apresentados os resultados obtidos no estudo de caso, que nos mostraram claramente, o efeito positivo na aprendizagem, melhoria de processos técnicos e organizacionais, a satisfação do cliente em receber entregas rápidas funcionais, com confiança e agilidade.

Este trabalho gera uma contribuição imensurável não só para Infopharma, mas também para outras organizações e empresas, sejam elas públicas e privadas que tem o cenário semelhante ao apresentado no estudo de caso, quanto para academia, pois relata os resultados observados.

O processo para a Infopharma surtiu com efeitos positivos, pois a confiança entre os envolvidos, a qualidade de desenvolvimento, a comunicação e relacionamento entre os membros tiveram uma alta elevação comparada como era antes do processo.

O objetivo é manter esse processo atualizado, e principalmente, segui-lo. Hoje na empresa, não se cria mais nenhuma funcionalidade sem antes criar o seu teste automatizado,

itens de documentações necessárias para compor o projeto, pois foram vistos o quão são benéficos para a equipe.

O desenvolvimento da Infopharma atualmente se encontra no termo, orientado a testes. Pois a confiança nas funcionalidades se baseia nos testes automatizados, com a cultura de sempre confiar nos testes prontos, algumas funcionalidades que sofreram alteração, e levarão a quebrar alguns testes, os erros foram identificados nos testes.

8. REFERÊNCIAS

COHN. Mike. **Desenvolvimento de Software com Scrum**: aplicando métodos ágeis com sucesso. Porto Alegre: Bookmam, 2011.

HIGHSMITH, Jim. **Gerenciamento Ágil de Projeto**: criando produtos inovadores. 2.ed. Rio de Janeiro: Alta books, 2012.

PHAM, Andrew. **Scrum em Ação**: gerenciamento e desenvolvimento ágil de projetos de software. São Paulo: Novatec Editora, 2011.

PRESSMAN, Roger S. **Engenharia de Software**. 6ª ed. McGraw-Hill, 2006.

SMITH, Preston G. MERRITT, Guy M. **Proactive Risk Management**: controlling uncertainty in product development. New York: Productivity Press, 2002.

[RUP2007] Rational Unified Process; Direitos Autorais (C) IBM Corporation 2000, 2007.

[UML2010] Site oficial da Linguagem de Modelagem Unificada, em www.uml.org acessada em 04/05/2016.

COHN, M. **User Stories Aplied**. Massachusetts, Addison Wesley (2004)

SBROCCO, Teixeira; MACEDO, Paulo. **Metodologias Ágeis:** Engenharia de Software Sob Medida. 1. ed. Érica, 2012.

NEGRINI, Célia. **Teste de Software no Scrum,** em www.devmedia.com.br/teste-de-software-no-scrum/28509 acessada em 11/10/2016.

PIMENTEL, Manoel. **Planejando seu Projeto com XP,** acessada em www.devmedia.com.br/planejando-seu-projeto-com-extreme-programming-parte-i/4273 acessada em 14/10/2016.