UNIVERSIDADE ESTADUAL DE GOIÁS

UNIDADE DE ITABERAÍ

SISTEMAS DE INFORMAÇÃO

Anamar Ferreira Arrais

Vanessa Apoliana Carvalho do Carmo

USO DE METODOLOGIAS ÁGEIS NO DESENVOLVIMENTO DE SOFTWARE

Itaberaí

2010

Anamar Ferreira Arrais

Vanessa Apoliana Carvalho do Carmo

Uso de Metodologias Ágeis no Desenvolvimento de Software

Trabalho de conclusão de curso apresentado à Universidade Estadual de Goiás – Unidade de Itaberaí, como requisito parcial à obtenção do título de bacharel em Sistemas de Informação, sob orientação da professora mestra, Luciana Nishi.

Itaberaí

2010

Anamar Ferreira Arrais

Vanessa Apoliana Carvalho do Carmo

Uso de Metodologias Ágeis no Desenvolvimento de Software

Este trabalho de conclusão de curso foi considerado adequado para obtenção dos créditos na disciplina de trabalho de conclusão de curso, obrigatória para obtenção do título de Bacharel em Sistemas de Informação.

_____

Prof. MSc. Luciana Nishi
Presidente da Banca

_____

Prof. Esp. Sidney Benedito da Silva

_____

Prof. Esp. Rogéria Luzia Wolpp

O único lugar aonde o sucesso vem antes do trabalho é no dicionário.

Albert Einstein

RESUMO

O processo de desenvolvimento de *software* é o resultado de um conjunto de atividades que envolvem identificação, planejamento e validação.

Mesmo com resultados que apresentam evolução nos vários processos envolvidos no desenvolvimento de *software*, ainda existem dificuldades a serem superadas.

Prova desse fato são as várias abordagens que dia após dia vêem sendo apresentadas como complemento ou alternativas na atividade de padronização e gerenciamento dos processos envolvidos. Todas elas têm como objetivo fornecer uma produção controlada, rápida e que gere produtos de *software* corretos.

Neste trabalho será apresentada uma abordagem relativamente recente, conhecida como Desenvolvimento Ágil de *Software* ou Metodologia Ágil. Além disso, pretende-se apresentar um panorama da utilização desta abordagem pela comunidade de desenvolvimento de *software*. O objetivo é trazer uma contribuição local, com o intuito de agregar conhecimento acadêmico abrindo um leque de possibilidades para futuros trabalhos de pesquisa com exploração em estudo de caso.

Palavras-chave: Desenvolvimento Ágil de *Software*, Metodologia Ágil, *Scrum*, FDD, *Extreme Programming*.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ES – Engenharia de *Software*

TI - Tecnologia de Informação

UML - Modelo de Linguagem Unificada

OO - Orientado a Objetos

XP - *Extreme Programming*

FDD - *Feature Driven Development*

PMI - *Project Management Institute*

Sumário

1 INTRODUÇÃO

Os analistas e engenheiros de *software* freqüentemente se deparam com problemas que tornam o processo de desenvolvimento de software uma atividade subjetiva e complexa, tais como:

- As estimativas de custo e prazo são, comumente, imprecisas;
- Poucos dados históricos para guiar estimativas;
- A comunicação entre analista e usuário é, freqüentemente, pobre;
- Poucas ou inexistentes estratégias de testes;
- Alto custo de manutenção, ocasionado principalmente pela análise incompleta.

Segundo Pressman para contornar estes e outros problemas que envolvem o processo de *software*, foi criada a disciplina Engenharia de *Software* (ES).

A ES é uma área que combina métodos abrangentes para todas as fases do desenvolvimento, ferramentas para automatizar estes métodos, técnicas para implementação mais consistente e robusta, técnicas para medição e garantia da qualidade, atividades para coordenação e controle administrativo do processo. [PETERS, 2001].

A ES visa aperfeiçoar o desempenho e garantir o desenvolvimento de software de alta qualidade, de forma prática, ordenada e que estejam dentro dos prazos e orçamento, de forma satisfatória.

“A Engenharia de *Software* é o conjunto total de atividades necessárias para transformar os requisitos de um usuário em *software*” [PETERS, 2001].

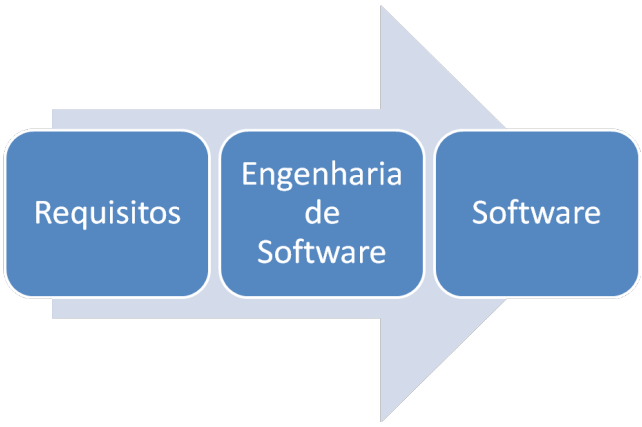


Figura 1 – Papel Engenharia de Software

Este trabalho aborda a Engenharia de *Software* focado: nas Metodologias Ágeis.

A parte inicial deste trabalho consiste em levantar dados como base para desenvolvermos o tema principal, posteriormente haverá abordagem sobre características e viabilidade de uso das metodologias estudadas e como desfecho, uma comparação teórica e em estudo de caso com cada metodologia ágil envolvida.

2 REFERENCIAL TEÓRICO

O capítulo que se segue trata-se de conceitos relacionados à área do desenvolvimento de *software* que objetiva sintetizar expressões relacionados ao mundo de TI.

2.1 Processos de *Software*

“O processo é um conjunto de atividades e resultados associados que produzem um produto de *software*". [Sommerville]

Segundo Humphrey (1990) um processo de *software* pode ser entendido como um conjunto de atividades exigidas para desenvolver um sistema. Como também auxiliar na redução dos problemas de treinamento, revisão e suporte para um projeto de *software*, bem como incorporar experiências aos processos que contribuem para melhorias no processo definido.

Um processo de *software* passa por três fases descritas por Schwartz:

1. Especificação de requisitos: descreve as necessidades ou requisitos do sistema para que seja descrita as funcionalidades desejadas para implementação;
2. Projeto de sistemas: traduz e descreve os requisitos de todos os componentes a serem codificados;
3. Programação: codificar ou produzir os códigos de controle do sistema;
4. Verificação e integração: verifica se o *software* desenvolvido alcançou sua meta e foi satisfatório ao cliente;

Caracteriza também a descrição do projeto de *software* e tradução dos requisitos. Representa inicialmente uma visão holística do *software* e refinamentos posteriores levam a uma representação bem próxima do código- fonte.

Segundo Pfleeger um processo envolve um conjunto de ferramentas e técnicas. Ele demonstra consistência e estrutura a um conjunto de atividades. É uma

junção de procedimentos organizados de modo que permita construir produtos que satisfaçam uma serie de objetivos e padrões. Sua estrutura orienta nossas ações permitindo examinar, entender, controlar e aprimorar as atividades que o compõem.

2.2 Metodologia Orientada a Objeto

O objetivo da metodologia OO (orientada a objetos) é definir todas as classes e relacionamentos relevantes para o domínio da aplicação pretendida para o desenvolvimento de *software*. [PRESSMAN]

2.3 Engenharia de Software

A definição criada por Bauer (2006) descreve um conjunto de boas práticas necessárias para a conclusão de um projeto com qualidade:

> "A engenharia de Software é a criação e utilização de sólidos princípios de engenharia a fim de obter softwares econômicos que sejam confiáveis e que trabalhem eficientemente em máquinas reais". (BAUER, 2006).

Como o período histórico da utilização e padronização do *software* é algo recente, para desenvolvê-lo foi necessário comparar a alguma produção existente. Logo a construção de software, se aproximou da Engenharia Civil.[TELES]

Por causa das etapas e seqüência aplicada na Engenharia civil, foi originado na ES o "modelo em cascata" – um dos mais tradicionais modelos de desenvolvimento de *software*.

2.3.1 Modelo Seqüencial ou Cascata

Este modelo é formado por uma estrutura linear e seqüencial. É também o modelo mais antigo e que mais foi utilizado na história do software sendo o modelo mais disseminado até a década de 90. Possui fases distintas de especificação, projeto e desenvolvimento. [PRESSMAN]

A estrutura desse modelo é um tanto quanto inflexível a respeito de mudanças nos requisitos e quando encaixam essas mudanças (o que é difícil de fazer nesse tipo de metodologia) também ocorrem grandes modificações no orçamento e prazo.

A figura a seguir descreve a maneira como flui o funcionamento deste.

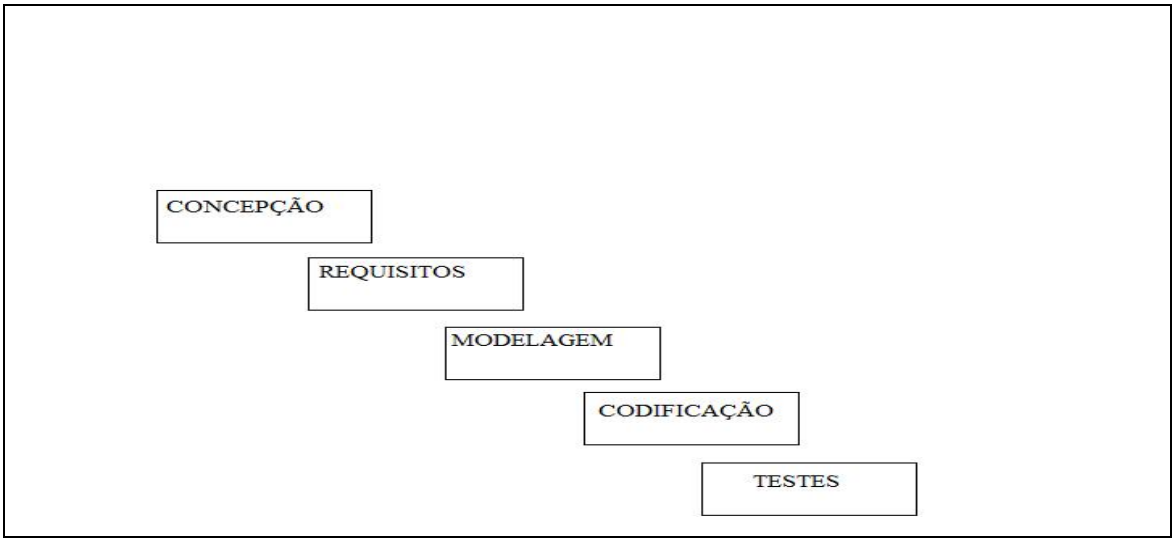


Figura 2 – Etapas do Modelo Sequencial ou Cascata

2.3.2 Modelo Evolucional ou Prototipação

Na prototipação, um escopo é delineado e no decorrer do projeto, requisitos são adicionados nas áreas que necessitam de complementar as definições. Os projetos são rápidos e visíveis ao usuário. Especificação, projeto e desenvolvimento de protótipos é o que ocorre neste modelo.

2.3.3 Modelo iterativo e incremental

Desenvolvimento é iniciado com um subconjunto simples de Requisitos de Software e interativamente alcança evoluções das versões até o sistema todo estar implementado. A preocupação com este tipo de modelo é adicionar uma nova funcionalidade a cada incremento.

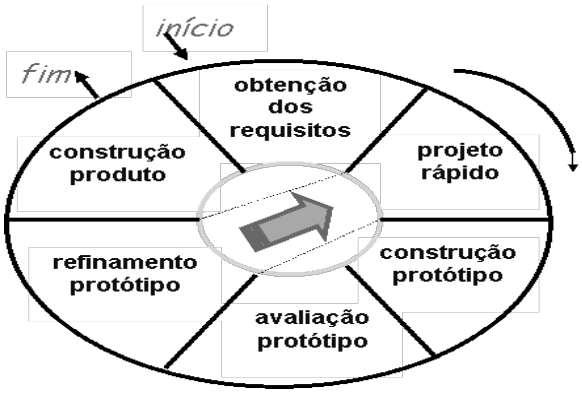


Figura 3- Etapas do Modelo Interativo e Incremental

3.3.4 Modelo Espiral

Neste modelo a evolução é proposta através de vários ciclos completos de especificação, projeto e desenvolvimento. É um junção do modelo Cascata e Prototipação.

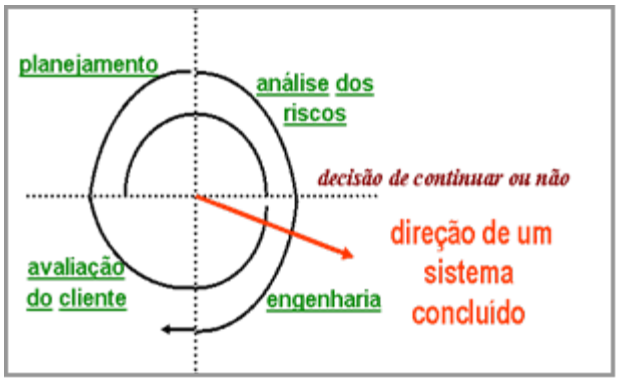


Figura 4 – Modelo Espiral

2.3.5 Modelo Rad

È um modelo baseado na construção usando componente, e objetiva um desenvolvimento rápido e dentro de um curto espaço de tempo. É a junção do modelo em cascata e modelo incremental.

2.3.6 Métodos Formais

Caracterizado por fazer uso de analise matemáticas que permite ao engenheiro fazer uso dessa técnica em todas as fases do projeto de *software*. Ou seja, desde especificações até verificação do *software*. Neste modelo há a promessa de que o *software* esteja livre de defeitos.

3 METODOLOGIAS ÁGEIS

A Engenharia de *software* surgiu em um momento em que a prática de desenvolver *software* estava sem padrões e modelo a seguir. Este período foi considerado o caos do desenvolvimento de *software*, onde características essenciais como tempo, confiabilidade, orçamento e funcionalidade deixavam muito a desejar no que existia até então.

Iniciava um novo marco para essa área. Mas ainda assim o desafio de mudar a imagem e a realidade de parte do desenvolvimento de *software* persistiu. A aplicação de metodologias que promovem o uso de documentação excessiva tornou realidade somente em organizações robustas. E para organizações em que a aplicabilidade destes conceitos não era uma tarefa viável, a alternativa que lhes restavam era o principio de desenvolvimento *ad-hoc*.[1]

As metodologias ágeis surgiram de uma reunião com os melhores desenvolvedores do mundo, onde levantaram os pontos em que os projetos por eles presenciados falhavam. Discutiram também os elevados orçamentos. Isso significa que o problema não foi resolvido como um todo no surgimento da Engenharia de *Software*.

A área de informática é um setor muito dinâmico que a cada dia exige mais de seus profissionais. O cliente tomando nota da evolução do setor não espera nada menos que um melhor desempenho. Essa atmosfera simplesmente sugere que tudo o que for possível para dinamizar esse trabalho seja feito. Como as metodologias são fundamentais e influenciam muito na qualidade do *software*, escolher qual utilizar é uma tarefa que exige um estudo aprofundado de seu impacto econômico e funcional.

Segundo Chaos (2009), estamos com os seguintes índices dentro do desenvolvimento de *software* como mostra a Figura 5:

[1] O termo *ad hoc* é utilizado para designar ciclos completos de construção de softwares que não foram devidamente projetados, para assegurar que este esteja de acordo com os padrões e especificações de prazo, qualidade e custo.

Figura 5 - Índice da Qualidade de Software

- **32% Sucesso** (no prazo, dentro do orçamento e com escopo completo)
- **44% Mudaram** (atrasaram, estourou o orçamento, e/ou reduziram escopo)
- **24% Falharam** (cancelados ou nunca usados)

Segundo essa pesquisa a maioria dos projetos analisados utilizavam de metodologias tradicionais. Mesmo porque ainda hoje existe um domínio de utilização dessas metodologias.

Dentro desse contexto, apresentamos as características de funcionamento das metodologias ágeis para que sejam uma alternativa dentro de cada realidade na hora de desenvolver.

As metodologias ágeis trabalham totalmente visando as variáveis de qualidade do desenvolvimento de *software*, que são:

- Tempo: Em cada uma das metodologias ágeis existe uma estimativa de tempo que são dispostas em ciclos pequenos, o que facilita atingir o objetivo final, que é entregar o projeto dentro do esperado.

- Custo: Por focar o que realmente vai compensar o cliente investir, e por implementar somente as reais necessidades, os custos com o uso das metodologias ágeis fazem com que o cliente esteja fazendo um investimento, que já começa logo a testar o seu retorno, evidenciando assim que acaba pagando por algo que lhe auxiliará a fazer a diferença em sua atividade.

- Qualidade: Segundo estudiosos, um *software* tem qualidade quando este funciona efetivamente, é confiável, é elaborado com um orçamento razoável e atende ao negócio do cliente. Essas são características em que os valores e práticas das metodologias ágeis norteiam alcançar.

- Escopo – O escopo é em grande parte dos *softwares* uma variável de difícil definição, mesmo porque na maioria das vezes os clientes dos mesmos, ainda não possuem uma idéia formada que execute na prática como idealizam inicialmente. Ou seja, os seus ideais computacionais se modificam, e na medida em que tomam nota de como realmente funcionam, a visão do cliente se volta para outra perspectiva.

Existem diversas metodologias de desenvolvimento ágeis de *software*. Nas próximas subseções tem-se três metodologias ágeis que serão abordados: XP (Extreme Programming), *Scrum* e FDD (*Feature Driven Development*).

Extreme Programming

Extreme Programming (XP) é uma metodologia de desenvolvimento de software que se originou na década de 90, mas que foi ganhando espaço no mercado a partir do Manifesto Ágil (2001).

Seus idealizadores foram Kent Beck e Ward Cunningham que conversaram sobre o histórico das atividades de ambos e enfatizaram práticas de suas experiências profissionais, focando em características como o uso de cartões simples, na hora de definir os requisitos e metáforas para trazer simplicidade, prática, e efetividade que resultou em *Extreme Programming*.

Embora aparente criar novos passos para a construção de *software*, a metodologia XP apenas apresenta outras prioridades e aplicabilidades ao desenvolver *software*, onde se continua usando práticas, regras e procedimentos já conhecidos como, por exemplo, o desenvolvimento iterativo. Ou seja, apenas uma interpretação que difere das metodologias tradicionais. Destaca-se em Programação Extrema a praticidade e simplicidade de suas regras e a priorização da satisfação do cliente.

Esta metodologia é indicada para qualquer tamanho de equipe, embora os melhores resultados se fossem obtidos com equipes menores .

Desenvolvimento XP

Ao iniciar o desenvolvimento de *software* usando XP, o cliente escreve de maneira coloquial o que deve ser feito para resolver o seu problema computacional. Essa descrição é chamada de *User Stories* ou *CrC Cards*. Depois de escrever as *User Stories* que representam os requisitos preliminares, a equipe juntamente com o cliente definirão a ordem de prioridades para as *User Stories*, ou seja quais serão implementadas primeiro.

O desenvolvimento é realizado em ciclos que duram cerca de 1 a 4 semanas, sempre sugerindo-se que o ciclo seja curto, e se a funcionalidade exigir

mais tempo que o recomendado, aconselha-se a dividir (quebrar) a tarefa para que seja implementada dentro desse período.

Práticas em *Extreme Programming*

Existem inúmeras práticas em XP, as que mais são utilizadas no desenvolvimento atual são as relacionadas a seguir.

Uma frisada prática é o uso de metáfora para definir o que o cliente quer. Pois a metáfora segundo os idealizadores de XP facilita a comunicação entre cliente e equipe; e entre desenvolvedores ajuda a definir a arquitetura do projeto.

Planejamento

Como o desenvolvimento em XP é iterativo, o planejamento é feito de forma mais flexível, de maneira a separar as funcionalidades (*User Stories*), que o cliente coloca como prioridade e a ordem em que deverão ser implementadas. Esse planejamento é feito com todos os desenvolvedores presentes e o cliente. As responsabilidades são dividas para cada integrante da equipe.

O cliente faz as decisões do negócio e os técnicos as decisões técnicas.

As decisões do negócio são: escopo, prioridade, composição dos *realeses* e data de entrega dos *realeses.*

As decisões técnicas são: estimativas, consequências, processos e cronograma detalhado.

Fases Pequenas

As iterações são feitas em pequenos prazos, cerca de uma ou duas semanas. Prazos maiores do que esses já começam a perder as características.

3.1.1.4 Design Simples

A ordem aqui é fazer com que apenas o que foi determinado para as iterações seja feito; seguindo o princípio de deixar mais simples possível, evitando duplicações de lógica e rodando todos os testes e sem acrescentar funcionalidades que poderão vir a ser utilizadas.

3.1.1.5 Testes

É a prática mais frisada em XP, e considerada uma das mais importantes. São executados testes de unidade e testes de funcionalidades.

Os testes de unidade são rodados diariamente e tem que funcionar 100%. Já os testes de funcionalidades são rodados de 0 a 100% e quando alcança o limite máximo o projeto terminou.

Antes de fundamentar *Extreme Programming* como conhecemos hoje, Kent Beck já havia desenvolvido um *framework* para testes unitários automatizados chamado de *SUnit* para a linguagem *SmallTalk*, e formado posteriormente um conjunto de ferramentas para outras linguagens que foi nomeada família *XUnit*. Essa ferramenta tem o papel de testar cada unidade de código, por exemplo, quando as classes são concluídas roda-se os testes unitários, e a ela é atribuído um valor, mediante isso é feita uma análise do retorno que a classe vai gerar se retornar o valor esperado tudo bem caso contrário o código é revisado. Existem outras características dessas ferramentas como gerar relatórios e fazer documentação por trecho de código, mas não convém citar todas as suas funcionalidades.

3.1.1.6 Refatoração

Significa melhorar o *software* sem modificar sua estrutura e código, apenas melhorar o que já está pronto, visando facilitar a manutenção. Por exemplo, encapsulamento e troca de nomes de variáveis.

3.1.1.7 Programação em par

Em XP a programação é feita em par. Enquanto um digita o outro observa. Isso minimiza muitos erros, pois a maioria deles podem ser detectados e corrigidos no ato. Existindo assim mais possibilidades de colocar o código o melhor possível no momento em que está sendo construído.

3.1.1.8 Propriedade Coletiva

Todos da equipe são responsáveis pelo desenvolvimento e o código fonte não pertence a um programador. Toda a equipe tem a liberdade de opinar no que for agregar valor ao projeto em qualquer momento.

3.1.1.9 Integração Contínua

Cada iteração deve ser integrada ao restante do código o quanto antes, para que possam ser feitos os testes e diminuir com isso as falhas. Essa integração deve ser feita em no máximo 24 horas.

3.1.1.10 Semana de 40 horas

A atividade de desenvolver *software* é extremamente desgastante e trabalhar além do limite não ajuda. Por isso esse é o limite de trabalho semanal para a equipe em XP, 40 horas.

3.1.1.11 Cliente Sempre Presente

O *software* é desenvolvido de acordo com as necessidades específicas de cada cliente, portanto ninguém melhor do que ele para esclarecer detalhes, para isso é necessária sua presença, na verdade o cliente é parte da equipe.

3.1.1.12 Padronizações

Para um melhor entendimento entre os integrantes da equipe e do próprio código é necessário que exista um padrão mantendo o entendimento do que foi e ainda será feito, isso reforça o valor da comunicação.

3.1.1.13 Reuniões em pé (*Stand up meeting*)

Essa é uma prática que visa um compartilhamento do andamento do desenvolvimento de maneira curta e direta. Durante a reunião o que está pronto e o que ainda vai ser feito é passado por toda a equipe, de modo a visualizar o andamento do projeto. Pelo fato de ser em pé evita distrações e mantém a equipe focada no que está sendo discutido. O principal objetivo dessa reunião é manter o valor da comunicação, visto que ela ocorre diariamente. Aqui também ocorre a escolha das *User Stories* que serão implementadas naquele dia, como também quem será responsável por cada uma.

Valores em *Extreme Programming*

Os valores em Extreme Programming são atitudes que norteiam o desenvolvimento como um todo. Essas atitudes são realizadas por toda a equipe desenvolvedora e estão descritos abaixo.

3.1.2.1 Comunicação

Desenvolvedores supõem que muitos projetos não saem como o esperado por causa da falha de comunicação, e/ou o longo caminho pelo qual ela pode passar, dificultando o entendimento da pessoa final que implementaria o projeto.

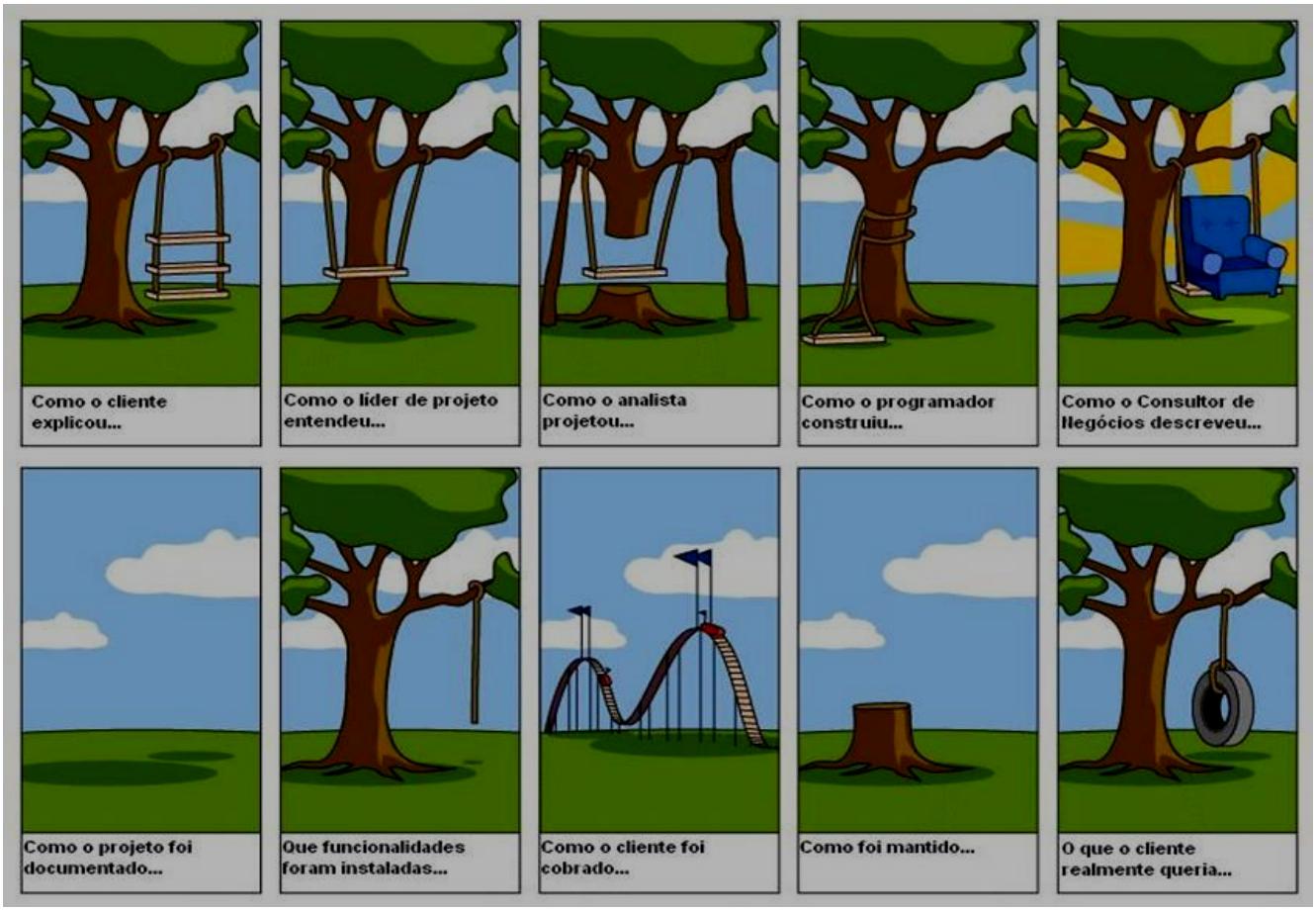


Figura 6- Extreme Programming

3.1.2.2 Simplicidade

No valor simplicidade, a ordem é não generalizar, não fazer previsões do que pode vir ser necessário, ou seja, projetar o que realmente é necessário, o que é indispensável, porque seguindo a esse quesito auxilia-se muito nas exigências de duas importantes das variáveis do desenvolvimento de *software*: custo e tempo. O que significa resolver o problema de hoje, hoje e o problema de amanhã, amanhã.

3.1.2.3 FeedBack

É essencial que no desenvolvimento com um ciclo curto, que as aprovações ou reprovações sejam feitas o quanto antes para que as funcionalidades em andamento sejam concluídas no prazo. É possível aplicar o *feedback* durante todo o tempo em XP, por isso é um valor que pode ser feito a todo momento.

3.1.2.4Coragem

Sabemos que o futuro não é previsível. Com base nessa premissa o desenvolvimento XP propõe aos seus utilizadores que além de estarem preparados para as mudanças que eventualmente ocorrem no decorrer do projeto, também estejam abertos ao fato de que terão que lidar com isso. O valor coragem quer deixar a equipe enfatizada no fato de que os esses riscos serão enfrentados. Estar preparado para um evento surpresa que provavelmente ocorrerá é melhor que não cogitar essa possibilidade. O mais interessante é que esses conceitos estão fortemente ligados as características do ser humano, o medo a incerteza, a frustração, a competitividade do mercado, e tudo isso afeta as pessoas o que elas estão fazendo, e como o estão fazendo. Trabalhar esse lado emocional da equipe só os prepara para estarem mais maduros na hora de lidar com isso.

Um fato interessante nessa metodologia, é que tudo indica que Kent Beck se inspirou fortemente na convivência com sua esposa que é psicóloga, trazendo para

o dia a dia dos desenvolvedores, práticas que melhoram a saúde interior e o bem estar.

3.1.2.5 Respeito

Em equipes XP não existem hierarquias. É fato que existem pessoas com as mais variadas personalidades, idéias e particularidades. Partindo desse contexto é essencial que os integrantes da equipe respeitem essas características que faz com que cada um seja único. E não olhe para si, mas para o que contribui com a equipe, que não é um, são todos.

SCRUM

Considerado por uns como metodologia, mas para outros como *framework* .

A origem do *Scrum* não foi à área de *software*, ele foi criado para o gerenciamento de uma empresa automobilística, e era utilizado como ferramenta para gerenciar outras empresas. Posteriormente por volta de 1993 o *Scrum* foi modificado por Jeff Sutherland da forma como o conhecemos hoje.

Assim como todas as outras metodologias ágeis, o foco de *Scrum* é adaptação, colaboração com o cliente e *software* funcional no menor espaço de tempo possível.

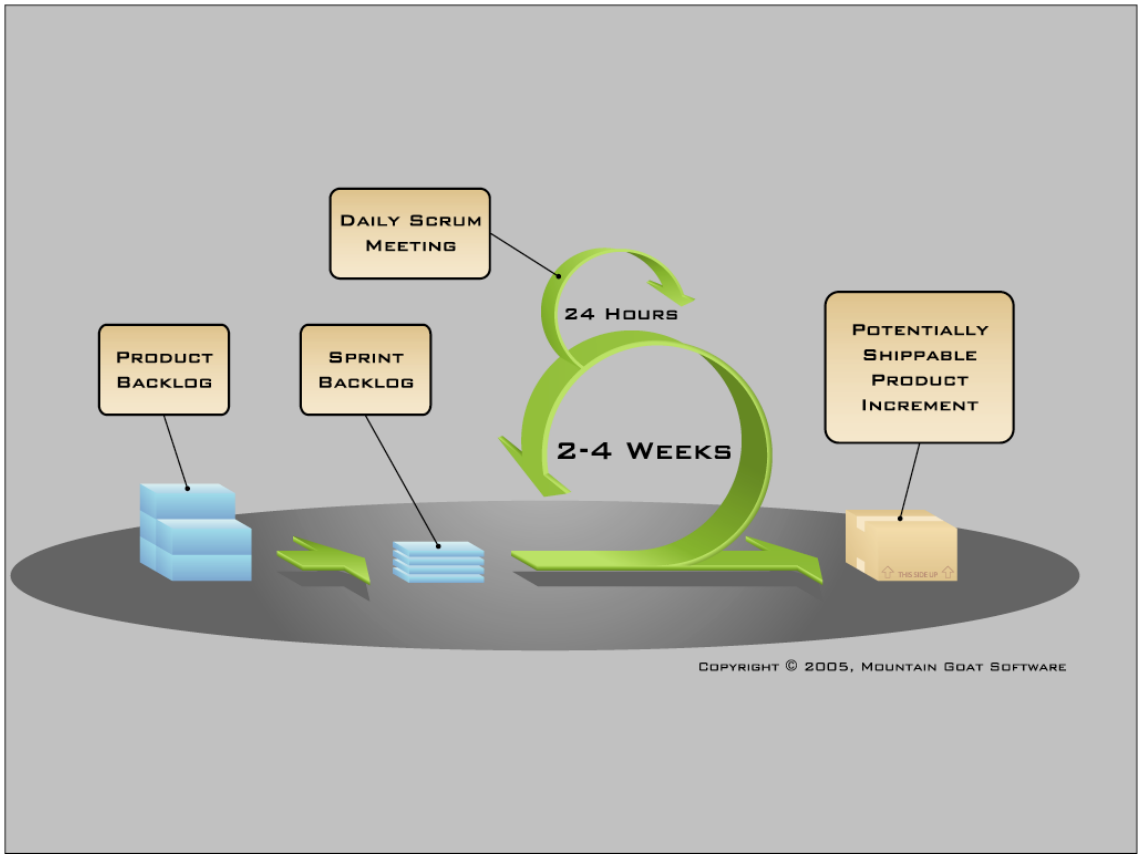


Figura 7 – Ciclo de Desenvolvimento Scrum

Nessa metodologia os períodos de desenvolvimento são de duas a quatro semanas.

Em Scrum o dono do *software* ou *Product Owner*, o *ScrumMaster* e a equipe são as pessoas envolvidas no desenvolvimento e representam um papel dentro dessa metodologia.

O dono do *software* é responsável por definir as prioridades, as datas de início e entrega das funcionalidades e realiza o *feedback*, aprovando ou rejeitando o que foi produzido. Ele também gerencia a parte financeira do investimento (ROI). As funcionalidades adicionadas ao *product backlog* pode ser feita por outras pessoas, mas somente o *Product Owner* é que pode definir prioridades.

O *Scrum Master* representa o gerente de projeto, mas não da maneira convencional. É responsável por fazer com que as fases do *Scrum* sejam desenvolvidas corretamente, garantindo produtividade, colaboração entre os papéis e toda a organização do projeto em si, removendo os impedimentos externos e facilitando as reuniões diárias.

A equipe no *Scrum* são os desenvolvedores como um todo, nesse sentido não tem nenhuma outra divisão de hierarquia a não ser o *Scrum Master*. Todos são responsáveis por alcançar os objetivos dos *Sprints*, colaborando com outros membros e participar das reuniões diárias. Na equipe *Scrum* as atividades não são atribuídas. Cada componente da equipe se responsabiliza por realizar as tarefas que são apresentadas no decorrer das iterações.

O dono do *software* decide as datas de lançamento dos conteúdos e todas as funcionalidades do produto, e também realiza o *feedback*.

3.2.1 Product Backlog

O *Product Backlog* representa nada mais nada menos que as funcionalidades definidas ou desejadas pelo cliente.

3.2.2 Sprint

O *Sprint* é uma parte do projeto que é desenvolvida num período de 30 dias, e essa parte é uma funcionalidade definida pelo cliente como prioridade.

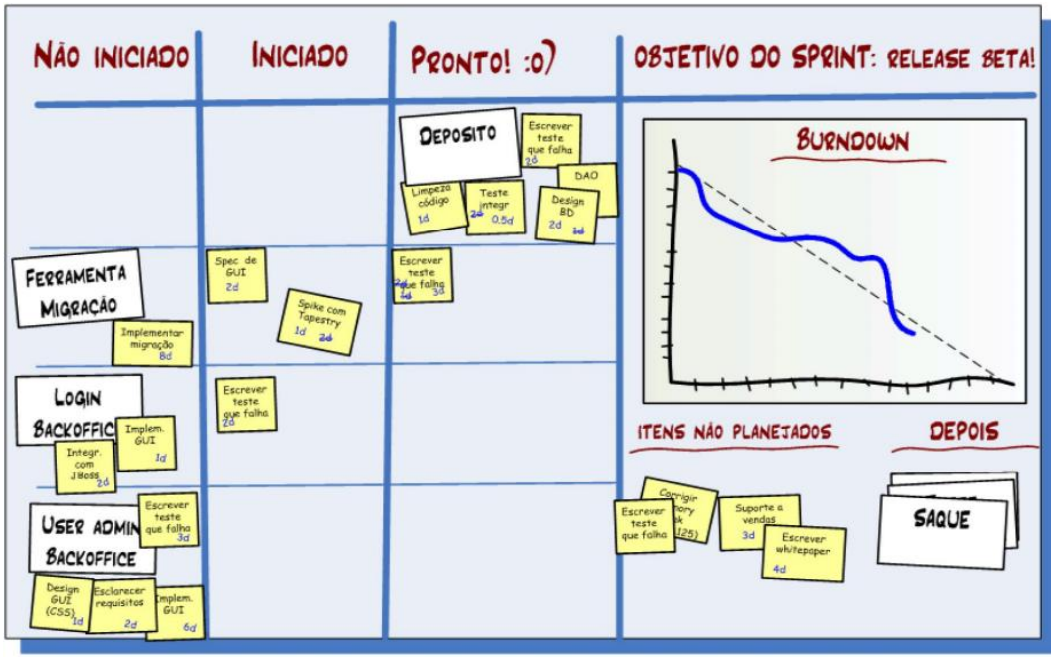


Figura 8 – Modelo de quadro de andamento Scrum

Os *Sprints* são compostos de quatro partes:

- ***Sprint Planning Meeting* (Planejamento)**

Antes de iniciar o *Sprint Planning Meeting* o *Product Owner* auxiliado pelo *Scrum Master* define o *Product Backlog* para dar início ao Planejamento.

No *Sprint Planning Meeting* o *Product Owner* expõe *Product Backlog* juntamente a toda equipe. E neste momento é que as funcionalidades mais urgentes é por ele apresentada e a equipe analisa em quanto tempo que tal funcionalidade pode ser entregue. Os detalhes do *Product Backlog* geram as tarefas que cada componente da equipe se prontifica a realizar uma. Essa tarefa tem uma estimativa de tempo em horas que é definida pela pessoa responsável por realizá-la Caso

aconteça da tarefa ter mais de 16 horas a mesma deve ser “quebrada” em mais horas.

Aqui é iniciado o momento de a equipe planejar com o quanto poderão se comprometer em relação às iterações, e também farão estimativas sobre a quantidade de funcionalidades que estarão entregando para o cliente nesse período.

The Sprint (Andamento)

Agora inicia algumas práticas que caracterizam o *Scrum* que são as reuniões diárias com duração estimada de 15 minutos, realizada preferencialmente no mesmo local e horário. Essa reunião é um momento da equipe comunicar entre si e prestar contas sobre a tarefa que cada um ficou responsável. O principal objetivo dessa reunião é mostrar o que foi feito no dia anterior, o que vai ser feito no dia atual e planejar o que será feito no dia seguinte. Nessa reunião é essencial a participação de todos os integrantes da equipe.

A equipe vai organizando o andamento do projeto pelo quadro de atividades, colocando as tarefas no seu respectivo *status*.

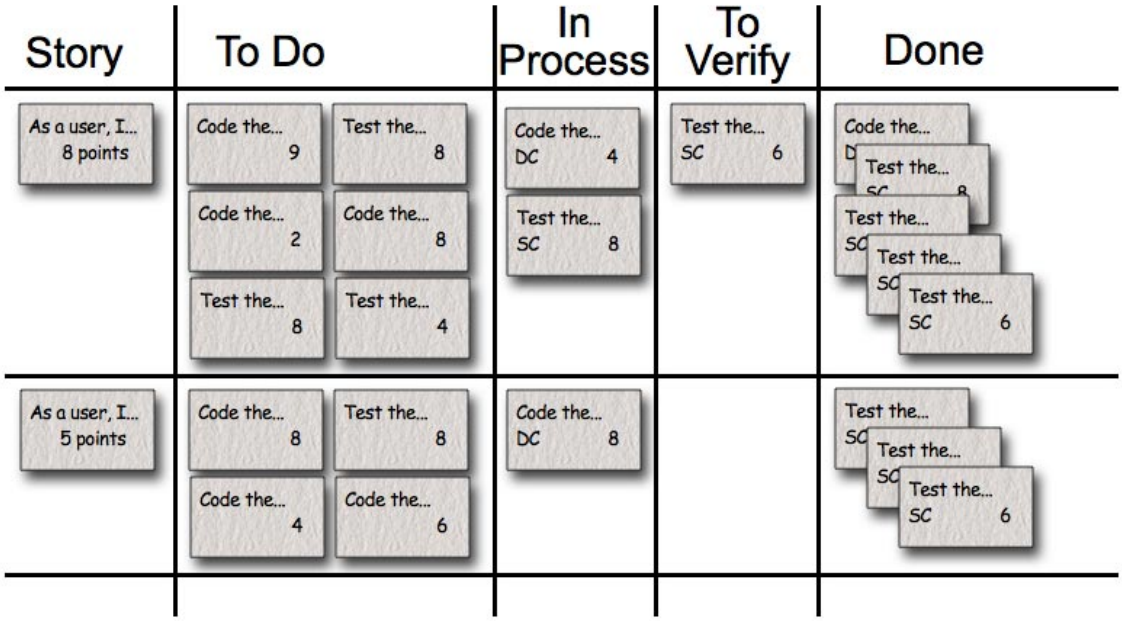


Figura 9 – Quadro de atividades

Jeff Sutherland afirma no prefácio de *Scrum* e *XP* Direto das Trincheiras[2] que é essencial que a equipe saiba sua velocidade de produção, visto que é de extrema importância saber disso para lidar com as estimativas de tempo das tarefas para que consequentemente seja feita uma boa estimativa de tempo de produção do *Product Backlog.*

Após a reunião diária o *Sprint Burndown* (representa o quadro de horas do projeto) é atualizado, informando o andamento do projeto nas métricas das horas, onde anteriormente foi passado a cada integrante da equipe uma quantidade de horas para implementação.

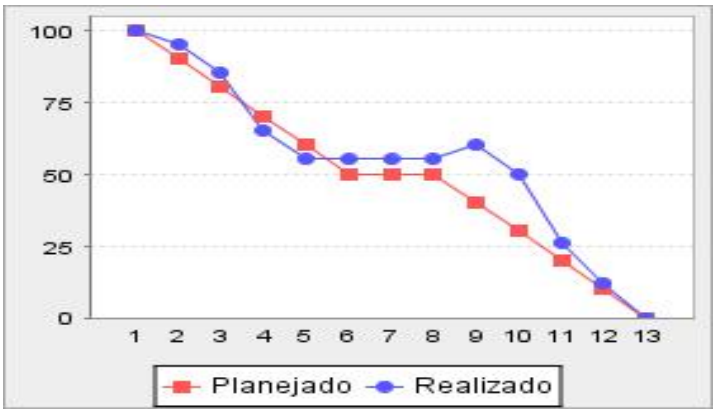


Figura 10 – Quadro de Horas

Sprint Review (Revisão)

Nessa etapa o que foi desenvolvido é apresentado em comparação com o que deve conter no produto final e os pontos positivos e negativos são expostos. Na verdade é uma etapa de avaliação, onde todos os componentes da equipe se reúnem (*Product Owner*, o *ScrumMaster* e *Scrum Team*) e verifica se o objetivo foi atingindo até então. Esse momento é essencial. É onde se recebe o *feedback* do cliente, e é aqui que o cliente vai deixar claro se o que está sendo produzido é o que ele esperava. Todos os artefatos produzidos devem ser apresentados, o *Sprint Burndown* por exemplo.

Já se define a data do próximo *Sprint Review* bem como os pontos a serem discutidos no mesmo.

[2] Livro sobre as metodologias ágeis Scrum e Xp (Extreme Programming) escrito por Henrik Kniberg.

Sprint Retrospective (Retrospectiva)

A retrospectiva é realizada para medir o que foi alcançado com qualidade suficiente e o que pode ser melhorado. Essa é uma apresentação feita em forma de demonstração do produto.

Feature Driven Development

Feature Driven Development (FDD) é uma metodologia ágil de desenvolvimento de *software* que une técnicas ágeis e tradicionais, e pelo fato das principais características das metodologias ágeis ressaltarem na mesma, ela se encaixa no quadro das ágeis.

Foi aplicada pela primeira vez em Singapura (1997-1998) em um projeto que estava praticamente falido. Os autores foram: Jeff De Luca e Peter Coad.

Fluxo do Desenvolvimento com FDD

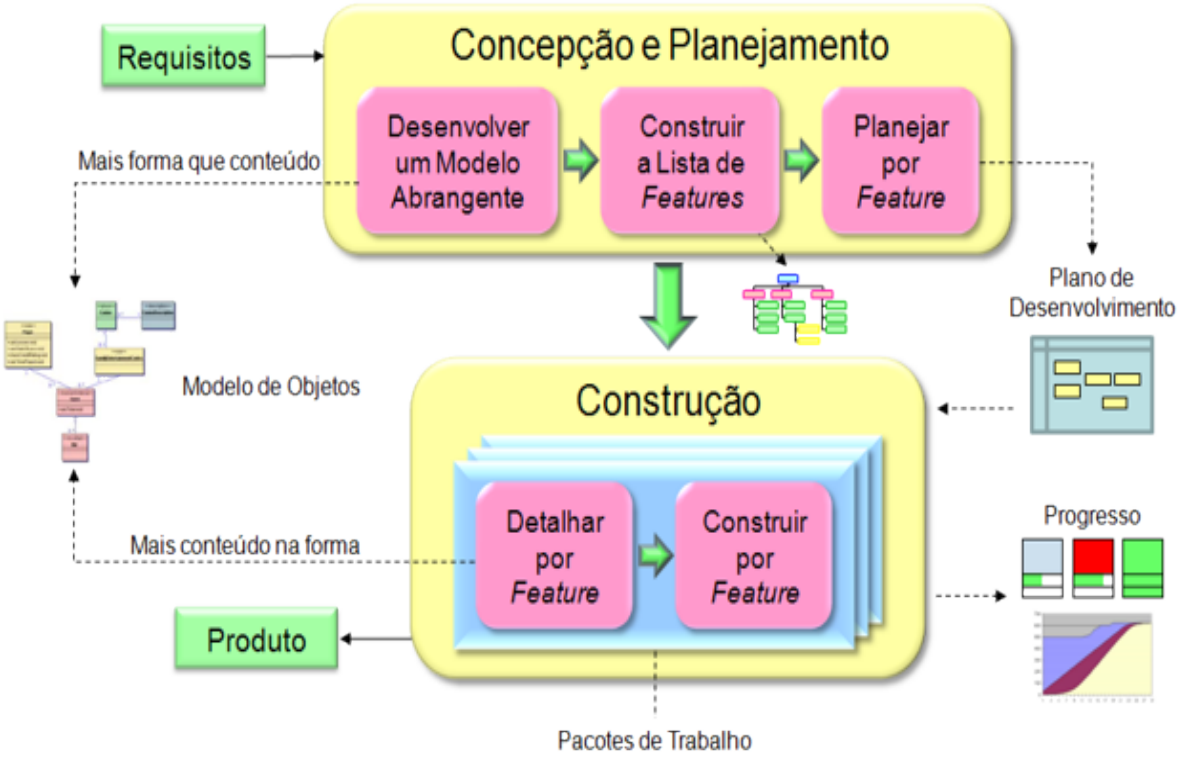


Figura 11- Concepção e Planejamento

Essa metodologia possui os seguintes princípios:

- Comunicação;
- Reduzir complexidade;
- Qualidade;

- Papéis.

O FDD é aplicado em cinco fases que se inicia com a definição de uma lista de requisitos nomeados *feature* que representam as funcionalidades, é selecionado através do que mais vai agregar valor ao negócio do cliente.

Essas *features* nada mais são que características desejáveis no *software* que seja pequena o suficiente para ser implementada em uma iteração. Então as *features*, serão utilizadas para planejar e gerenciar o projeto.

Os processos de *software* que já foi descrito anteriormente envolve cada pequena parte de um projeto desenvolvido em FDD. Segue abaixo como ocorre cada uma das partes envolvidas em um projeto FDD.

3.3.1 Desenvolver um modelo abrangente;

Ao iniciar o projeto, o modelo abrangente significa a definição dos objetivos. Isso implica em construir um modelo de dados que englobará os requisitos, utilizando de técnicas UML em cores que resultará em um modelo de alto nível que será a base para nortear o projeto. No estudo para formar o escopo ainda não é detalhado. As especificações são feitas por um modelador de objetos que posteriormente orienta os desenvolvedores a realizar a modelagem dos detalhes de cada área de domínio.

UML em cores é uma técnica que nasceu em 1997 pelos desenvolvedores Peter Coad, auxiliado por Eric Lefebvre e Jeff De Luca e publicada em 1999. A utilização da UML em cores facilita a visualização da modelagem, bem como a definição dos arquétipos, desde o seu surgimento ela vem sendo utilizada juntamente com a metodologia na qual originou.

3.3.2 Construir a lista de *feature*

Nessa etapa os detalhes de cada funcionalidade são desenvolvidos. O modelo abrangente é expandido, decompondo em hierarquia todo projeto, trazendo-o para um nível mais baixo do que o descrito no modelo abrangente.

3.3.3 Planejar por *feature*

Nesse momento a ordem de prioridade das *features* é definida, e o critério é o que mais agrega valor ao negócio do cliente. Também é levado em conta a ordem de codificação dos componentes, o nível de dependência entre eles.

3.3.4 Detalhar (projetar) por *feature*

Nessa etapa os requisitos são detalhados e o código esqueleto (classes) são construídos para que posteriormente sejam implementados.

3.3.5 Construir por *feature*

As classes são implementadas e testadas unitariamente e inspecionadas, e o que foi produzido até esse momento é entregue para o cliente usufruir.

3.3.6 Boas práticas da FDD

As principais práticas da FDD são as seguintes:

- Modelagem de objetos do domínio (negócio);
- Desenvolvimento por funcionalidade;
- Posse individual de classe (código);
- *Time* de funcionalidades;
- Inspeções de modelo e de código;

- *Builds* regulares;
- Gerenciamento de configuração;
- Relatório/visibilidade de resultados.

4 O USO DESSAS METODOLOGIAS

Em se tratando de metodologia cada um utiliza a que é mais adequada a sua realidade, e, portanto para cada situação a mesma metodologia que para um determinado grupo é a melhor coisa do mundo, para outros não funciona assim. E é tratando desse assunto que se inicia esse capítulo.

Mesmo sendo respaldado por pesquisas que características como custo, prazo e funcionalidade foram muito bem atendidas por metodologias ágeis, em alguns casos o uso destas foi cercado por inadaptabilidade.

Por exemplo, o uso de algumas práticas de XP mexeu muito com a cultura de algumas organizações, como o caso da programação em par. Codificar com alguém do lado é coisa absurda para muitos, o que faz com que as pessoas não aceitem o fato de mudar o modo como faziam o que fazem, tornando portanto a implantação da metodologia como um todo só pela metade. E mesmo sendo recomendado por Beck (um dos idealizadores da metodologia XP) que seja aderido uma prática por vez, a não intenção de aderir ao conjunto dissemina o uso de algumas práticas isoladas que sendo assim não dão suporte para ganhar com o conjunto da metodologia em si.

A mudança de cultura é algo muito difícil e essa é uma realidade em qualquer lugar. Equipes pequenas acabaram adotando “práticas isoladas” desenvolvendo requisitos de uma maneira muito informal, não realizando o desenvolvimento em equipe, porque a cultura da maioria das organizações até então é cada um trabalhando em seu canto, e essas ocorrências sinalizam características que agregam dados desvantajosos ao se pensar em aderir a XP, que foi a que teve um percentual mais alto de utilização dessa maneira.

Com esse critério as metodologias *Scrum* e FDD, tem um grau de aceitação maior que a XP, embora práticas das três já estejam em utilização isolada no mercado.

Segundo estudiosos, a falta de uma gerência de riscos, uma maior formalidade na definição dos requisitos, fazem com que tenham resistência em aderir a XP.

Outro fator é que mesmo quando todos estão dispostos a realizarem essa mudança de cultura, não é algo que ocorre do dia para noite, ainda mais se as práticas forem muito distantes da realidade local

Nisso o FDD leva vantagens sobre Scrum e XP, porque é uma metodologia mais próxima da realidade tradicional, os seus passos são mais semelhantes ao desenvolvimento comum.

Uma desvantagem que provavelmente é temporária são as poucas ou nenhuma ferramenta de apoio no desenvolvimento. A única metodologia que nesta pesquisa retornou que possui uma ferramenta é o XP, a família *XUnit* citada anteriormente e o *XPlaner*.

3.1 Economia

Na definição de Bauer sobre perfil ideal do desenvolvimento de *software* a economia vem como característica de qualidade no decorrer de todo o projeto. As pesquisas sobre desenvolvimento de *software* mostram que o escopo muda mais de 50% dentro dos projetos. Segundo a PMI (*Project Management Institute)* Brasil 2006 ocorrem cerca 69% de mudanças de escopo dos projetos, relacionando esse dado com o impacto econômico que ele exerce sobre o desenvolvimento confirmam e destacam as metodologias ágeis.

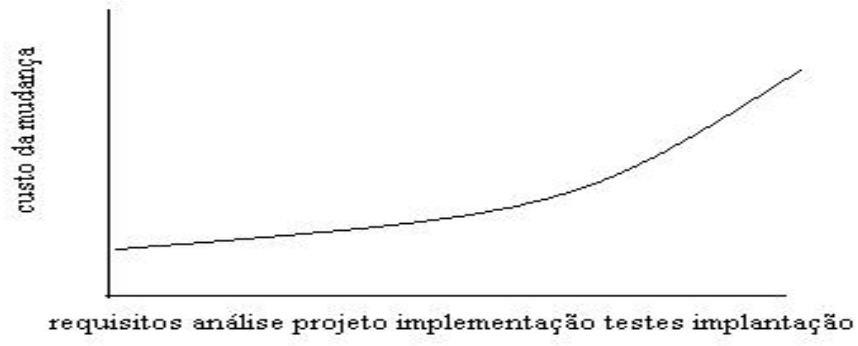


Figura 12 - Gráfico do custo com o modelo em cascata.

No momento em que as mudanças ocorrem no desenvolvimento tradicional é o momento em que os custos sobem. O mesmo não ocorre com as metodologias ágeis, o que enfatiza a eficácia destas no projeto.

Outra vantagem das metodologias ágeis é o fato de trabalharem com a questão das mudanças incentivando-as, de modo que assim fique sempre o mais próximo do que o cliente deseja.

Em relação à adoção as metodologias ágeis *Ambler* (2006) identifica e classifica as empresas a esquerda (ver figura 13) como inovadoras, como as que assumem o risco e optam por aderir ao novo, as que foram classificadas como primeira maioria e segunda maioria estão á espera por mais resultados, e as chamadas tardia necessitam de mais provas palpáveis de seu funcionamento.

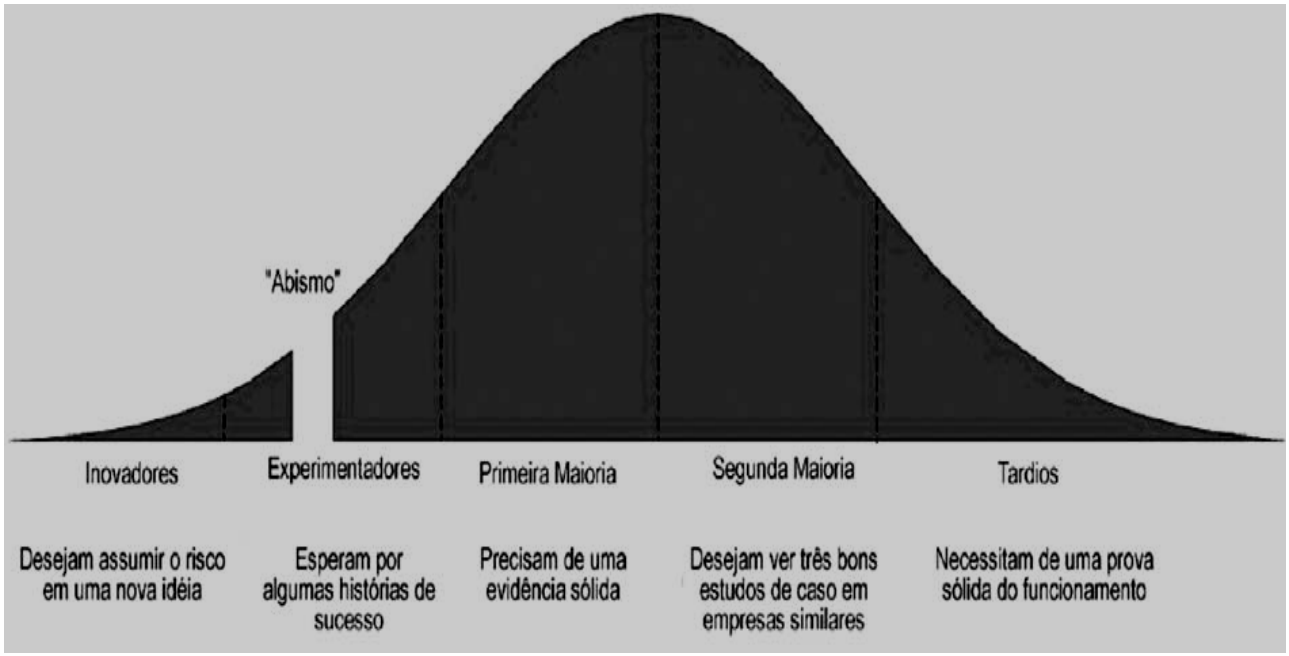


Figura 13 - Curva de adoção de tecnologia.

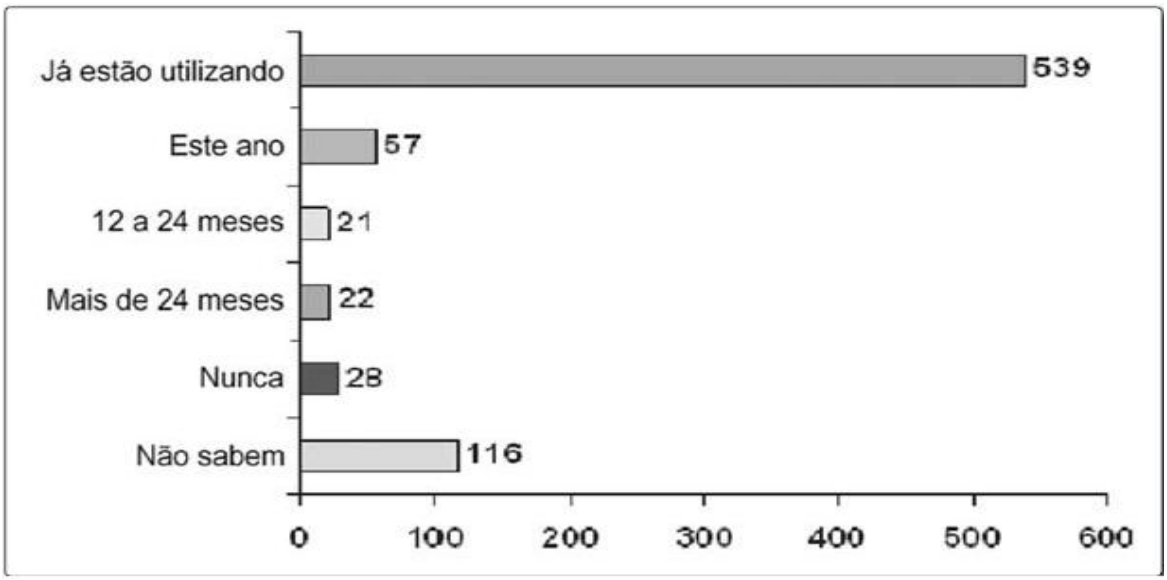
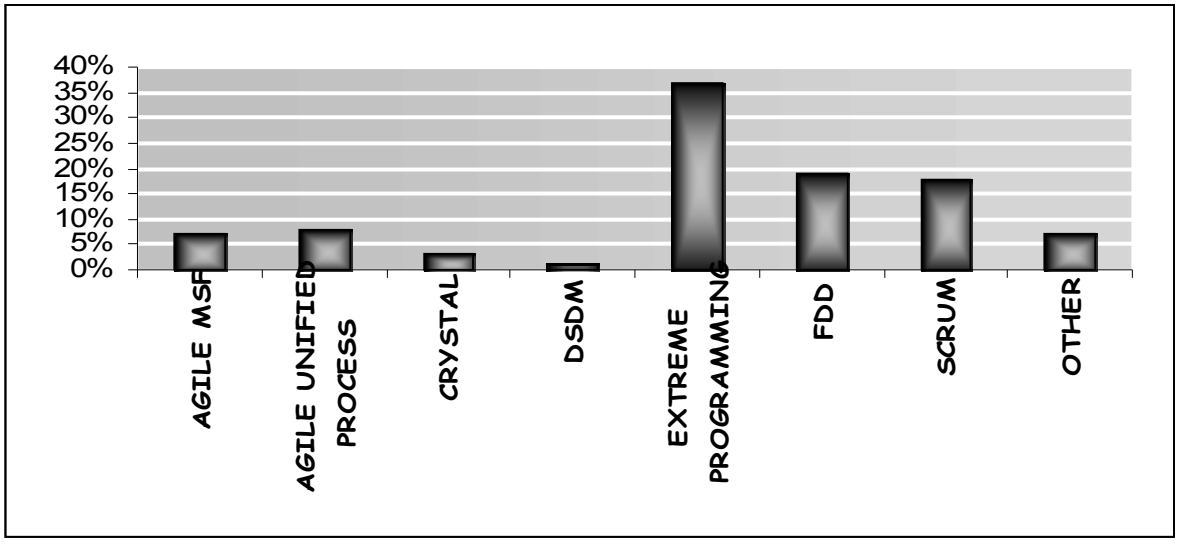


Figura 14- Adoção de método ágil.

Baseando ainda em pesquisas realizadas por Ambler (2006) esse é o percentual do uso atual do desenvolvimento de *software* utilizando metodologias ágeis, salientando que essa foi uma das maiores pesquisas realizadas sobre o uso dessas metodologias, e apontando claramente que *Scrum*, *XP* e FDD são as ágeis mais usadas. Na figura a abaixo segue o percentual.

Figura15 - Percentual do uso de metodologias ágeis



Seguindo a comparação de características segue as tasbelas que colocam em evidência os pontos destacados de cada metodologia.

DISTRIBUIÇÃO DE PAPÉIS NAS EQUIPES AGÉIS		
FDD	XP	SCRUM
Gerente de Projeto Arquiteto Chefe Especialistas de domínio Gerente de Desenvolvimento Programadores Chefe *Class Owners*	Programadores Arquitetos Analista de Testes Analista de Negocio Projetista de Iteração Gerentes de projeto Gerente de produto Executivos Redatores técnicos Usuários Recursos Humanos	*Product Owner* *ScrumMaster* Time

Tabela 1 - Tabela da composição da equipe nas metodologias ágeis

Em XP, qualquer integrante da equipe pode assumir em qualquer momento os papéis e exercer mais de um papel ao mesmo tempo. A seguir uma tabela com alguns nomes de grandes empresas que utilizam metodologias ágeis para desenvolver projetos.

EMPRESAS QUE UTILIZAM METODOLOGIAS ÁGEIS		
SCRUM	XP	FDD
Kwnotec H2J OneCast Technologies, InterBusiness Technologies Ci&T, Fundação Paulo Feitosa, FemaHosp PowerlogicS. A Living Consultoria Localweb	Improve it Objective Solutions. Localweb	Heptagon, Cognizant, Ivis, Nebulon, Gbst, ProcessExchange, Corbis TechExcel

Tabela 2 - Tabela de empresas que utilizam metodologias ágeis

Salientando que têm sido propostas por adeptos as respectivas tecnologias, o uso de práticas combinadas de duas metodologias, como tem sido comum o caso do uso de XP e *Scrum*. Abaixo, segue tabelas comparando aspectos das metodologias estudadas:

Especificação dos Requisitos	
Scrum	Definição de características que formam o *Product Backlog*
XP	Clientes escrevem as *User Stories*
FDD	Produção de artefatos para a lista de requisitos

Tabela 3 – Comparação 1

Implementação dos Requisitos	
Scrum	O que vai ser implementado é definido a cada *Sprint* pelo *Product Owne*r, e definido diariamente pelo time de acordo com a lista contida no *Product Backlog.* Implementações com duração máxima de 30 dias.
XP	Técnicos e cliente(s) tomam as deciões de prioridade de implentação - *User Stories* que serão implementadas. Ciclo de 1 a 4 semanas.
FDD	Requisitos são agrupados de acordo com o grau de dependência e priorizados de acordo com a importância. Implementações feitas em no máximo duas semandas.

Tabela 4 – Comparação 2

Incrementação do Projeto

Scrum	Definido a cada Sprint. Novos requisitos são levantados.
XP	Definido a cada ciclo. Novos requisitos são levantados.
FDD	Realização de refinamento dos requisitos, geração de diagramas (UML)Requisitos são levantados,

Tabela 5 – Comparação 3

Validação da iteração	
Scrum	Não exite método definido validação.
XP	Testes de unidade realizado pelos programadores e testes de aceitação realizados pelo(s) cliente(s)
FDD	Realização de inspeções e tetes desenvolvido pelos programadores.

Tabela 6 – Comparação 4

Integração	
Scrum	Integração ao final de cada Sprint.
XP	Em XP, é realizado integração contínua. Integrar sempre e o quanto antes. Realizado paralelamente.
FDD	Realizado ao final dos testes de inspeções.

Tabela 7 – Comparação 5

Validação do Sistema	
Scrum	Validações são feitas na Revisão dos *Sprints*. O cliente presente desempenha essa tarefa.
XP	Como é uma prática realizada ao final de cada iteração quando

	terminam as *User Stories* não há sugestão dessa atividade. Somente a validação final realizada pelo cliente.
FDD	Realizado ao final das iterações inspeções aos testes e integração.

Tabela 8 – Comparação 6

Finalização do sistema	
Scrum	O sistema é entregue quando não há mais itens no *Product Backlog.*
XP	Termina quando o cliente está satisfeito e não tem mais funcionalidades a acrescentar.
FDD	Concretizado quando o sistema já passou por todas as etapas de projeção e construção.

Tabela 9 – Comparação 7

4 ANALISE REGIONAL

Para o desenvolvimento desta analise foi necessária a busca de opiniões de profissionais que trabalham ou conhecem o desenvolvimento com metodologias ágeis e tradicionais. porem uma grande dificuldade foi encontrar voluntários para este desenvolvimento. Problema solucionado com muita conversa e explicações sobre os reais interesses desta pesquisa.

Somado aos atributos utilizados para concluir o desfecho deste trabalho segue abaixo um capítulo cujo o principal objetivo é analisar a opinião de desenvolvedores reais que opinaram através de um questionário sobre aspectos do desenvolvimento que foi tratado no decorrer deste instrumento de pesquisa. O domínio de pesquisa foi realizado com desenvolvedores locais da cidade de itaberaí, Cidade de Goiás e Goiânia, compondo um total de quinze participantes no período do mês de outubro/2010.

Para iniciar a pesquisa o ponto de partida foi o tempo de utilização do ramo de desenvolvimento de software que segue na figura abaixo:

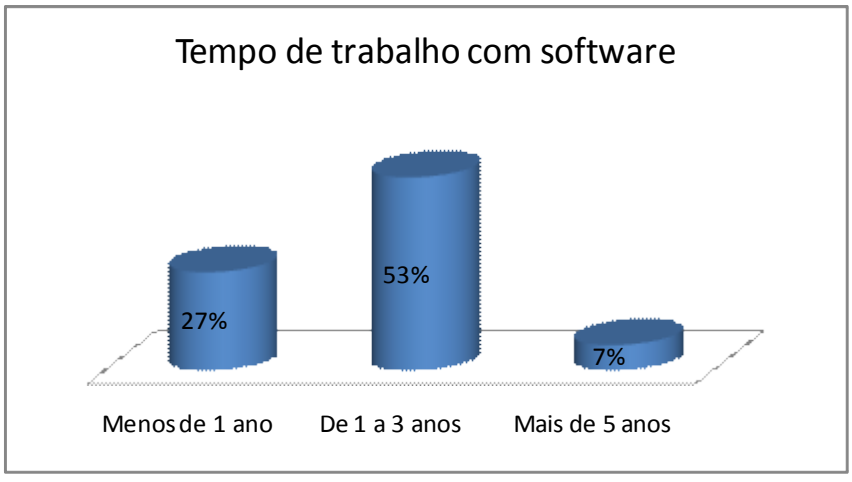


Figura 16 – Questão 1

Isso significa que a maioria dos participantes estão inseridos dentro de um contexto mais recente no que diz respeito a tempo de desenvolvimento.

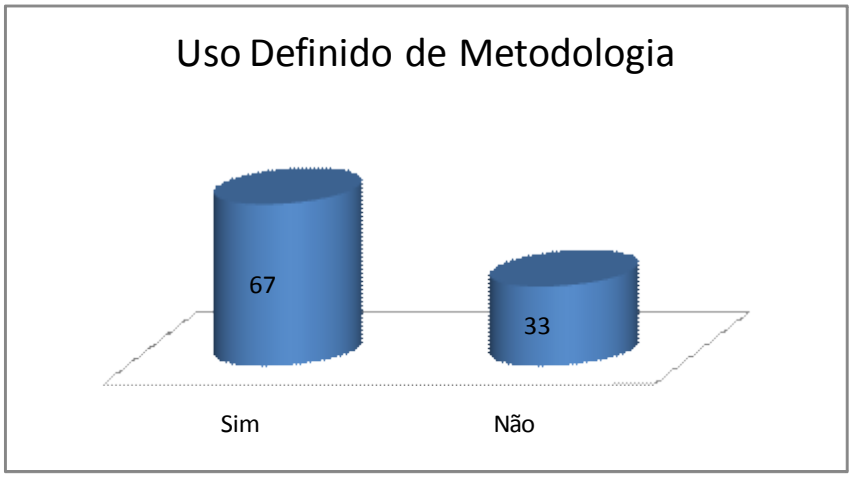


Figura 17 – Questão 2

No decorrer da história da informática, especificamente anterior ao surgimento da Engenharia de *Software*, as metodologias não eram consideradas relevantes na hora de desenvolver mesmo porque não havia ainda um padrão, hoje através do que se vê no mercado e no contexto dessa pesquisa nota-se que esse perfil teve grande mudança, e a opinião dos profissionais da área mudaram significativamente nesse ponto, o que se comprova através da figura acima que mostra mais de 50% dos participantes em seus respectivos domínios utilizam uma metodologia definida de desenvolvimento.

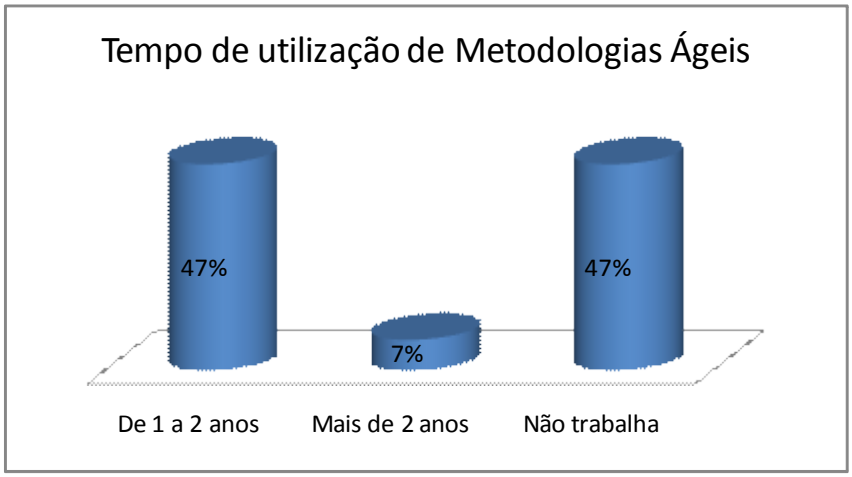


Figura 18 – Questão 3

Em se tratando especificamente da utilização de metodologias ágeis, nota-se que os desenvolvedores que estão inseridos no mercado recentemente focam

diretamente nas metodologias em estudo. Isso faz lembrar o fato tratado anteriormente de que a cultura organizacional interfere de maneira direta na utilização de novas abordagens. Reforçando também outras pesquisas, como a realizada pelo *Chaos Report*, também citada anteriormente neste documento que apresenta ainda um grande índice de insucessos no desenvolvimento devido a variáveis do desenvolvimento que por causa da cultura não são considerados. O que valida o trabalho realizado, no sentido de apresentar para a comunidade acadêmica como um todo a necessidade de considerar as opções na hora de desenvolver.

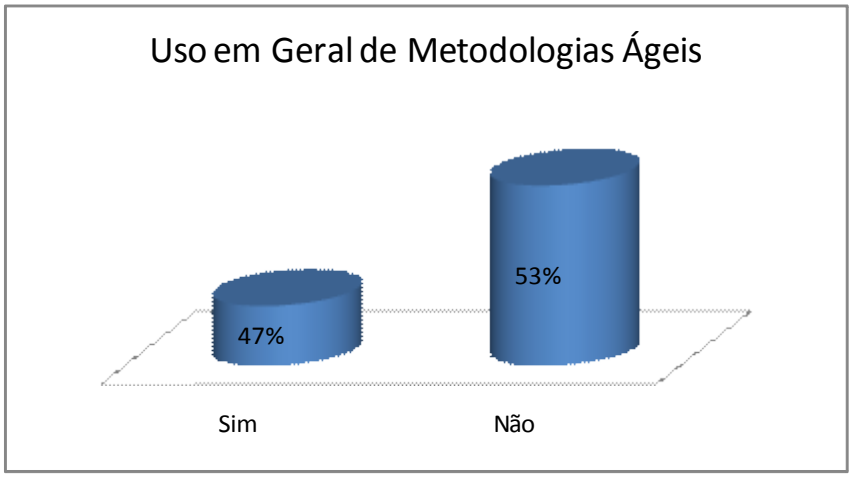


Figura 19 – Questão 4

A figura acima demonstra que ainda não há uma utilização em grande escala das metodologias ágeis.

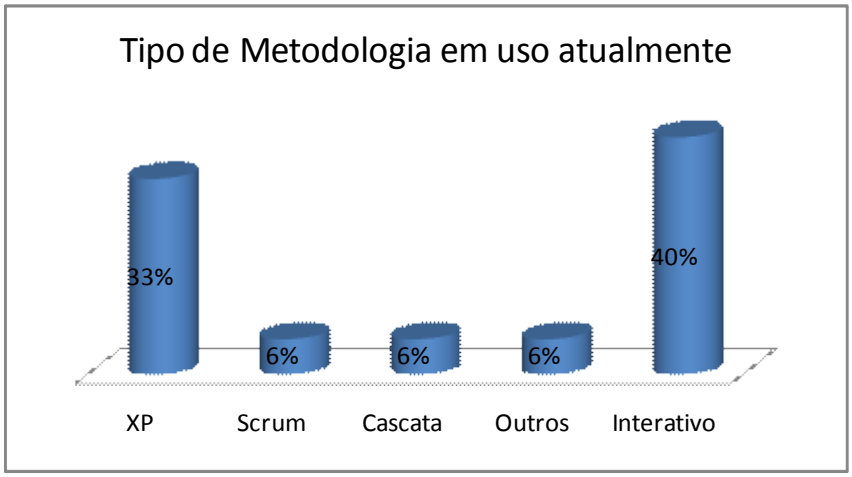


Figura 20 – Questão 5

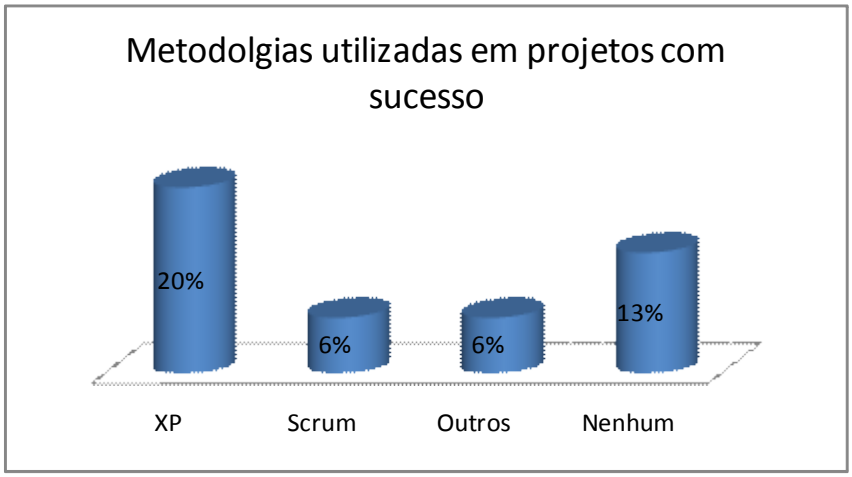


Figura 21 – Questão 6

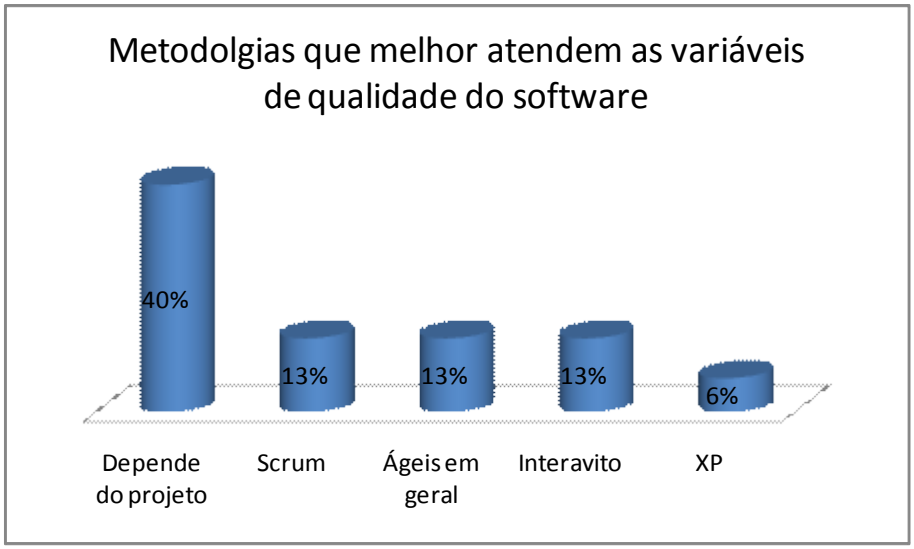


Figura 22 – Questão 7

Segundo os desenvolvedores, mesmo os que ainda não fazem uso de alguma metodologia ágil afirmam que são maleáveis de se trabalhar por isso é mais fácil de adequar os seus mecanismos, outros já dizem depender do projeto para que as variáveis de qualidade de *software* sejam atendidas, de acordo com cada metodologia.

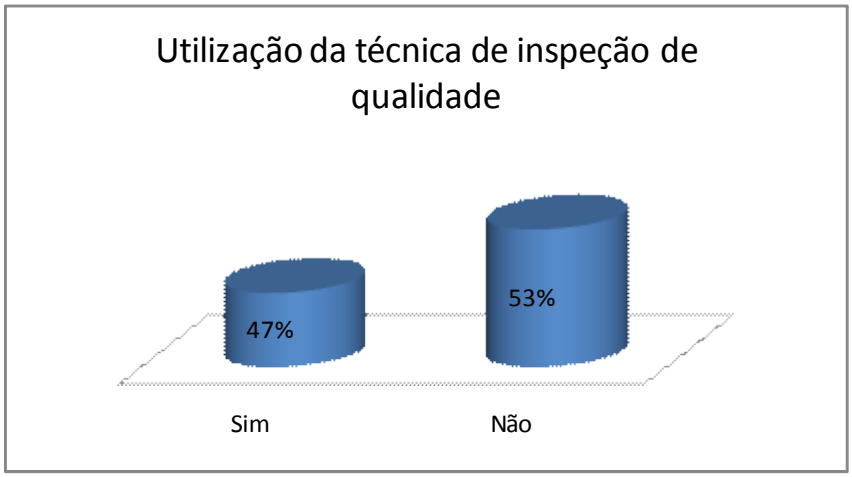


Figura 23 – Questão 8

A técnica de inspeção que se refere a uma questão de testes, interferindo diretamente na qualidade de *software* ainda não é uma prática majoritária dentro do desenvolvimento realizado pelos entrevistados.

6 CONSIDERAÇÕES FINAIS

Espera-se através deste instrumento de pesquisa ter alcançando o objetivo de somar a comunidade local perspectivas diferentes quanto ao uso de processos no desenvolvimento de software.

Conclui-se, por meio das ferramentas de pesquisa utilizada que os profissionais que estão sendo inseridos atualmente no mercado estão mais abertos quando o assunto se trata de inovação e experimentação, visto que estes já haviam trabalhado com vários tipos de metodologias. Os profissionais que já atuam na área a algum tempo possuem certa resistência até mesmo quanto ao assunto.

Também é possível verificar que mesmo os profissionais que não trabalham atualmente com alguma metodologia em estudo, concordam que para atender as principais características de qualidade do desenvolvimento de software, as metodologias ágeis são as que melhor respondem a este padrão.

A contribuição desta pesquisa é deixar para a comunidade regional de desenvolvimento de software uma fonte de pesquisa e apoio para trabalhar o desenvolvimento de software dentro de qualquer contexto.

Referencias Bibliográficas

Humphrey, Watts S., ***Gerenciamento de Processo de Software***. Addison-Wesley Publishing,Company, Massachussets, 1990.

Introdução ao Scrum. Disponível em <http://improveit.com.br/scrum>. Acessado em 12 ago. 2009

KNIBERG, Henrik. **Scrum e Xp direto das Trincheiras**. C4 Media, 2007

MANIFESTO ÁGIL. Disponível em <http://agilemanifesto.org>. Acessado em 16 set. 2009.

PETERS, J. F. **Engenharia de Software**. Campus, 2001.

PETERS, JAMES F. **Engenharia de software: teoria e prática**, 2001.

PRESSMAN, R. **Engenharia de Software**. Makron Books, 2006.

PRESSMAN, R. S. **Engenharia de Software, uma abordagem prática**. 4Ed. McGraw-Hill, 1997.

PRESSMAN, R. S. **Engenharia de Software, Pearson Education do Bra**sil, 1995

PFLEEGER, Shari Lawrence, **Engenharia de Software: teoria e prática**, 2. Ed., 2004.

SCHWARTZ, J. I. , **Construção de Software. In: Estratégia Praticas para Desenvolvimento de Grandes Sistemas**. Menlo Park: Addison-Wesley, 1. Ed., 1975.

SOMMERVILLE, I. *Engenharia de Software*. 5. ed. Addison-Wesley, 1995.

STANDISH GROUP. **Chaos Report.** Disponível em <http://www.standishgroup.com/newsroom/chaos_2009.php> 11 out. 2009 Acessado em 18. Mai. 2009

TELES, V. M. **Extreme Programming**. Disponível em <http://www.improveit.com.br/xp>. Último acesso em 10 de abril de 2009.

TELES, V. M. **Extreme Programming: Aprenda como encantar seus usuários desenvolvendo software com agilidade e alta qualidade.** São Paulo - SP: Novatec Editora Ltda, 2004.

TOM, Gilb. FINZI, Susannah. **Principles of Software Engineering Management**. 1988.

APÊNDICE A

FORMULÁRIO DE PESQUISA ACADÊMICA_

Este instrumento se destina coletar informações para o trabalho de conclusão do curso de Graduação em Sistemas de Informação da UEG. Ele se propõe a investigar questões que indicam a utilização de técnicas de métodos ágeis no processo de desenvolvimento de software.

1. Há quanto tempo você trabalha com desenvolvimento de software?

( ) menos de 1 ano

( ) de 1 a 3 anos

( ) de 3 a 5 anos

( ) mais de 5 anos

2. A empresa onde você trabalha possui alguma metodologia definida de desenvolvimento?

( ) Não.

( ) Sim.

3. Caso a resposta anterior seja afirmativa, utiliza alguma metodologia de desenvolvimento ágil?

( ) Não

( ) Sim

4. Há quanto tempo você participa de projetos que utilizam metodologias ágeis?

( ) menos de 6 meses

( ) de 6 meses a 1 ano

( ) de 1 a 2 anos

( ) mais de 2 anos

( ) não trabalho

5. Com quais metodologias de desenvolvimento de software abaixo você trabalha ou já trabalhou?

( ) XP

( ) Scrum

( ) FDD

( ) Crystal/Clear

( ) Cascata

( ) RAD

( ) Iterativo ou Incremental

( ) Outros

6. Considerando os projetos em que teve participação, o que foi mais bem sucedido fez uso de alguma metodologia? Se a resposta for sim, qual?

( ) XP

( ) Scrum

( ) FDD

( ) Crystal/Clear

( ) DSDM

( ) ASD

( ) Outros

7. Em relação a custo, funcionalidade, prazo e implantação, os projetos que melhor responderam a essas variáveis utilizavam qual metodologia?

( ) XP

( ) Scrum

( ) FDD

( ) Crystal/Clear

( ) DSDM

( ) ASD

( ) Outros

8. Você utiliza alguma técnica de inspeção para avaliação da qualidade do software?

( ) sim

( ) não