REVISTA DA JORNADA DA PÓS-GRADUAÇÃO E PESQUISA - CONGREGA
ISSN: 2526-4397 1982-2960

ANALISE COMPARATIVA DAS METOLOGIAS ÁGEIS: SCRUM, XP E FDD

COMPARATIVE ANALYSIS OF AGILE METHODS: SCRUM, XP E FDD

Fabio Josende Paz[1], Fernanda da Silveira Duarte[2], Enivaldo Soares Bigão[3]

RESUMO: As organizações estão sofrendo com a rápida evolução das tecnologias de informação e a necessidade de sistemas computacionais mais complexos, porém a agilidade no desenvolvimento destas aplicações torna-se uma necessidade, nesse contexto os métodos ageis de desenvolvimento surgem com uma boa alternativa. Existem no mercado vários métodos que utilizam a abordagem agil e seguem seus principios, apresentando muitas semelhanças no processo de desenvolvimento. Este artigo tem como objetivo avaliar e comparar as similaridades e diferenças nas metodologias ágeis empregadas pelos processos *extreme programming* (xp), scrum e *feature driven developent* (fdd) a fim de auxilar na escolha do método que melhor se adapte a realidade do desenvolvedor e sua equipe. Para alcançar os resultados foi realizada uma pesquisa exploratória, descritiva, comparativa e qualitativa. E como resultados foram apresentados às similaridades e diferenças das metodologias, auxiliando no entendimento e na escolha de uma metodologia ágil.

Palavras-chave: analise comparativa; desenvolvimento de software; métodos ageis;

ABSTRACT: *Organizations are suffering from the rapid evolution of information technologies and the need for more complex computer systems, but the agility in the development of these applications becomes a necessity, in this context the developmental methods come with a good alternative. There are several methods in the markets that use the agile approach and follow its principles, presenting many similarities in the development process. This article aims to evaluate and compare the similarities and differences in the agile methodologies used by the extreme programming (xp), scrum and feature driven developent (fdd) processes in order to assist in choosing the method that best suits the reality of the developer and its team. To achieve the results an exploratory, descriptive, comparative and qualitative research was carried out. And as results were presented to the similarities and differences of the methodologies, helping in the understanding and the choice of an agile methodology.*

Keyords: comparative analysis; software development; ageis methods;

INTRODUÇÃO

Historicamente existem conjuntos de metodologias que vieram dos sistemas de informação com uma abordagem técnica aos processos de negócio. São metodologias orientadas para a especificação de *software* e que fornecem modelos rigorosos do ponto de vista do *software*. Entretanto, são modelos de difícil assimilação (PINHEIRO, 2004).

Atualmente a melhoria contínua é de suma importância para as organizações que buscam se manter competitivas no mercado. A adoção desta prática promove a evolução constante e melhoria dos processos da empresa (PAZ E KIPPER, 2015). Nota-se que as empresas estão enfrentando vários problemas referentes da rápida evolução da Tecnologia da Informação (TI) e do descompasso causado pelo atraso na introdução dessa evolução nos modelos de gestão de negócios. As maiores dificuldades estão na criação de um ambiente operacional no qual a rapidez do processo decisório e o desempenho organizacional são essenciais para a agilidade nos serviços (OLIVERA *et al.*, 2012).

Nesse contexto, evidencia-se a dependência das organizações na indústria do software, onde os problemas relacionados ao processo de desenvolvimento de sistemas: alto custo,

alta complexidade, dificuldade de manutenção, e uma disparidade entre as necessidades dos usuários e o produto desenvolvido (SOMMERVILLE, 2011).

De acordo com Bernardo e Kon (2008), controlar a qualidade dos sistemas é um grande desafio devido à alta complexidade dos produtos e às várias barreiras relacionadas ao processo de desenvolvimento, que envolve questões humanas, técnicas, burocráticas, de negócio e políticas. Idealmente, os sistemas de *software* devem não só fazer corretamente o que o cliente precisa, mas também fazê-lo de forma segura, eficiente, flexível, de fácil manutenção e evolução frente aos contextos sociais e econômicos.

Nos modelos tradicionais, o foco é maior na análise de requisitos do *software* e projeto do que no desenvolvimento e testes. Sendo assim, em um cenário de um sistema de pequeno ou médio porte, a mudança repentina de requisitos causa mudança na especificação e no projeto, fazendo que voltem a “estaca zero”, ou seja, um grande retrabalho (AUGUSTO, 2014). Um aspecto importante de ser destacado é que a flexibilidade passou a ser um imperativo para lidar com a realidade contemporânea nas organizações. (VALLE; OLIVEIRA, 2009). Portanto, os métodos ágeis tornam-se imprescindíveis para produzir softwares na velocidade e qualidade que o mercado exige.

Muitos métodos ágeis recomendam que todos os colaboradores de um projeto (programadores, gerentes, equipes de homologação e clientes) trabalhem monitorando a qualidade do produto todos os dias e a todo o momento, pois acreditam que prevenir defeitos é mais fácil e barato que identificá-los e corrigi-los (BERNARDO; KON, 2008).

Para verificar a pertinência deste estudo foi realizada uma pesquisa bibliométrica no portal de periódico Capes nos últimos cinco anos, o qual conta com mais de 38 mil periódicos nacionais e internacionais em seu banco de dados, as palavras utilizadas para pesquisa foram:

Extreme Programming: 534 artigos;

Scrum: 12.733 artigos;

Feature Driven Developent: 169 artigos.

Percebe-se que o assunto é muito pesquisado, porém em relação à abordagem do tema deste artigo que é a analise comparativa de metodologias ágeis, não foram encontrados artigos tanto com a expressão em português quanto em inglês, o que nos permite supor que o tema desta pesquisa tem uma enorme relevância, pois estes métodos são muito utilizados, porém a relação entre eles não foi ainda objeto de estudo. Neste contexto o objetivo deste trabalho será avaliar e comparar as similaridades e diferenças nas metodologias ágeis empregadas pelos processos *Extreme Programming* (XP), Scrum e *Feature Driven Developent* (FDD).

Este trabalho está organizado da seguinte forma: além desta introdução, a seção dois apresenta os aspectos conceituais que embasam o trabalho: métodos ágeis, XP, Scrum e FDD. A seção três apresenta a metodologia da pesquisa. Na seção quatro é discutida a comparação entre os Métodos Ágeis *Scrum*, *Extreme Programming* e *Feature Driven Development*, e por fim a seção cinco aborda as considerações finais do trabalho.

1.1 Métodos ágeis

Em 2001, um grupo de dezessete profissionais experientes na área de desenvolvimento de *software*, se reuniu para discutir que técnicas usar e que ações poderiam melhorar o desempenho de projetos nesta área (AMBLER, 2004). Durante a discussão o grupo percebeu que determinadas técnicas utilizadas e respeitadas durante a execução do projeto contribuíram para que estes dessem certo. O resultado deste conjunto de percepções foi o Manifesto Ágil (MANIFESTO, 2001). Esse manifesto traz um conjunto de ações e valores criados a partir da experiência de profissionais da área com o foco de melhorar o desempenho dos projetos de desenvolvimento de *software*, através de conceitos chave(LIBARD; BARBOSA, 2010; SOARES, 2015; MANIFESTO, 2001).

De acordo com Abrahamsson *et al.* (2002), para os métodos ágeis alcançarem seus objetivos, os mesmos precisam seguir tais linhas:

- Produzir a primeira entrega em semanas e alcançar *feedback* rápido e mais cedo;
- Criar soluções mais simples de modo que se houver mudanças que haja mais facilidade e menor volume de alterações a serem feitas;
- Melhorar continuamente a qualidade do projeto, fazendo com que a iteração seguinte tenha menor custo de implementação;
- Testar constantemente, para detectar defeitos mais cedo e removê-los com menor custo.

Para Soares (2015), uma característica das metodologias ágeis é que elas são adaptativas ao invés de serem preditivas, ou seja, elas se adaptam a novos fatores decorrentes ao invés de procurar analisar previamente tudo o que pode acontecer no decorrer do desenvolvimento. Para o autor, essa análise prévia é difícil e apresenta alto custo, além de tornar-se um problema caso não se queira fazer alterações nos planejamentos. Por exemplo, para seguir estritamente o planejado, pode ser necessário que a equipe trabalhe sobre pressão, o que prejudica a qualidade do *software*.

Para Fowler (2003), os métodos ágeis são considerados como um meio terno entre a ausência de processo e o processo exagerado. Como exemplo pode-se citar a geração de documentação. Esta continua fundamental, mas a preocupação deve ser em não

desperdiçar tempo com a criação de documentos que não serão uteis. O planejamento também deve ser utilizado, mas com a consciência de que os planos podem sofrer alterações.

Pressman (2010) e Summervile (2011) mencionam que existem os seguintes métodos ágeis: XP (*Extreme Programming*), DAS (Desenvolvimento Adaptativo de *Software*), DSDM (*Dynamic Software Development Method*), *Scrum, Crystal*, FDD (*Feature Driven Development*), Modelagem Ágil (AM) e Processo Unificado Ágil (AUP). De acordo com Ambler (2014) e VersionOne (2009), dentre os métodos ágeis existentes, a Programação Extrema e o *Scrum* são os mais conhecidos e adotados nas organizações. Mediante ao entendimento referente à métodos ágeis, esse trabalho irá explicar as funções dos seguintes métodos ágeis: *Scrum*, *Extreme Programming* e *Feature Driven Developent*. Pois após o estudo citado acima e a pesquisa bibliométrica realizada, percebeu-se que esses métodos são os mais importantes e completos da atualidade.

1.2 *Scrum*

O método ágil *Scrum* tem como objetivo definir um processo para projeto e desenvolvimento de *software* orientado a objeto, que seja focado nas pessoas e que seja criado para ambientes em que os requisitos surgem e mudam rapidamente. Este método é considerado específico para o gerenciamento do processo de desenvolvimento de *software* (SCHWABER; BEEDLE, 2002).

Esse método destaca-se por ser um processo de desenvolvimento de *software* incremental em ambientes complexos, onde os requisitos não são claros ou mudam com muita frequência (SILVA *et al.*, 2013). Tem como objetivo a entrega de um *software* de qualidade dentro de iterações, formadas por intervalos de tempo chamados *Sprints* que possui aproximadamente um mês de duração (BEEDLE *et al.*, 1998). Para atingir seus objetivos o *Scrum* emprega um estrutura iterativa e incremental da seguinte maneira: no início de cada iteração, a equipe analisa o que deve ser feito e então seleciona aquilo que acreditam poder se tornar um incremento de valor ao produto ao final da iteração. A equipe então faz o seu melhor para realizar o desenvolvimento daquela iteração e ao final apresenta o incremento de funcionalidade construído para que os *stakeholders* possam verificar e requisitar alterações no momento apropriado (LIBARD; BARBOSA, 2010).

De acordo Schwaber e Beeldle (2002) o *Scrum* baseia-se em princípios como: equipes de até sete pessoas, requisitos que são pouco estáveis ou desconhecidos e iterações curtas. O método divide o desenvolvimento em intervalos de tempos de no máximo 30 dias, também chamadas de *Sprints*. Este método não requer ou fornece qualquer forma específica para a

fase de desenvolvimento, somente estabelece conjuntos de regras e práticas que devem ser utilizadas para o sucesso de um projeto.

- *Product Backlog*: é o início do *Scrum*, sendo considerada a prática responsável pela coleta dos requisitos. Nesta ação, através de reuniões com todos *stakeholders* do projeto, são revelados os itens com todas as necessidades do negócio e os requisitos técnicos a serem desenvolvidos, ou seja, o *Product Backlog* é uma lista (por ordem de prioridade) de atividades que provavelmente serão desenvolvidas durante o projeto (SCHWABER; BEEDLE, 2002).

- *Sprint*: é considerada a principal prática do método, é o nome como é chamado às interações que ocorrem no *Scrum*, ou seja, o período de trabalho para cada fase. É onde são executados os itens definidos no *Product Backlog* onde cada *Sprint* tem duração em média de 30 dias e tem seu objetivo claro e definido, conhecido por toda a equipe. Dento de cada *Sprint* acontecem reuniões diárias com duração média de 15 minutos chamadas de *Daily Scrum*, onde proporciona ao *Scrum* Master a atualização do status do projeto e auxilia na tomada de decisões do mesmo (LIBARDI e BARBOSA, 2010; SILVA *et al.*, 2013; LINDA e NORMAN, 2000).

Se o projeto tiver mais de um *Sprint,* cada prática deve conter uma nova implementação no produto, cabendo ao proprietário do projeto ao final de cada *Sprint* a decisão de implantar o produto que já esta desenvolvido ou tomar esta decisão mais tarde em outro final de *Sprint*. A cada final de iteração o produto que foi definido para ser desenvolvido deve estar pronto, codificado e testado (SILVA *et al.*, 2013; ABRAHAMSSON E SALO, 2002).

- Reunião de Planejamento da *Sprint*: cada iteração inicia com essa reunião, que tem por objetivo analisar os itens do *Product Backlog* a fim de priorizá-los para o desenvolvimento e, assim, definir o *Sprint Backlog* (SCHWABER; BEEDLE, 2002).

- *Sprint Backlog*: consiste em um conjunto de funções selecionadas do *Product Backlog* para serem executadas durante a Sprint. Quando todos os itens do *Sprint Backlog* estiverem prontos, uma nova parte do sistema é entregue ao Cliente (SCHWABER; BEEDLE, 2002).

- Revisão da *Sprint*: no ultimo dia de cada iteração, o *Scrum Team* e o *Scrum Master* apresentam o incremento para o cliente, o gerente e o Dono do Produto em uma reunião. Os membros avaliam a parte do sistema e decidem sobre as atividades seguintes. Na reunião poderão ser adicionados novos itens ao *Product Backlog* (SCHWABER; BEEDLE, 2002).

- *Burn Down Chart*: o monitoramento do projeto é realizado através de dois gráficos principais: *Product Burndown Chart* e *Sprint Burndown Chart*. Nestes gráficos é apresentada a quantidade de trabalho que ainda resta ser feito, sendo um excelente mecanismo para

visualizar a correlação entre a quantidade de trabalho que falta ser feita (em qualquer ponto) e o progresso do time do projeto em reduzir este trabalho (MARÇAL *et al.*, 2011). Os papéis no SCRUM que são quatro (Scrum Master, dono do produto, Scrum Team e cliente) podem ser visualizados no artigo de Schawaber e Beedle (2002).

1.3 *Extreme Programming* (XP)

A *Extreme Programming* (Programação Extrema) é ideal para ser utilizada em projetos em que os *stakeholders* não sabem exatamente o que querem e com isso, podem mudar de opinião durante o desenvolvimento do projeto. Com *feedback* constante, é possível adaptar rapidamente eventuais mudanças nos requisitos. Estas alterações nos requisitos são muitas vezes críticas nas metodologias tradicionais, que não apresentam meios de se adaptar rapidamente às mudanças (SOARES, 2015).

O XP é um método eficiente, flexível e de baixo risco para equipes pequenas e médias que desenvolvem *software* com requisitos dinâmicos ou em constante mudança (BECK, 2000).

As práticas sugeridas pelo XP causa polêmica à primeira vista e muitas não fazem sentido se aplicadas isoladamente. Segundo Beck (2000), nenhuma prática consegue se manter sozinha. É a união de várias práticas que sustenta o processo de desenvolvimento do XP.

As práticas referente à Programação Extrema de acordo com Beck (2000), Astels *et al.* (2002), Fowler (2003) e Jeffries (2001), são:

- Jogo do Planejamento: os participantes são o cliente e os programadores. O cliente decide sobre: escopo, prioridade, composição e datas das entregas. As responsabilidades dos programadores incluem: estimativas de tempo, desenvolvimento para cada função, avaliação dos riscos e decisão sobre o processo de trabalho. Essa prática divide-se em duas etapas chaves: Planejamento da Entrega e Planejamento da Iteração.

- Metáfora: o projeto de *software* é comandado por uma representação simplificada. Esta metáfora ajudar a conservar o grupo unido e em sintonia com o projeto.

- Projeto Simples: se o projeto for claro ele permanece ágil e flexível. Para manter um projeto descomplicado, é necessária uma revisão diária. A equipe XP não produz um enorme esboço inicial, mas projeta continuamente durante todo o tempo.

-Testes: os desenvolvedores escrevem Testes de Unidade e os clientes Testes de Aceitação frequentemente com o objetivo de tornar o sistema mais confiável. Os Testes de Unidade são escritos anteriormente ao processo de codificação e só após são executados.

- *Refactoring*: essa metodologia utiliza um processo de melhoria contínuo. *Refactoring* é a técnica usada na reestruturação do código, onde o objetivo é fazer com que

o código fique reutilizável e de fácil entendimento, sem que haja mudança no seu comportamento.

- Programação em Pares: todo sistema que é produzido pelo método XP é constituído por dois programadores que trabalham no mesmo computador. Desse modo essa prática assegura que todo código produzido é revisado por, pelo menos, outro desenvolvedor.

- Propriedade Coletiva: qualquer membro da equipe do XP tem a oportunidade de agregar valor a alguma parte do código, pois todos são responsáveis pelo sistema. Vale salientar que não é porque todos podem dar suas opiniões que os mesmos devam conhecer o código igualmente bem, e sim que cada pessoa deve saber o mínimo de cada parte do projeto. Um sistema de controle de versões é importante para garantir a aplicação da Propriedade Coletiva.

- Integração Contínua: o código de programação é integrado e testado depois de algumas horas ou no máximo depois de um dia que o mesmo foi desenvolvimento. A Integração Continua evita ou descobre cedo possíveis problemas de compatibilidade.

- Semana de Quarenta Horas: para os métodos ágeis, não é aconselhável fazer horas extras por períodos maiores que uma semana e também não é produtivo passar a noite acordado e trabalhar no dia seguinte. Um desenvolvedor cansado possui o raciocínio lento e se distrai facilmente, ou seja, podem surgir erros de código que poderão ser mais difíceis de serem corrigidos, levando a equipe à perda de tempo.

- Cliente Presente: o cliente que utilizará o programa deve estar disponível para auxiliar e fazer parte da equipe do XP. Pois com isso o usuário poderá fornecer detalhes do sistema quando surgirem dúvidas, o que pode acontecer em qualquer momento. Se o cliente não puder estar disponível é necessário um meio de comunicação constante, por exemplo, a internet.

- Padrões de Codificação: as equipes acompanham um modelo de codificação, ou seja, todo código pode ser visto como se fosse escrito por um programador. Para o XP, esse padrão pode ser composto da forma que a equipe achar melhor, o importante é que todo o código deve parecer familiar e simples para todos os membros.

Durante todo o processo de desenvolvimento são realizadas reuniões diárias onde as pessoas fiquem em pé, garantindo mais agilidade (BECK, 2000). A metodologia *Extreme Programming* enfatiza o desenvolvimento rápido do projeto e visa garantir a satisfação do cliente, além de favorecer o cumprimento das estimativas. As regras, práticas e valores da XP proporcionam um agradável ambiente para os seus usuários, que são conduzidos por quatro valores: comunicação, simplicidade, *feedback* e coragem (LIBARDI; BARBOSA, 2010). Na XP não existe a preocupação formal em fazer a análise e o planejamento de

riscos. Como riscos acontecem normalmente em projetos, este é um fator negativo dessa metodologia (SOARES, 2015). Os papéis do XP podem ser acompanhados no trabalho de Beck (2000).

1.4 *Feature Driven Developent* (FDD)

O método *Feature Driven Developent* (Desenvolvimento Dirigido a Características), tem como objetivo iniciar pequenas iterações que normalmente duram em torno de duas semanas, onde ao final acontece a entrega de uma parte do *software* funcionando (HIGHSMITH, 2002). Segundo Abrahamsson e Salo (2002), o FDD baseia-se em um conjunto de ações que devem ser utilizadas para assegurar o êxito do método em um projeto de *software*, são elas:

- Modelagem dos Objetos de Domínio: compreende a criação de diagramas de classes UML que descrevem os objetos interessantes do problema e também os relacionamentos entre eles. Para acrescentar, são desenvolvidos diagramas se sequência UML que devem descrever como os objetivos comunicam-se para desempenhar suas responsabilidades. Desenvolver um modelo dos objetos de domínio força a resolução de problemas logo no início, mantendo a integridade do sistema e reduzindo a quantidade de tempo de uma equipe gasta para fazer o *Refactoring* das suas classes, caso precise adicionar um novo item (PALMER, 2003).

- Desenvolvendo Através de Características: são constatadas as características do programa através de modelos de Caso de Uso, *User Stories* (que podem ser incorporadas do XP) ou qualquer outra ferramenta que sirva para descrever os itens do *software*. As características são divididas em grupos apresentados através de uma lista em ordem de importância. Depois de a lista ser definida o projeto inicia. Nesse método o objetivo é apresentar o aperfeiçoamento através da implementação dessas características. Os atributos selecionados para serem implementados devem ser executas dentro de duas semanas (limite máximo) (PALMER, 2003).

- Propriedade Individual de Classe: deve existir um responsável para cada classe ou conjunto de classes.

- Equipes de Características: a prática da Propriedade Individual da classe atribui classes a desenvolvedores específicos. Contudo, sabe-se que o desenvolvimento deve ser por características, com isso é necessário organizar equipes para as mesmas. São escolhidos desenvolvedores para serem líderes de equipes e é atribuído a eles um conjunto de características. Abaixo algumas considerações sobre as equipes de características (PALMER, 2003):

- Inspeções: esse método faz inspeções durante e ao final de cada iteração para assegurar a qualidade do projeto e do código. O objetivo das inspeções é a detecção de defeitos. É importante que a equipe veja as inspeções como uma ferramenta para eliminação de erros e como uma grande oportunidade de aprendizado (PALMER, 2003).

- Construções Regulares: possibilitam a detecção de erros e assegura que haja sempre um sistema atual e executável para ser apresentado ao cliente. Essa prática acontece durante as iterações, onde acontece o desenvolvimento de um conjunto de características (PALMER, 2003).

- Administração de Configuração: o FDD aconselha a utilização de um sistema de controle de versões para datar e manter um histórico das alterações feitas em cada classe. É sugerido solicitar uma autenticação dos clientes e gerentes do projeto para cada modificação, dando confiabilidade ao projeto (PALMER, 2003).

- Relatório dos Resultados: o método sugere que os resultados ocorridos durante o projeto sejam apresentados para toda a equipe e clientes. Podem ser disponibilizados através da internet e relatórios impressos, por exemplo (PALMER, 2003). O método FDD baseia-se em descrições referentes às fases do processo de desenvolvimento, pois essas etapas ajudam significativamente para o êxito do projeto. (COAD, 1999).

- Desenvolver um Modelo Global: primeira etapa do processo, onde são definidos o contexto e os requisitos do sistema. Os colaboradores envolvidos nesta etapa são o Especialista do Domínio e o Projetista. A tarefa requerida nesta fase é a modelagem do domínio da aplicação, onde são construídos o diagrama de classe UML, o(s) diagrama(s) de sequencia UML e uma lista de características (HIGHSMITH, 2002).

- Construir uma Lista de Características: o objetivo é construir uma lista completa de todas as características do programa. Nessa lista, a equipe de desenvolvimento apresenta cada função esperada pelo cliente baseada na lista de características gerada na etapa anterior, podendo ser necessária à inclusão de novas classes no modelo de domínio, que deverá ser refeito, apresentando agora os atributos e os métodos da nova classe. Cada característica da lista apresenta uma avaliação de tempo e uma prioridade que determina a ordem no desenvolvimento. Essa lista principal é dividida em listas menores que são agrupadas de acordo com as suas dependências. A execução de cada grupo de características não devera exceder duas semanas (HIGHSMITH, 2002).

- Planejar a Construção por Características: é construído um plano de execução dos grupos de características. As classes são distribuídas aos seus proprietários, a equipe de planejamento que é responsável pela elaboração do plano de quais características que

serão desenvolvidas é formada pelo Gerente de Projeto, Programador Chefe e Gerente de Desenvolvimento (HIGHSMITH, 2002).

Segundo o autor Highsmith (2002), planejar no FDD é também averiguar os fatores que poderão auxiliar ou prejudicar no desenvolvimento do sistema, por exemplo, determinação dos riscos, complexidade, balanceamento dos trabalhos, entre outros.

- Projetar Cada Característica: são executadas as seguintes atividades pela equipe e o Programador Chefe: Estudar a documentação existente; Desenvolver o diagrama de sequencia para o conjunto de características; Refinar o modelo do objeto; Reescrever as classes e os métodos e Inspecionar o projeto. Como resultado tem-se: o diagrama de sequência detalhado, o diagrama de classes atualizado, além das características com toda a documentação.

- Construir cada Característica: última fase de cada iteração do processo. As tarefas dessa etapa que são executadas pela equipe e o Programador Chefe: Implementação das classes e métodos; - Inspeção de código; Teste de unidade; Integração; Testes de integração e Entrega do incremento.

Todos os conjuntos de características devem passar pelas etapas de projeto e construção até que o sistema esteja finalizado (HIGHSMITH, 2002). Existem diversos papéis chaves, de suporte e adicionais no método FDD (PALMER E FELSING, 2002).

2 MATERIAL E MÉTODOS

Este trabalho se caracteriza por ser uma pesquisa com finalidade exploratória e descritiva, que busca aprimorar os conceitos e métodos já existentes sobre o tema abordado e descrever as características conhecidas em relação ao assunto, sendo este um dos objetivos da pesquisa. (BARROS; LEHFELD, 2000). Quanto aos procedimentos de coleta de dados serão caracterizados como: pesquisa bibliográfica sobre o assunto e comparativo que segundo Bulgakov (1998) é uma metodologia que facilitam a compreensão sobre assuntos através da investigação, ou seja, por se tratar de um estudo onde se selecionam as variáveis que seriam capazes de influenciar o trabalho, se define as formas de controle e de observação dos efeitos que a variável produz no objeto em questão, que neste caso seria a análise comparativa dos principais métodos ágeis (KAUARK; MANHÃES *et al.*, 2010). As variáveis metodológicas da pesquisa para análise serão qualitativas (SANTOS, 2000).

3 RESULTADOS E DISCUSSÃO

Nesta sessão será exposta a análise comparativa em relação aos métodos ágeis apresentados neste trabalho. Para melhor entendimento serão exibidos quadros resumindo as principais atividades propostas, práticas e papéis dos métodos ágeis escolhidos de acordo com a autora Fagundes, 2005:

Quadro 1 - Definição dos Requisitos

Metodologia	Atividades	Práticas	Papéis
XP	Clientes escrevem as *User Stories.*	- Jogo do Planejamento; - Metáfora; -Cliente Presente.	- Cliente; -Gerente; - Programadores.
Scrum	Definição do *Product Backlog.*	- *Product Backlog.*	- *Scrum Master;* - Dono do Produto; - Cliente.
FDD	Criação de artefatos para a documentação dos requisitos (Casos de Uso, *User Stories*, diagrama de Classe e de Sequência da UML).	- Desenvolvendo Através de Características.	- Gerente de Projeto; -Arquiteto Principal; -Especialista no Domínio; - Gerente de Domínio.

Conforme o Quadro 1, no método XP a definição dos requisitos acontece na etapa de Exploração. As práticas são: Jogo do Planejamento, Metáfora e Cliente Presente. Os papéis para essa atividade de levantamento dos requisitos são o Cliente e o Gerente. No *Scrum*, a definição dos requisitos acontece na subfase de Planejamento durante a etapa de *PreGame*, onde os requisitos conhecidos são listados originando o *Product Backlog*. Os papéis responsáveis por essa atividade são: o *Scrum Master*, o Dono do Produto e o Cliente. No FDD, os requisitos são listados, definidos e documentados através de Casos de Uso ou *User Stories* durante a fase Desenvolver um Modelo Global. A prática fica por conta do Desenvolvendo Através de Características. Os papeis escolhidos para a definição dos requisitos são: Gerente de Projeto, Arquiteto Principal, Especialista no Domínio e o Gerente de Domínio.

Quadro 2 - Atribuição dos Requisitos

Metodologia	Atividades	Práticas	Papéis
XP	Equipe técnica e clientes definem as *User Stories* que serão desenvolvidas na próxima iteração.	- Jogo do Planejamento; - Cliente Presente.	- Cliente; -Gerente.
Scrum	Definição do *Sprint Backlog*. As *Sprints* duram no máximo trinta dias.	- *Product Backlog*; - *Sprint Backlog*; - Reunião de Planejamento da *Sprint.*	- *Scrum Master*; - Dono do Produto; - *Scrum Team*.
FDD	As características são agrupadas, priorizadas e distribuídas aos responsáveis.	- Desenvolvendo Através de Características; - Propriedade Individual da Classe.	- Programador Chefe; -Especialista no Domínio; - Proprietário de Classe.

Conforme o Quadro 2, no XP, as atividades que acontecem na atribuição dos requisitos são: definição das *User Stories* que serão executadas durante cada iteração, essa operação é realizada nas práticas Jogo do Planejamento e Cliente Presente. Os papéis desta atividade são: Cliente e o Gerente.

No método *Scrum* os requisitos que serão aperfeiçoados em cada Sprint são definidos a partir da lista que existe no *Product Backlog*. As práticas relacionadas a esta atividade ação: *Product Backlog*, *Sprint Backlog* e Reunião de Planejamento da *Sprint*. Os papeis relacionados a esta atividade são: O *Scrum Master*, Dono do Produto e o *Scrum Team*. No FDD, é nesta atividade que as características são agrupadas, priorizadas e distribuídas aos responsáveis pelo seu desenvolvimento. As práticas relacionadas a esta atividade são: Desenvolvendo Através de Características e Propriedade Individual da Classe. Os papéis para que essa execução ocorra, são: Programador Chefe, Especialista no Domínio e o Proprietário de Classe.

Quadro 3 - Projeto da Arquitetura do Sistema

Metodologia	Atividades	Práticas	Papéis
XP	Atividade realizada ao mesmo tempo em que a escrita das *User Stories*.	- Projeto Simples.	- Consultor.
Scrum	Desenvolvimento de um projeto geral baseado nos itens atuais do *Product Backlog*.	-	- *Scrum Master*; - *Scrum Team*.
FDD	Construção de diagramas de Classe e diagrama de Sequência da UML.	- Modelagem dos Objetos de Domínio.	- Arquiteto Principal.

De acordo com o Quadro 3, o método XP indica que em conjunto com à escrita das *User Stories*, ocorra o projeto da arquitetura do sistema. O método não sugere a maneira como o projeto deve ser feito, mas utiliza a prática Projeto Simples. O papel designado para esta atividade é a do Consultor.

No *Scrum* esta atividade pode ser realizada na subfase de Arquitetura no *PreGame*. É construído um projeto geral baseado nos itens atuais do *Product Backlog*. O *Scrum* aconselha que ocorra esse projeto, mas não sugere prática associada a esta atividade. Os papéis responsáveis por essa ação são: *Scrum Team* e *Scrum Master*. O método FDD sugere que esta atividade aconteça através da construção de diagramas de Classe e diagramas de Sequência da UML referente à prática Modelagem dos Objetos de Domínio. O membro da equipe responsável por efetuar essa atividade é o Arquiteto Principal.

Quadro 4 - Desenvolver Incremento do Sistema

Metodologia	Atividades	Práticas	Papéis
XP	Implementação das *User Stories* que fazem parte da iteração corrente.	- Padrões de Codificação; - Programação em Pares;	- Programador; - Cliente; - Rastreador; - Treinador;

		- Propriedade Coletiva; - *Refactoring*; - Cliente Presente; - Entregas Frequentes.	- Consultor.
Scrum	Implementação dos requisitos contemplados no *Sprint Backlog* para a *Sprint* corrente.	- *Sprint*; - Reuniões Diárias do *Scrum*.	- *Scrum Master*; - *Scrum Team*.
FDD	Análise da documentação, geração de diagrama de Sequência da UML, refinamento do modelo gerado nas atividades anteriores e implementação das características que serão desenvolvidas durante a próxima iteração.	- Equipes de Características; - Administração de Configuração; - Construções Regulares.	- Programador Chefe; - Proprietário de Classe; - Gerente de Versão; - Especialista na Linguagem; - *Toolsmith*; - Administrador de Sistema.

Conforme o Quadro 4, o método XP realiza a atividade de desenvolvimento dos requisitos durante a fase de Iterações para a Entrega. As práticas utilizadas são: Padrões de Codificação, Programação em Pares, Propriedade Coletiva, *Refactoring*, Cliente Presente e Entregas Frequentes. Os papeis para a realização desta atividade são: Programador, Cliente, Rastreador, Treinador e Consultor.

No *Scrum* esta atividade acontece na fase de Desenvolvimento no decorrer da Sprint. Schwaber (1995) afirma que o desenvolvimento dos requisitos não adota qualquer processo determinado, isto é essa atividade pode ocorrer conforme algum recurso que a organização pretenda aderir. Além dessa opção os gestores podem utilizar outros métodos nessa etapa que possuem atividades definidas nesta questão (ABRAHAMSSON; SALO, 2002). As práticas para essa atividade são: *Sprints* e Reuniões Diárias do *Scrum*. Mesmo não abrangendo esse processo o método aconselha que sejam os seguintes papeis os responsáveis pela implementação: *Scrum Master* e o *Scrum Team*.

No método FDD a atividade ocorre nas fases Projetar e Construir cada Característica onde são realizadas várias atividades como: análise da documentação, geração de diagramas, refinamento do modelo gerado nas atividades anteriores e implementação das características. As práticas utilizadas nessa atividade são: Equipes de Características, Administração de Configuração e Construções Regulares. Os papeis são: Programador Chefe, Proprietário de Classe, Gerente de Versão, Especialista na linguagem, *Toolsmith* e Administrador de Sistema.

Quadro 5 - Validar Incremento

Metodologia	Atividades	Práticas	Papéis
XP	Os programadores executam os Testes de Unidade e os clientes executam os Testes de Aceitação.	- Testes; - Cliente Presente.	- Programador; - Cliente; - Testador.
Scrum	As validações são realizadas ao final de cada *Sprint*.	- *Sprint*.	- *Scrum Team*.
FDD	Os testes e inspeções são executados pelos próprios programadores após a implementação.	- Inspeções.	- Testadores; - Proprietário de Classe.

De acordo com o Quadro 5, no XP a validação dos incrementos acontece na fase de Iterações para a Entrega, através da execução dos testes de aceitação e dos testes de unidade. As práticas adotadas são: Testes e Cliente Presente. Os papéis selecionados para a atividade são: Programador, Cliente e Testador. O método *Scrum* sugere que esta atividade seja executada no final de cada Sprint, durante a fase de Desenvolvimento. O *Scrum* não realiza nenhum processo para esta atividade, com isso a equipe pode escolher com realizar esse processo. O papel selecionado é o *Scrum Team*. No FDD os testes ocorrem na fase Construir cada Característica, onde os testes são feitos pelos programadores. A prática realizada nesta atividade é a de Inspeções. Os papéis são: Proprietários de Classe e os Testadores.

Quadro 6 - Integrar Incremento

Metodologia	Atividades	Práticas	Papéis
XP	A integração acontece paralelamente ao desenvolvimento das *User Stories*.	- Integração Contínua.	- Programador.
Scrum	Atividade realizada ao final de cada *Sprint*. Não é proposta nenhuma diretriz para a realização desta atividade.	-	-
FDD	Atividade realizada após os testes no incremento.	-	-

De acordo com o Quadro 6, no método XP, essa atividade acontece na fase de Iterações para a Entrega e a prática executada é Integração Continua. O papel correspondente é o Programador. No *Scrum* a atividade sucede-se através por meio das estórias na fase de Desenvolvimento. O método *Scrum* não fala sobre práticas nem papéis relacionados a esta questão. No FDD essa atividade ocorre ao final de cada iteração durante a fase Construir cada Característica. Esse método também não mostra práticas e papéis associados à atividade acima.

Quadro 7 - Validar Sistema

Metodologia	Atividades	Práticas	Papéis
XP	O sistema é disponibilizado ao cliente para que o mesmo realize validações.	-	-
Scrum	O cliente realiza a validação do sistema em uma reunião no último dia da *Sprint*.	- *Sprint*; - Revisão da *Sprint*.	- *Scrum Master;* - Cliente.

FDD	Esta atividade ocorre através das inspeções e dos testes de integração.	-	- Testadores; - Proprietário de Classe.

Conforme o Quadro 7, o XP sugere que a equipe libere o sistema para o cliente onde o mesmo irá aprovar ou não o *software* por completo, mas esse método não aconselha práticas nem papeis referentes a esta atividade. O *Scrum* realiza essa atividade ao final de cada Sprint durante a fase de Desenvolvimento. As práticas utilizadas são: Sprint e Revisão da Sprint. Os papeis que participam dessa operação são: Cliente e *Scrum Master*. No método FDD a validação dos itens acontece na fase Construir cada Característica. O FDD não sugere prática relativas a atividade mas os papeis sugeridos são: Testadores e Proprietários de Classes.

Quadro 8 - Entrega Final

Metodologia	Atividades	Práticas	Papéis
XP	O cliente deve estar satisfeito e não ter mais nada a acrescentar em relação as funcionalidades do sistema.	-	-
Scrum	Não devem existir mais itens no *Product Backlog* a serem desenvolvidos.	-	-
FDD	O sistema é entregue após todos os conjuntos de características passaram pelas etapas do projeto e construção.	-	-

De acordo com o Quadro 8, no XP esta atividade ocorre na fase de Fim do Projeto. Para que a entrega aconteça o cliente deve estar completamente satisfeito com o sistema, sem ter nada a acrescentar no mesmo. O método não fala sobre nenhuma prática ou papel para a finalização da entrega, mas aconselha que a equipe preta serviços de suporte após a entrega caso os usuários possuam alguma dúvida. O *Scrum*, utiliza a fase *PostGame* para a entrega do *software*. Para que isso ocorra não deve existir itens no *Produtc Backlog* a serem aperfeiçoados. Esse método também não sugere praticas ou papeis para essa atividade.

O FDD não contem alguma fase distinta para essa atividade e não apresenta práticas ou papeis relacionados. De uma forma geral o método aconselha que o sistema passe por todas as etapas do projeto para depois ser entregue ao cliente. Após essa análise pode-se verificar as práticas e os papeis utilizadas para cada atividade dos métodos ágeis selecionados, assim foi possível visualizar suas diferenças e semelhanças em cada ação exposta do desenvolvimento de software.

4 CONSIDERAÇÕES FINAIS

Os Métodos Ágeis foram criados para solucionar certos obstáculos do desenvolvimento de sistemas, os processos *Scrum*, XP e FDD foram selecionados para fazer parte deste trabalho porque através da pesquisa realizada, vários autores mencionaram estes como sendo os melhores da atualidade. O objetivo proposto do trabalho foi cumprido através de

uma análise bibliográfica ampla e atual, onde foi possível realizar a comparação dos métodos, apresentar suas diferenças e similaridades, lembrando que este trabalho não teve a intenção de dizer qual é o melhor método, entre os resultados apresentados destaca-se que todas as etapas do desenvolvimento de software estão contempladas nas metodologias estudadas, com atividades bastante semelhantes em todas elas.

Porém algumas particularidades podem ser apresentadas como as inspeções de código em FDD além de esta possuir um foco maior em documentação, a XP apresenta a escrita dos testes antes da implementação e a programação em pares demonstrando uma ênfase maior com a etapa de testes, no entanto não existe a preocupação formal em fazer a análise dos riscos e por fim a *Scrum* com suas reuniões diárias e revisões demonstram uma ênfase na agilidade.

Acredita-se que este estudo auxilie aos interessados em utilizar metodologias ágeis em seus projetos, permitindo que os mesmos tenham uma visão geral. Vale salientar que quando um projeto for elaborado os membros da equipe devem procurar utilizar o método ágil que melhor se adapte a realidade da proposta a ser trabalhada. Como trabalhos futuros pode-se verificar a possibilidade de utilizar os métodos ágeis juntamente com a gestão de processos para auxiliar na evolução das organizações.

REFERÊNCIAS

ABRAHAMSSON et al. **Agile software development methods. Review and analysis.** Number 478 In VTT Publications**.** Espoo: VTT Technical Reserch Centre of Finland, 2002.

ABRAHAMSSON, P.; SALO, O. **Agile software development methods – review and analysis**. Espoo: VTT Publications 478, 2002.

AMBLER. S. **Agile adoption rate survey**. Disponível em: <http://www.ambysoft.com/surveys/agileMarch2006.html> Acesso em: 20 nov. 2014.

AMBRÓSIO, CRISTINA WEBER; LEITE, MARIA SILENE ALEXANDRE. Contratação por Desempenho em Serviços de Manutenção: O Caso da CST Arcelor Brasil. **Revista Científica Eletrônica Produção Online**, Florianópolis, Vol. 8, Num. 3, 2008.

AUGUSTO. C. **Introdução a métodos ágeis**. Disponível em: <http://www.devmedia.com.br/introducao-a-metodos-ageis/24526#ixzz3JZdmFvpe> Acesso em: 20 nov. 2014.

ASTELS, D.; MILLER, G.; NOVAK, M. **Extreme programming explained: guia prático**. Rio de Janeiro: Campus, 2002.

BARROS, A.; LEHFELD, N. **Fundamentos de metodologia científica: um guia para a iniciação científica**. 2. ed. São Paulo: Pearson Education do Brasil, 2000.

BECK, K. **Extreme programming explained: exchange reading**. Massachusetts: Addison-Wesley, 2000.

BEEDLE et al. **SCRUM: An extension pattern language for hyperprodutive software development**. Pattern Languages of Program's98 Conference, 1998.

BERNARDO, P. C.; KON, F. A Importância dos testes automatizados - controle ágil, rápido e confiável de qualidade. **Engenharia de Software Magazine**, v.3, n.1, p.54-57, 2008.

BULGAKOV, SERGIO. Estudos comparativos e de caso de organizações e estratégias. **Scielo** O&S. V. 5, N. 11, Jan – abril, 1998.

COAD, P. **Java modeling in color with UML**. Prentice Hall, 1999.

FAGUNDES, P. B. **Framework para comparação e análise de métodos ágeis**. 2005. 134f. Dissertação (Mestrado em Ciência da Computação) - Universidade Federal de Santa Catarina, Florianópolis, 2005.

FOWLER, M. **The new methodology.** 2003. Disponível em: <http://www.martinfowler.com/articles/newMethodology.html> Acesso em: 5 fev. 2015.

HIGHSMITH, J. **Agile Software Development Ecosystems**. Addison Wesley, 2002.

JEFFRIES, R**. XP magazine contents: what is extreme programming**. 2001. Disponível em: <http://www.xprogramming.com/xpmag/index.htm> Acesso em: 21 fev. 2015

KAUARK, F.; MANHÃES, F. C.; MEDEIROS, C. H. **Metodologia da pesquisa: um guia prático**. Itabuna: Via Litterarum, 2010. 88p.

KNIBERG, H. **SCRUM e XP direto das trincheiras – como fazemos SCRUM**. São Paulo: InfoQ, 2007. Disponível em: < http://infoq.com/br/minibooks/scrum-xp-fromthe-trenches> Acesso em: 20 fev. 2015

KIOSKEA. **Métodos ágeis (RAD, XP)**. Disponível em: <http://pt.kioskea.net/contents/229-metodos-ageis-rad-xp> Acesso em: 20 nov. 2014

LIBARDI, P. L. O.; BARBOSA, V. **Métodos ágeis**. 2010. 35f. Monografia (Especialização em FT-027 Tópicos em Computação) - Universidade Estadual de Campinas – UNICAMP Faculdade de Tecnologia – FT, Limeira - SP, 2010.

LINDA, R.; NORMAN, J. **The Scrum software development process for small teams**. IEEE Software, 2000.

OLIVEIRA, S. B.; OLIVEIRA, A. S.; MOTTA, R. A. S. M. Gestão de processos e tecnologia da informação: em busca da agilidade em serviço. **Gestão.org**, v.10, n.1, p.172-194, 2012.

MARÇAL, A. S. C. et al. **Entendendo o SCRUM segundo as áreas de processo de gerenciamento de projetos do CMMI.** 2011. Disponível em: <http://repositorio.roca.utfpr.edu.br/jspui/bitstream/1/1120/3/MD_ENGESS_I_2012_15.pdf> Acesso em: 4 mar. 2015.

PALMER, S.; FELSING, J. **A practical guide to feature-driven development**. Prentice Hall, 2002.

PAZ, Fábio Josende Paz; KIPPER, Liane Mahlmann. Sustentabilidade nas organizações: Vantagens e desafios. **GEPROS. Gestão da Produção, Operações e Sistemas**, Ano 11, nº 2, p. 85-102, 2016.

PINHEIRO, A. J. F. **BP2IT dos processos de negócio às tecnologias de informação**. 132f. Dissertação (Mestrado em Engenharia Eletrotécnica e Computadores) - Universidade do Porto Faculdade de Engenharia, Porto, 2004.

PRESSMAN, R. S. **Engenharia de software**. 6. ed. Porto Alegre: Mc Graw Hil, 2010.

SANTOS, A. **Metodologia científica: A construção do conhecimento**. 3. ed. Rio de Janeiro: DP&A Editora, 2000.

SCHWABER, K.; BEEDLE, M. **Agile software development with SCRUM**. Prentice Hall, 2002.

SCHWABER, K.; SUTHERLAND, J. **SCRUM guide – o guia definitivo para o SCRUM – as regras do jogo**. Disponível em: <http://www.scrum.org/storage/Scrum %20Guide %202011%20-%20PTBR.pdf> Acesso em: 20 fev. 2015

SILVA, D. E. S.; SOUZA, I. T.; CAMARGO, T. Metodologias ágeis para o desenvolvimento de software: aplicação e uso da metodologia Scrum em contraste ao modelo tradicional de gerenciamento de projetos. **Revista Computação Aplicada**, v.2, n.1, p.30-46, 2013.

SOARES, M. S. **Comparação entre métodos ágeis e tradicionais para o desenvolvimento de software**. Disponível em: <http://www.dcc.ufla.br/infocomp/artigos/v3.2/art02.pdf> Acesso em: 18 fev. 2015

SOMMERVILLE, Engenharia de software. São Paulo: Addison-Wesley, 2011.

VALLE, R.; OLIVEIRA, S. B. **Análise e modelagem de processos de negócio: foco na notação BPMN (business process modeling and notation)**. São Paulo: Atlas, 2009.

VERSION ONE. **State of agile development - 4th annual survey.** Disponível em: <http://www.versionone.com/pdf/2009_State_of_Agile_Development_Survey_Results.pdf> Acesso em: 20 nov. 2014

MANISFESTO, A. 2001. Disponível em: <http://agilemanifesto.org> Acesso em: 18 fev. 2015