Fundação Educacional do Município de Assis
Instituto Municípal de Ensino Superior de Assis - IMESA

Marcel Carvalho Manfio

ENGENHARIA DE SOFTWARE: uma nova abordagem.

Assis, SP

2012

Marcel Carvalho Manfio

ENGENHARIA DE SOFTWARE: uma nova abordagem.

Trabalho de Conclusão de Curso apresentado ao Curso de Bacharelado em Ciência da Computação do Instituto Municipal do Ensino Superior de Assis – IMESA e Fundação Educacional do Município de Assis – FEMA, como requisito para a obtenção do Certificado de Conclusão.

Orientador: Luiz Carlos Begosso
Área de Atuação: Ciências da Computação

Assis, SP

2012

FICHA CATALOGRÁFICA

Manfio, Marcel

ENGENHARAI DE SOFTWARE: uma nova abordagem / Marcel Carvalho Manfio. Fundação Educacional do Município de Assis - FEMA – Assis, 2012 33p

Orientador: Drº Luiz Carlos Begosso

Trabalho de Qualificação – Instituto de Ensino Superior de Assis – IMESA

1. Engenharia de Software Orientado a Agentes

2. Engenharia de Software Orientado a Aspectos

3. Desenvolvimento Ágil

CDD: 001.6

Biblioteca da FEMA

ENGENHARIA DE SOFTWARE: uma nova abordagem.

Trabalho de Conclusão de Curso apresentado ao Instituto Municipal de Ensino Superior de Assis, como requisito do Curso de Graduação, analisado pela seguinte comissão examinadora:

Orientador: Luiz Carlos Begosso
Analisador: Domingos de Carvalho Villela Junior

Assis, SP

2012

DEDICATÓRIA

Dedico este trabalho a minha família, e ao professor Luiz Carlos Begosso que me ajudou a concluir este trabalho.

AGRADECIMENTOS

Agradeço, a minha família por me apoiar sempre em todas as minhas decisões. A todos os professores do curso de Ciências da Computação por terem dado toda a base necessária para o desenvolvimento deste trabalho.

E em especial ao professor Luiz Carlos Begosso por toda a ajuda oferecida, e a confiança para que este trabalho pudesse ser concluído.

RESUMO

Este trabalho promove discussão sobre alguns paradigmas de Engenharia de Software utilizados para o desenvolvimento de sistemas, apresentando seus conceitos e melhores aplicações. Os paradigmas apresentados serão Engenharia de Software Orientada a Agentes, Desenvolvimento de Software Orientado a Aspectos e Desenvolvimento Ágil (XP, Scrum). O objetivo principal deste trabalho é demonstrar as principais características de cada paradigma e em que área cada um deles é melhor aplicado. Ao final são apresentadas as características de cada um deles, visando contribuir para um desenvolvimento de softwares complexos com qualidade, manutenibilidade, aproveitamento de código e agilidade no desenvolvimento.

Palavras-chave: Engenharia de Software Orientado a Agentes, Engenharia de Software Orientado a Aspectos, Desenvolvimento Ágil.

ABSTRACT

This paper presents some paradigms of software engineering used to develop systems, concepts and presenting their best applications. The paradigms are presented Oriented Software Engineering Agents, Development of Aspect-Oriented Software Development and Agile (XP, Scrum). The main objective of this paper is to demonstrate the main features of each paradigm and each paradigm in which area is best applied, after presenting a mix of the best characteristics of each, to contribute to development of complex software quality, maintainability, use code and agility development.

Keywords: Agent-Oriented Software Engineering, *Aspect*-Oriented Software Engineering Agile Development.

LISTA DE ILUSTRAÇÕES

Figura 1- As interações de um agente genérico com seu ambiente.(Russel e Norvig, 1995). 19

Figura 2 – Utilização de pacotes para expressar protocolos aninhados. (GIRARDI, 2004). 25

Figura 3 – Exemplo de um diagrama de sequência entre agentes. (Odell, Parunak e Bauer, 2000). 26

Figura 4 – Exemplo de um diagrama de atividades. (GIRARDI, 2004). 26

Figura 5 - Exemplo de um diagrama de estados. (GIRARDI, 2004). 27

Figura 6 - Representação de classes em Orientação a Objetos. 31

Figura 7 - Representação de classes em Orientação a Aspectos.....32

Figura 8 – Custo de Mudanças, métodos tradicionais.(Luiz, 2010).....34

Figura 9 – Custo de Mudanças, métodos ágeis.(Luiz, 2010).....35

Figura 10 – Ciclo de Vida do Modelo Cascata. (SHORE, WARDEN, 2008).....38

Figura 11 – Ciclo de Vida da Programação XP.....39

Figura 12 – Agile Scrum Sprint Process Flow.....49

Figura 13 – Fases do Desenvolvimento utilizando a nova abordagem.....55

SUMÁRIO

1. INTRODUÇÃO.....12

2.ENGENHARIA DE SOFTWARE.....16

3.ENGENHARIA DE SOFTWARE ORIENTADO A AGENTES.....18

3.1.AGENTE 18

3.2.SISTEMA MULTIAGENTE (SMA).....20

3.2.1.Cooperação Entre Agentes.....21

3.2.1.1. Arquitetura Quadro-Negro.......21

3.2.1.2. Arquitetura de Troca de Mensagens.....22

3.2.1.3. Arquitetura Federativa.....22

3.2.2. Coordenação entre Agentes.....22

3.2.3. Comunicação entre Agentes.....23

3.3. TÉCNICAS PARA ANÁLISE DE REQUISITOS.....24

3.4. AGENT UML.....24

3.5. ÁREAS DE APLICAÇÃO.....27

3.6. OBSTÁCULOS NO DESENVOLVIMENTO DE SOFTWARE ORIENTADO A AGENTES.....28

4.DESENVOLVIMENTO DE SOFTWARE ORIENTADO A ASPECTOS.....29

4.2. ASPECTO.....30

4.3. UML PARA ORIENTAÇÃO A ASPECTO.....31

5. DESENVOLVIMENTO ÁGIL.....33

5.1. EXTREME PROGRAMMING (XP).....37
5.1.1. Comunicação.....39
5.1.2. Simplicidade.....40
5.1.3. Feedback.....40
5.1.4. Coragem.....41
5.1.5. Respeito..... 41
5.2. COMO FUNCIONA O XP.....42
5.2.1. Planejamento.....43
5.2.2. Análise.....43
5.2.3. Projeto.....43
5.2.4. Codificação.....44
5.2.5. Testes..... 44
5.2.6. Implementação..... 45
5.3. ÁREAS DE APLICAÇÃO..... 45
5.4. OBSTÁCULOS NO EXTREME PROGRAMMING.....45
5.5. SCRUM.....45
5.5.1. Product Owner.....47
5.5.2. Scrum Master.....47
5.5.3. Scrum Team.....48
5.5.4. Sprint.....48
5.6. APLICAÇÕES DO SCRUM.....49
6. UMA NOVA ABORDAGEM.....50
7. CONCLUSÃO.....57
REFERÊNCIAS.....58

1. INTRODUÇÃO

A Engenharia de Software é algo muito discutido no meio do desenvolvimento de software, pois existem várias maneiras e métodos para se desenvolver um software.

O que torna esta área algo muito importante no processo de desenvolvimento é a maneira como o software será desenvolvido, organizado, planejado, qual a melhor tecnologia para desenvolver, como será documentado, seus diagramas e assim poder contribuir com a produção de um software com características de qualidade.

Antes do surgimento da Engenharia de Software, todos os softwares eram desenvolvidos apenas com base em alguns requisitos determinados pelo cliente. Assim que estes requisitos eram identificados o desenvolvimento começava, porém muitos clientes não tem conhecimento em desenvolvimento de softwares, tornando várias vezes a interpretação desses requisitos, pelo desenvolvedor, algo muito diferente do que o cliente realmente necessita. Resultando em um software que não atende as necessidades do cliente.

Quando um software não é especificado de forma correta, podem ocorrer vários problemas decorrentes desta falta de especificação, por exemplo: atraso na entrega, mudanças muito impactantes, um software que não atende as necessidades do cliente e um produto de baixa qualidade, e decorrente de todos estes erros o custo acaba se tornando muito alto fazendo com que o cliente ou a empresa de desenvolvimento tenha um grande prejuízo, ou porque o software não atende as reais necessidades ou porque o software demandou muito mais tempo do que o esperado.

Assim surgiram alguns paradigmas de Engenharia de Software: Engenharia de Software Orientada a Agentes, Engenharia de Software Orientada a Aspectos, metodologias de Desenvolvimento Ágil, entre outros. Estes paradigmas são métodos utilizados para produzir software seguindo um planejamento pré-determinado onde

um paradigma é escolhido levando em consideração a natureza do projeto e do software a ser desenvolvido.

Alguns paradigmas tradicionais como Ciclo de Vida Clássico ou Cascata, Prototipação, Modelo Espiral e Técnicas de Quarta Geração ainda são muito usados, porém ao longo do tempo a complexidade dos softwares vem aumentando, tornando necessária uma abordagem mais flexível e com a possibilidade de efetuar mudanças ao longo do desenvolvimento sem prejudicar a qualidade do produto final. Com o objetivo de contribuir com a melhoria do software surgiram alguns paradigmas de Engenharia de Software como fruto da junção e aprimoramento dos paradigmas já existentes.

“Os paradigmas podem e devem ser combinados de forma que as potencialidades de cada um possam ser obtidas num único projeto. Os paradigmas da Engenharia de Software podem ser combinados num único esforço de desenvolvimento de software. O trabalho inicia-se com a definição dos objetivos, alternativas e restrições. A partir desse ponto qualquer caminho pode ser seguido.” (PRESSMAN, 1995).

Levando em consideração a citação de Pressman, neste trabalho de Conclusão de Curso serão apresentados três paradigmas de desenvolvimento, a saber: Engenharia de Software Orientada a Agentes, Engenharia de Software Orientada a Aspectos e metodologias ágeis de desenvolvimento. Ao final serão apresentadas suas melhores práticas de aplicação para o desenvolvimento de software. A contribuição deste trabalho reside no fato da apresentação de nova abordagem para o desenvolvimento de software.

A Engenharia de Software Orientada a Agentes é um método novo no mercado que surgiu para suprir algumas necessidades no desenvolvimento de softwares complexos, as consequências de se desenvolver um software de tamanha proporção utilizando os paradigmas tradicionais não são boas, pois estes paradigmas possuem ferramentas de desenvolvimento apropriadas para a construção de sistemas com comportamento pré-determinado, assim se algo inesperado acontecer o programa não saberá se comportar de forma apropriada.

Em um sistema grande e complexo as respostas obtidas pelo software podem ser variadas devido ao seu tamanho, assim paradigmas tradicionais não seriam uma

boa opção. Para isso foi criada a Engenharia de Software Orientada a Agentes como uma alternativa, onde um agente precisa se comportar de forma bastante flexível e autônoma, pois ele precisa atuar em um cenário computacional não previsível e com rápidas mudanças. Assim a orientação a agentes vem facilitar o desenvolvimento de grandes sistemas complexos que necessitam de respostas rápidas.

A Engenharia de Software Orientada a Aspectos foi criada para facilitar o desenvolvimento tornando os problemas mais simples para serem resolvidos e posteriormente, se necessário, reutilizá-los em outras funcionalidades. Assim o programa é dividido em vários problemas, que são denominados de aspectos, e cada um deles desempenha apenas uma tarefa, assim se outro procedimento precisar desta funcionalidade não será necessário implementá-lo novamente. Com isso podemos dividir todo o sistema em várias partes, por exemplo: persistência de dados, interfaces, segurança, distribuição, auditoria, registro de logs, controle de ocorrência, tratamento de erros e outros que normalmente são encontrados na programação orientada a objetos em vários pontos do código.

Utilizando a Engenharia de Software Orientada a Aspectos a complexidade é reduzida no desenvolvimento do código-fonte contribuindo com desenvolvimento mais rápido e com melhor qualidade.

Já o Desenvolvimento Ágil engloba um conjunto de metodologias que está sendo utilizado há algum tempo e com bons resultados no ganho de produtividade. Na Engenharia de Software tradicional há uma grande ênfase em projetar o sistema antes de construí-lo, assim as mudanças que aparecem ao longo do desenvolvimento do projeto acabam se tornando algo desagradável, pois é necessário voltar e mudar todo o projeto novamente. O Desenvolvimento Ágil prioriza as mudanças que ocorrem no início, no meio e no fim do desenvolvimento. Assim é necessária uma técnica de desenvolvimento que permita fazer várias mudanças sem comprometer a qualidade do software.

Métodos de Desenvolvimento Ágil são uma coleção de metodologias que aplicam práticas, guiadas por princípios e valores que são aplicados para um desenvolvimento de software com qualidade. São exemplos da metodologia ágil: o Extreme Programming e o Scrum. Assim serão apresentadas estas formas de

desenvolvimento de software de forma detalhada a fim de se esclarecer cada uma delas e suas melhores formas de utilização.

Para atingir os objetivos propostos, este trabalho está dividido em 7 capítulos. O primeiro capítulo contextualiza e apresenta os objetivos para o trabalho. O segundo situa a Engenharia de Software, suas origens, desenvolvimento e importância. O terceiro capítulo apresenta a Engenharia de Software Orientada a Agentes. O capítulo quatro destaca as características da Engenharia de Software Orientada a Aspectos. No quinto capítulo são apresentados o Extreme Programming e o SCRUM, utilizados no desenvolvimento ágil de software. O sexto capítulo faz a proposta de uma nova abordagem para o desenvolvimento de software. O trabalho é finalizado com discussões e direcionamentos futuros, no capítulo sete.

2. ENGENHARIA DE SOFTWARE

Antes do surgimento da Engenharia de Software, os sistemas eram desenvolvidos sem qualquer tipo de planejamento, apenas eram levantados alguns requisitos e já era iniciada a fase de desenvolvimento, não era utilizada nenhuma padronização para o processo de desenvolvimento. Porém ao desenvolver sistemas com um grau maior de complexidade é extremamente necessário fazer um planejamento antes de começar o desenvolvimento, pois sem o devido planejamento o desenvolvimento do sistema acaba tomando rumos diferentes e ao final é entregue um produto totalmente diferente do qual foi solicitado pelo cliente. Assim a padronização no processo de desenvolvimento acabou se tornando algo necessário.

Conforme Bauer et al.(1969 apud PRESSMAN et al., 1995, P. 17), a Engenharia de Software é a criação e a utilização de sólidos princípios de engenharia a fim de obter softwares econômicos que sejam confiáveis e que trabalhem eficientemente em máquinas reais.

Segundo Pressman (1995) a engenharia de software é uma tecnologia em camadas onde todas as engenharias devem estar apoiadas em um compromisso organizacional de qualidade.

A Engenharia de Software contém vários métodos e ferramentas que fornecem toda a estrutura necessária para desenvolver softwares de qualidade onde os métodos abrangendo análise de requisitos, comunicação, construção de programas, modelagem de projeto, testes e manutenção e as ferramentas fornecem apoio automatizado para os métodos.

No entanto a Engenharia de Software Tradicional propõe a utilização de uma grande documentação do sistema em questão, onde são apresentados todas as funcionalidades do sistema, diagramas de classe, diagramas de sequência, diagramas de casos de uso, entre outros. Essa documentação é feita a partir dos levantamentos de requisitos e concluída para assim começar o desenvolvimento.

Os desenvolvedores envolvidos com o projeto começam a programar tendo como base todas as funcionalidades especificadas na documentação, porém durante o desenvolvimento podem surgir diversas outras funcionalidades e algumas que já foram especificadas não serem mais necessário, fazendo com que o documento inicial fique totalmente em desacordo com o sistema. Assim é preciso escrever o documento novamente para que ele fique de acordo com o sistema para que no futuro quando alguma manutenção for necessária o sistema esteja totalmente documentado. A documentação então acaba se tornando um atraso para o cronograma do projeto, pois ao ter que escreve-la duas vezes o tempo dedicado a esta tarefa é muito grande.

3. ENGENHARIA DE SOFTWARE ORIENTADO A AGENTES

Com a crescente demanda por sistemas grandes e complexos ao longo do tempo surgiram novos paradigmas de Engenharia de Software visando aprimorar o desenvolvimento de sistemas cada vez melhores e que atendam de fato as necessidades do cliente.

Assim alguns paradigmas tradicionais como Ciclo de Vida Clássico ou Cascata, Prototipação, Modelo Espiral e Técnicas de Quarta Geração estão dando espaço para novos paradigmas como a Orientação a Agentes.

Devido a grande complexidade envolvida nestes sistemas, torna-se necessário uma abordagem mais flexível e com a possibilidade de efetuar mudanças ao longo do desenvolvimento sem prejudicar a qualidade do produto final.

A Engenharia de Software tradicional possui ferramentas de desenvolvimento apropriadas para a construção de sistemas com comportamento pré-determinado, consequentemente se algo inesperado, que não foi especificado no desenvolvimento do projeto, acontece, o sistema falha. Para minimizar essa situação a Engenharia de Software Orientada a Agentes se apresenta como alternativa.

Um agente precisa se comportar de forma bastante flexível e autônoma, pois deve atuar em um cenário computacional não previsível e com rápidas mudanças. Assim a orientação a agentes vem facilitar o desenvolvimento de grandes sistemas complexos que necessitam de respostas rápidas.

3.1. AGENTE

De acordo com Tahara (1999), agentes são unidades de software que podem lidar com mudanças ambientais e com vários requisitos de redes abertas através de características como autonomia, mobilidade, inteligência, cooperação e reatividade.

Assim um agente é capaz de realizar tarefas específicas através de sensores e executores para atingir um objetivo. No entanto para que os agentes tomem decisões apropriadas para cada entrada de dados, é preciso definir corretamente como serão construídas suas percepções e suas ações.

Pode se dizer então que um agente é um componente com interface bem definida que interage com o ambiente onde está situado percebendo as mudanças ao seu redor agindo para modificá-las, sempre tendo controle sobre seu estado e seu comportamento, e ao mesmo tempo sendo reativo e proativo. A figura 1 ilustra as interações de um agente genérico com um certo ambiente.

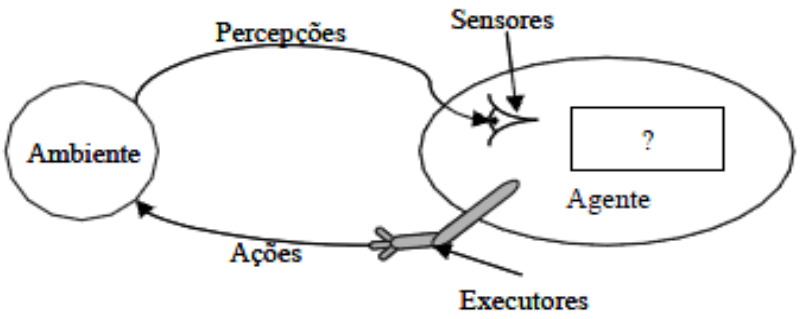

Figura 1- As interações de um agente genérico com seu ambiente.

(Russel e Norvig, 1995).

Não há uma definição exata para agente, contudo neste trabalho será adotada a definição proposta por Wooldridge (2002): “Um agente é um sistema computacional encapsulado que está situado em algum ambiente e é capaz de ação flexível autônoma neste ambiente, a fim de alcançar seus objetivos de projeto”

Para se ter um software orientado a agentes de qualidade é preciso que cada agente possua as seguintes propriedades, de acordo com Silva (2004):

- Autonomia: agir de forma direta sem intervenção de outros agentes ou humanos.

- Deliberatividade: tomar decisões considerando experiências anteriores e informações vinda do ambiente em que ele está atuando.

- Reatividade: responder a tempo as mudanças que ocorre no ambiente.

- Organização: se dá a partir de grupos de agentes que se comunicam de forma regular por um conjunto de normas sociais para atingir um objetivo geral.

- Habilidade de socializar-se: capacidade de interagir com outros agentes em uma sociedade.

- Interação: habilidade de se comunicar de forma regular com o ambiente e outros agentes.

3.2. SISTEMA MULTIAGENTE (SMA)

Quando se tem um problema grande e muito complexo para resolver, é essencial repartimos este problema em vários problemas menores e menos complexos para assim buscar uma resolução mais fácil e exata, consequentemente resolvendo todos os problemas menores, ao fim o problema inicial estará resolvido.

Levando isto em consideração podemos dividir um sistema complexo em vários agentes, tendo assim cada agente com uma complexidade menor e mais fácil para ser resolvida.

Para que esta abordagem dê certo é necessário que todos os agentes envolvidos no sistema interajam entre si para, a partir de seus objetivos específicos, alcançar o objetivo geral do sistema.

3.2.1. Cooperação entre Agentes

É a partir da cooperação entre os agentes que eles expressam suas necessidades e seus resultados para assim serem ou não utilizados por outros agentes dentro da sociedade, e é a partir destes resultados ou necessidades que há efetiva cooperação entre os agentes para atingir o objetivo geral.

A cooperação entre os agentes pode ser realizada de duas formas:

- Partilha de Tarefas: ocorre quando algum agente necessita do auxílio de outro agente para executar determinada tarefa.

- Partilha de Resultados: ocorre quando um agente necessita do resultado de outro agente para executar determinada tarefa.

Assim é necessário que todos os agentes conheçam a localização exata uns dos outros dentro da sociedade. Para isso é preciso de uma arquitetura bem definida.

As principais abordagens de arquitetura SMA são: quadro-negro, troca de mensagem e federativa.

3.2.1.1. Arquitetura Quadro-Negro

Segundo Girardi (2004), na arquitetura quadro-negro os agentes não se comunicam entre si de forma direta, mas sim através de um quadro-negro. Esta é uma estrutura persistente onde existe uma divisão em regiões ou níveis, visando facilitar a troca de

informações entre os agentes da sociedade. É um meio onde os agentes escrevem e leem mensagens que futuramente serão utilizadas para realizar outras tarefas. Assim o quadro-negro é uma memória de compartilhamento onde todos os agentes tem acesso.

Utilizando esta arquitetura um sistema muito grande ou em tempo real pode acabar ficando muito lento, pois as ações de escrita e leitura podem ser demoradas.

3.2.1.2. Arquitetura de Troca de Mensagens

Na Arquitetura de Troca de Mensagens os agentes comunicam-se de forma direta uns com os outros por meio de mensagens assíncronas, para que isso ocorra de forma correta é necessário que os agentes saibam a localização de todos os outros dentro da sociedade. Assim é preciso de um protocolo de comunicação que dita as regras de como as mensagens devem ser trocadas.

3.2.1.3. Arquitetura Federativa

De acordo com Girar (2004), a Arquitetura Federativa é utilizada quando o número de agentes é muito grande, assim todos eles são divididos em grupos ou federações segundo o critério utilizado para o agrupamento. Entre eles existe um facilitador que é responsável por receber e direcionar as mensagens para o agente correto em seu grupo.

Arquitetura Federativa diminui o fluxo de mensagens desnecessárias entre os agentes que formam a sociedade.

3.2.2. Coordenação entre Agentes

É a maneira com que os agentes estão organizados a fim de cooperar uns com os outros para chegar a um objetivo geral. Para isto existem dois mecanismos básicos para coordenar os agentes, proposto por Paraiso (1997).

- Mestre-Escravo: existem duas classes de agentes, os gerentes (mestres) e os trabalhadores (escravos).Os trabalhadores são coordenados pelos gerentes que distribuem as tarefas entre eles a espera de resultados.

- Mecanismo de Mercado: todos os agentes estão localizados no mesmo nível e sabem o que cada um deve desempenhar, visando diminuir a quantidade de mensagens trocadas.

3.2.3. Comunicação entre Agentes

A Comunicação entre Agentes é muito importante, característica da qual os agentes inteligentes necessitam para alcançar seus objetivos. Para haver a comunicação de forma que todos os agentes entendam as mensagens é utilizada uma linguagem de comunicação ACL(Agent Communication Language) que foi definida a partir dos seguintes critérios por Finin (1993):

- Transporte: forma com que os agentes enviam e recebem mensagens.
- Linguagem: o significado de mensagens individuais.
- Política: como os agentes estruturam conversações.
- Arquitetura: como conectar sistemas em concordância com os agentes.

Para a definição da linguagem foram utilizadas duas abordagens: a procedural onde as mensagens são trocadas de forma direta, podendo transportar programas

completos e comandos para a execução de outras funções no lado receptor, e a declarativa, realizada através da troca de declarações.

3.3. TÉCNICAS PARA ANÁLISE DE REQUISITOS

A fase de análise tem como objetivo definir o que o software deve fazer de acordo com o paradigma de desenvolvimento escolhido. Os métodos para análise de requisitos de sistemas multiagentes focalizam em sua maioria a modelagem de objetivos, agentes e interações de indivíduos de uma organização. Por ser algo recente, a engenharia de software orientada a agentes, ainda não existe uma definição exata dos conceitos.

Dentro de uma organização existem objetivos gerais e específicos que devem ser alcançados, assim a partir do alcance de todos os objetivos específicos é possível alcançar o objetivo geral. Desta forma é preciso modelar os objetivos, agentes e interações para que o sistema funcione.

- Modelagem de Objetivos: a partir da identificação de um problema que o sistema deve resolver, é obtido um objetivo geral, e assim é possível fracioná-lo em diversos objetivos específicos tornando o alcance do objetivo geral mais simples.

- Modelagem de Agentes: para cada objetivo específico, são identificadas as responsabilidades que serão exercitadas pelos agentes internos e externos da organização.

- Modelagem de Interação: através de uma análise de suas atividades respectivas são identificadas as interações entre os agentes internos e externos.

3.4. AGENT UML

Assim como no desenvolvimento de software orientado a objeto, que utiliza a UML, foi criada a AUML (*Agent Unified Modeling Language*). A AUML é uma linguagem de especificação para sistemas multiagentes onde são especificados Protocolos de Interações entre Agentes (AIP) que foi criado por OMG (Object Management Group) e a FIPA(Foundation for Intelligent Physical Agents).

Ela propõe uma representação gráfica em três camadas para os protocolos de interação, essas camadas definem os padrões que serão utilizados para a comunicação entre os agentes, definindo as sequências das mensagens e as restrições sobre o conteúdo delas.

- Camada 1: representação dos protocolos de interações globais do sistema.
- Camada 2: representação das interações entre os agentes.
- Camada 3: representação do processamento interno do agente.

Na camada 1, os protocolos oferecem algumas soluções reutilizáveis que podem ser aplicadas em vários tipos de sequências de mensagens, para isso ela oferece uma técnica denominada de pacotes. Os pacotes são utilizados para organizar os modelos em grupos, podendo assim um pacote estar dentro de outro pacote como exemplificado na figura 2.

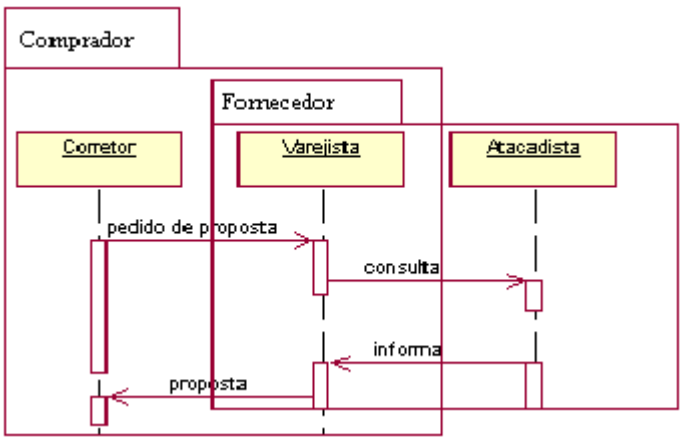

Figura 2 – Utilização de pacotes para expressar protocolos aninhados. (GIRARDI, 2004).

A camada 2 utiliza diagramas de colaboração (figura 3), atividade (figura 4), estado (figura 5) e sequência (figura 6) que possuem a finalidade de mostrar como os agentes interagem trocando mensagens.

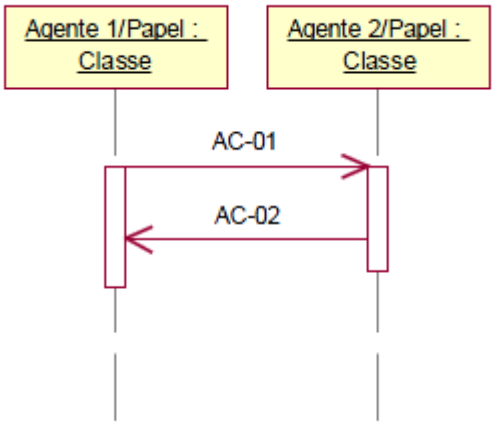

Figura 3 – Exemplo de um diagrama de sequência entre agentes. (Odell, Parunak e Bauer, 2000).

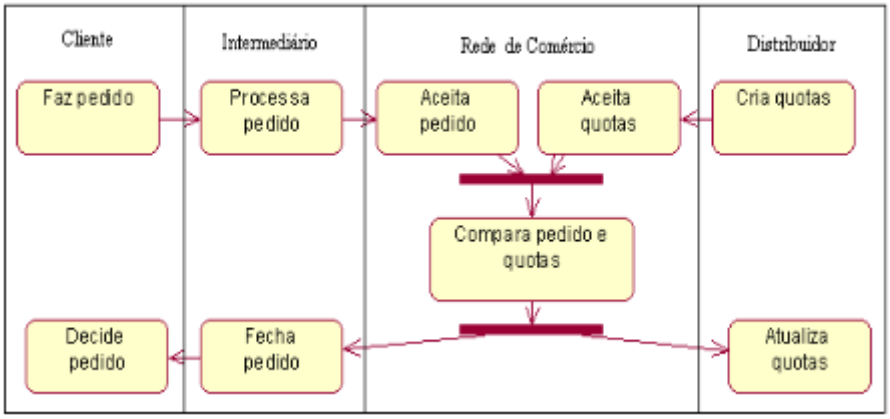

Figura 4 – Exemplo de um diagrama de atividades. (GIRARDI, 2004).

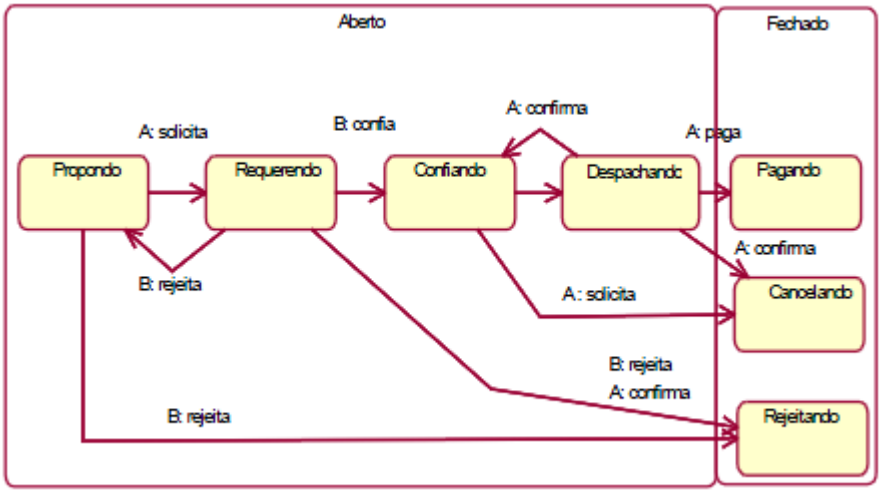

Figura 5 - Exemplo de um diagrama de estados. (GIRARDI, 2004).

Já na camada 3 é abordado o comportamento interno dos agentes utilizando-se de diagramas de atividade e estado.

3.5. ÁREAS DE APLICAÇÃO

A Engenharia de Software Orientada a Agentes está sendo aplicada nas seguintes áreas: industrial, comercial, de entretenimento e médica.

Alguns exemplos dessas aplicações:

- Fabricação: projeto e configuração de projetos de fabricação, cronograma e controle da fabricação de robôs.

- Controle de tráfego aéreo: agentes utilizados para representar os equipamentos de aviação.

- Jogos interativos: agentes animados jogam com humanos ou outros agentes.

- Sistemas de transporte: onde agentes representam os veículos ou usuários de transporte.

- Médica: monitoramento de pacientes.

- Comércio eletrônico: agentes são utilizados para automatizar processos de compras eletrônicas, por exemplo, ajudando o usuário a decidir o que comprar.

3.6. OBSTÁCULOS NO DESENVOLVIMENTO DE SOFTWARE ORIENTADO A AGENTES

Segundo Wooldridge (2001), por ser algo muito complexo, o Desenvolvimento de Software Orientado a Agentes, é preciso tomar muito cuidado no desenvolvimento de software orientado a agentes, pois as interações entre os agentes devem ser muito bem definidas no projeto. Por serem autônomos, os agentes tem capacidade de reagir sozinhos de forma diferenciada, podendo assim fazer algo muito diferente daquilo que tenha sido especificado anteriormente. Para que tudo ocorra perfeitamente é preciso que a tomada de decisão de cada agente seja sensível ao contexto, tornando a imprevisibilidade muito pequena.

4.DESENVOLVIMENTO DE SOFTWARE ORIENTADO A ASPECTOS

O paradigma de Orientação a Aspectos foi criado para facilitar o desenvolvimento tornando os problemas mais simples para serem resolvidos e posteriormente, se necessário, reutilizá-los em outras funcionalidades. Assim o programa é dividido em requisitos funcionais e não funcionais.

Na Engenharia de Software Orientada a Objetos o conceito de requisitos funcionais e não funcionais também existe, porém esses requisitos se entrelaçam fazendo com que as funcionalidades dos dois tornem-se dependentes umas das outras, e se for preciso a utilização destas funcionalidades por outra classe ao longo do sistema, será necessário implementá-las novamente. Para resolver este problema de entrelaçamento dos requisitos, e de fato separá-los, surgiu o paradigma de Orientação a Aspecto que é uma extensão do paradigma de Orientação a Objetos que propõem a divisão dos requisitos.

Assim com está separação é possível trabalhar em cada um deles de forma separada e aprofundada tornando a resolução do problema mais fácil e de forma mais adequada e como consequência da separação a manutenção acaba se tornando mais fácil, pois se várias classes utilizam determinado aspecto, basta corrigi-lo que todas essas classes estarão consequentemente corrigidas.

O Desenvolvimento de Software consiste na separação de requisitos do sistema em requisitos funcionais e não funcionais:

- Requisitos funcionais: são responsáveis por descrever o comportamento do programa, por exemplo, qual tarefa o sistema deve realizar para determinado tipo de entrada, são as regras de negócios do sistema;

- Requisitos não funcionais: são responsáveis por definir como tal comportamento do sistema deve funcionar.

A partir dos requisitos do sistema, o Desenvolvimento de Software Orientado a Aspecto trata estes requisitos como interesses (concerns) e interesses transversais (crosscutting concerns). Os interesses são as regras de negócio e os interesses transversais representam persistência, segurança, auditoria e tratamento de exceções. Se implementados de forma integrada esses interesses acabam ficando todos entrelaçados dificultando a reutilização e manutenção do sistema.

Com a divisão desses interesses do sistema são obtidos os aspectos, que são os interesses transversais já separados corretamente. Um interesse transversal normalmente, em outros paradigmas, está espalhado por todo o sistema mesmo sendo implementado da mesma forma. Já no paradigma de Orientação a Aspecto um interesse transversal é implementado apenas uma vez e reutilizado para todo o sistema através de pontos de junção (joinpoints).

Assim o desenvolvimento é dividido em três etapas:

- Decomposição: é a identificação dos requisitos do sistema, sua divisão e separação em interesses e interesses transversais;

- Implementação: é a implementação de todos os interesses de forma separada.

- Recomposição: é junção de toda a implementação dos interesses.

4.2. ASPECTO

Um aspecto é um interesse transversal (crosscutting concerns) do sistema, que encapsula uma funcionalidade que atravessa outras classes do programa, como por exemplo, persistência de dados, gravação de logs, tratamento de erros, segurança, distribuição, auditoria e controle de ocorrência que normalmente são utilizados por várias classes do sistema. Os aspectos podem introduzir atributos, métodos, declarações de implementação de interface e declaração de extensão de classes.

4.3. UML PARA ORIENTAÇÃO A ASPECTO

Como a Orientação a Aspecto é uma extensão de Orientação a Objetos, a sua representação também é feita pela UML(*Unified Modeling Language*) com apenas algumas modificações.

As classes do sistema são representadas da mesma forma, utilizando um retângulo com seus atributos e métodos, e para representar os aspectos é utilizado um retângulo com o identificador <<aspect>>. Os aspectos são os interesses transversais do sistema que são ligados a classe através de um designador <<crosscut>>.

Na figura 6 estão representadas três classes na Orientação a Objetos onde cada classe implementa um método gravarLog(), tendo de ser reescrito individualmente para cada uma das classes. Já na figura 7 o método gravarLog() é tido como um aspecto, assim todas as classes que necessitarem fazer o log não precisarão reescrever o método para cada classe.

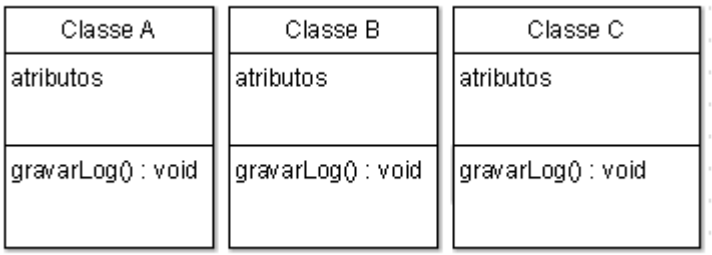

Figura 6 - Representação de classes em Orientação a Objetos.

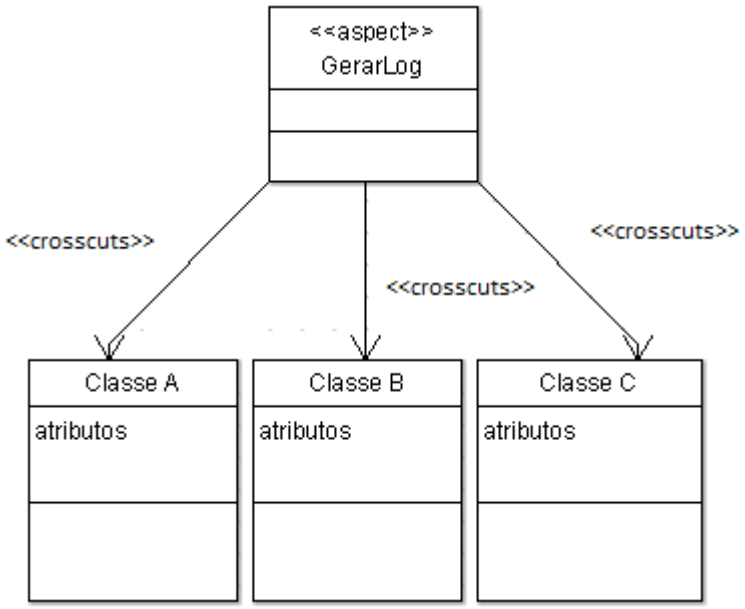

Figura 7 - Representação de classes em Orientação a Aspectos.

5. DESENVOLVIMENTO ÁGIL

O Desenvolvimento Ágil surgiu da necessidade de se produzir software de qualidade em um curto prazo de tempo. Com a crescente demanda por softwares de qualidade, as empresas tiveram de começar a diminuir os prazos de entrega, porém a complexidade envolvida nos softwares aumentou com o passar dos anos.

Há muitos anos a Engenharia de Software Tradicional, citada no segundo capítulo, deu suporte a diversos tipos de aplicações. Seus métodos estão baseados no levantamento de requisitos e em uma documentação extremamente fiel ao software, mas como hoje em dia, no processo de desenvolvimento, podem acontecer grandes mudanças ao longo do projeto, por exemplo, o cliente identifica novos requisitos durante o desenvolvimento, assim o uso da Engenharia de Software Tradicional acaba se tornando um obstáculo. Isso se dá pelo fato dos métodos tradicionais como Ciclo de Vida Clássico, Prototipação, Modelo Espiral e Técnicas de Quarta Geração ser totalmente planejados e documentados antes de começar a implementação, fazendo com que mudanças ao longo do projeto tenham um custo cada vez maior (Figura 8).

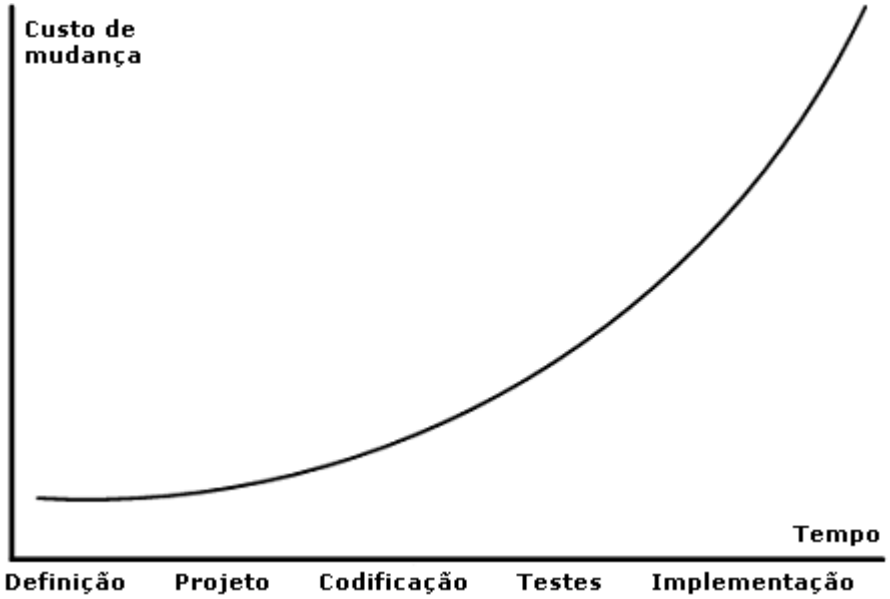

Figura 8 – Custo de mudança, métodos tradicionais.(Luiz, 2012)

Contudo como estes métodos tradicionais de desenvolvimento pregam a necessidade de projetar antes de implementar, mudanças ao longo do projeto acabam atrapalhando todo o desenvolvimento, gerando um aumento nos custos, descumprimento de prazos e diminuindo a qualidade do software, porém os softwares de hoje necessitam de mudanças constantes sem afetar nos custos (Figura 9) e o prazo, para assim ter um software de alta qualidade.

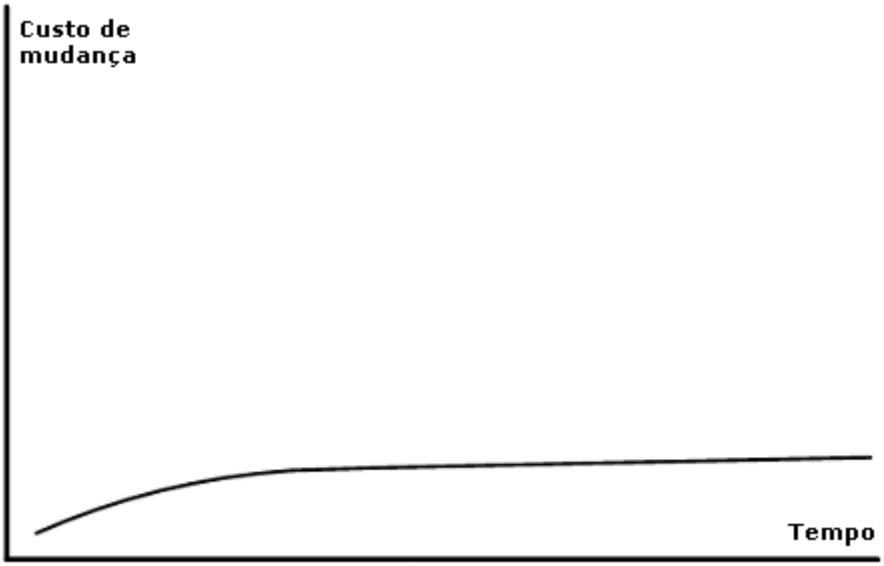

Figura 9 – Custo de Mudanças, métodos ágeis.(Luiz, 2010)

Devido as dificuldades enfrentadas para se desenvolver novos softwares com a Engenharia Tradicional um grupo de dezessete desenvolvedores fundou o Movimento Ágil, que a partir dessas necessidades encontradas, de tornar o desenvolvimento mais rápido, flexível e chegar a um produto final de ótima qualidade, criaram quatro valores fundamentais para o Desenvolvimento Ágil:

- Os indivíduos e suas interações acima de procedimentos e ferramentas;
- O funcionamento do software acima de documentação abrangente;
- A colaboração dos clientes acima da negociação de contratos;
- A capacidade de resposta à mudanças acima de um plano pré-estabelecido;

E através destes quatro valores fundamentais os desenvolvedores do Manifesto Ágil chegaram a doze princípios que devem reger as metodologias ágeis:

1. Nossa maior prioridade é satisfazer o cliente, mediante a rápida e contínua entrega de software funcional.

2. Permitir mudanças nos requisitos, mesmo que tardiamente no projeto. Processos ágeis aproveitam mudanças para o benefício do cliente.

3. Entregar software funcional frequentemente, podendo variar entre semanas ou meses, com uma preferência por escalas de tempo menores.

4. Especialistas do domínio e desenvolvedores devem trabalhar juntos diariamente durante o projeto.

5. Desenvolver projetos em torno de indivíduos motivados. Dar a eles o ambiente e apoio de que precisam, e confiar neles para a realização do trabalho.

6. O método mais prático e eficiente de se comunicar com a equipe de desenvolvimento é conversar cara-a-cara com os integrantes.

7. Software funcional é a medida primária de progresso.

8. Processos ágeis promovem desenvolvimento sustentável. Os patrocinadores, desenvolvedores e usuários devem ser capazes de manter um ritmo constante.

9. Atenção contínua a excelência técnica e bom design aumenta a agilidade.

10. Simplicidade - a arte de maximizar a quantidade de trabalho não terminado - é essencial.

11. As melhores arquiteturas, requisitos e modelos surgem de equipes auto organizadas.

12. Em intervalos regulares de tempo, a equipe reflete em como se tornar mais eficiente, então se prepara e se ajusta de acordo.

A partir desses valores e princípios foram surgindo algumas metodologias ágeis como Extreme Programming (XP) e Scrum que serão apresentadas nos próximos capítulos.

“Métodos ágeis são processos que suportam a filosofia ágil. Os métodos ágeis consistem de elementos individuais chamados práticas. As práticas incluem utilização do controle de versão, determinação de padrões de código e demonstração semanais ás partes interessadas.” (SHORE, WARDEN, 2008)

5.1. Extreme Programming (XP)

O método de desenvolvimento Extreme Programming foi criado por Kent Beck nos anos 90 nos Estados Unidos, quando ele começou a trabalhar com um sistema de folha de pagamento, de nome Sistema de Compensação Abrangente da Crysler (Chrysler Comprehensive Compensation), que então foi aplicado todo o conceito de XP, e que teve um ótimo resultado, tornando-se assim um dos métodos mais conhecidos no Desenvolvimento Ágil.

Ao se desenvolver um software utilizando XP todos os esforços da equipe são concentrados em atividades que geram resultados expressivos testando o software de maneira intensa para assim atender todas as necessidades do cliente.

Esse método surgiu visando diminuir o tempo e o custo de desenvolvimento, ser flexível a mudanças nos requisitos que surgem ao longo do processo e a entrega de um software de qualidade que realmente atenda as necessidades impostas pelo cliente. A equipe que compõem o desenvolvimento deve possuir no máximo dez integrantes, pois o XP foi criado para projetos pequenos e médios que não tenham requisitos totalmente formados e que mudam constantemente.

Em Métodos Tradicionais o processo de desenvolvimento segue sempre um ciclo sequencial, por exemplo, o Ciclo de Vida do Modelo em Cascata (Figura 10) que se inicia a partir de um plano, análise, projeto, código, teste e por fim a implementação, fazendo com que o aparecimento de novos requisitos ao longo deste processo sequencial prejudique o prazo e o custo do projeto.

Figura 10 – Ciclo de Vida do Modelo Cascata. (SHORE, WARDEN, 2008)

Já no XP este ciclo é mais curto, fazendo com que ele se repita várias vezes durante o desenvolvimento (Figura 11). Essa abordagem faz com que requisitos identificados após o término da primeira análise possa ser incorporado ao projeto sem nenhum problema.

Com o desenvolvimento utilizando XP a equipe envolvida entrega funcionalidades todas as semanas, consequentemente a equipe consegue obter um feedback do cliente, para assim dar continuidade ao projeto sem impedimentos passados e se por acaso algum erro for encontrado o tratamento deste erro acaba se tornando muito mais simples pois se trata apenas de um funcionalidade do sistema.

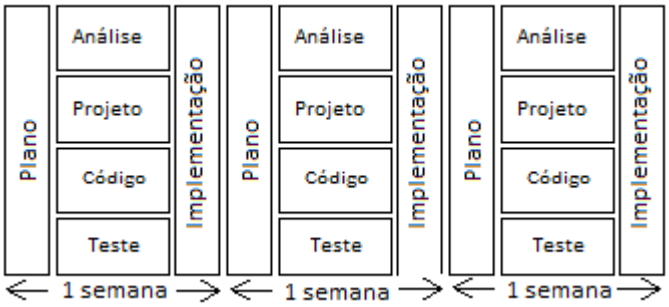

Figura 11 – Ciclo de Vida da Programação XP.

Para que o desenvolvimento do software seja feito utilizando XP é necessário seguir cinco valores ao pé da letra: Comunicação, Simplicidade, Feedback, Coragem e Respeito.

5.1.1. Comunicação

A comunicação em um projeto de software é algo muito importante para de fato atingir todos os objetos impostos no início do desenvolvimento. Pois estabelecendo uma comunicação frequente entre os desenvolvedores a solução de problemas decorrentes da implementação acabam sendo resolvidas mais rapidamente, por exemplo algum desenvolvedor tem certo conhecimento específico mais aprofundado que outro, assim se ocorrer algum erro, este erro pode ser solucionado mais rapidamente fazendo com que o processo de implementação flua de forma constante. A comunicação entre os desenvolvedores e o cliente é algo muito importante também, pois com a equipe mantendo contato frequente com o cliente alguns requisitos podem sofrer modificações que são essências para a necessidade do cliente.

Esta comunicação constante se da pelo fato de que muitos clientes não terem conhecimento da área de tecnologia, fazendo com que alguns requisitos informados por ele sejam obscuros para a equipe de desenvolvimento, no entanto ao longo do desenvolvimento esses requisitos que não foram totalmente formados podem ser esclarecidos. Porém se a comunicação não estiver presente de forma intensa um outro valor do XP, o feedback, pode ser afetado.

5.1.2. Simplicidade

A simplicidade se torna algo primordial para o desenvolvimento, pois ao ser simples o sistema atende apenas as necessidades do cliente, e isso é o que se espera do sistema. Não ser simples no desenvolvimento significa querer pensar no futuro e implementar funcionalidades que não foram especificadas pelo cliente. Isso faz com que o software acabe se tornando muito complexo, podendo acarretar diversos tipos de problemas que não seria necessário trata-los se tivesse apenas implementado o necessário. Para que a simplicidade seja aplicada de forma efetiva é preciso de bom senso e conhecimento técnico para assim manter o projeto ágil, suscetível a mudanças e simples.

5.1.3. Feedback

O feedback trata-se de uma constante comunicação entre os desenvolvedores e o cliente. Em métodos de Desenvolvimento Tradicional também encontra-se a técnica de feedback, porém esses feedbacks acontecem em um espaço de tempo muito grande, fazendo com que a identificação de erros seja mais demorada, assim podendo se torna uma tarefa complexa para resolvê-los. No XP o feedback ocorre em um espaço de tempo curto, assim o cliente tem a possibilidade de acompanhar o desenvolver de seu software, muitas vezes validando ou não cada função do sistema. Ocorrendo em um espaço de tempo curto os desenvolvedores conseguem saber realmente as necessidades do cliente fazendo com que o software evolua de forma concreta e de acordo com suas reais necessidades.

Sendo assim no feedback os desenvolvedores consegue passar para o cliente através de apresentações como está ficando a aparência do sistema, os riscos técnicos e estimativas forçando assim o cliente a lhes retornar mais algumas informações necessárias para que o sistema atinja a qualidade desejada. Com isso os requisitos que antes não muito esclarecedores para a equipe de desenvolvimento acabam se esclarecendo.

5.1.4. Coragem

A coragem em conjunto com todos os outros valores se torna algo muito proveitoso para o desenvolvimento com XP, pois não basta apenas saber o que fazer na teoria, e não colocar em prática. È necessário ter coragem para se comunicar, tendo que falar coisas boas ou ruins para alguém da equipe, para fazer o feedback com o cliente de forma realista e para ser simples, não ter medo de jogar implementações ruins fora. Pelo fato do desenvolvimento de software ser algo complicado, muitas pessoas ficam com medo e é a coragem que elas utilizam para lidar com esses medos que irá resultar em um software de qualidade ou não.

Segundo Beck (2005):

> “Às vezes a coragem manifesta-se como uma propensão a uma ação; se você souber qual é o problema, faça algo sobre ele. Às vezes a coragem manifesta-se como paciência; se você sabe que há um problema mas não sabe a solução, é preciso coragem para esperar o problema tornar-se claro “

Encorajar a equipe a trabalhar em conjunto independentemente da situação força o projeto a se desenvolver com menos riscos tornando o produto final um software de extrema qualidade. E principalmente ter coragem para seguir todas as premissas do Desenvolvimento Ágil.

5.1.5. Respeito

No desenvolvimento de software como em outras áreas o respeito é o mais importante, nenhuma pessoa é mais ou menos importante que outra, fazendo com que todos envolvidos seja um parte especial do projeto. O desrespeito não causa apenas inimizades, mas também grandes prejuízos para o desenvolvimento,

podendo assim atrasar o cronograma do projeto. Consequentemente com a falta de respeito todos o valores serão prejudicados, pois não haverá clima entre a equipe para se comunicar de formar efetiva a contribuir com o projeto, a equipe não fará o feedback de forma eficaz, não se preocuparão em fazer programas simples para o entendimento de todos os envolvidos e não terão coragem para se impor na hora em que mais é preciso.

De acordo com IMPROVE IT (2012):

> “Respeito é um valor que dá sustentação a todos os demais. Membros de uma equipe só irão se preocupar em comunicar-se melhor, por exemplo, se se importarem uns com os outros. Respeito é o mais básico de todos os valores. Se ele não existir em um projeto, não há nada que possa salvá-lo. Saber ouvir, saber compreender e respeitar o ponto de vista do outro é essencial para que um projeto de software seja bem sucedido.”

5.2. COMO FUNCIONA O XP

A equipe envolvida no desenvolvimento XP, que é composta por no máximo 10 integrantes, trabalham nas fases de desenvolvimento de forma simultânea possibilitando uma alta produtividade. Estás atividades de planejamento, análise, projeto, codificação, testes e implementação que ocorrem de forma simultânea tem a duração de uma semana, e ao final, uma funcionalidade do sistema já está concluída e devidamente testada.

5.2.1. Planejamento

A fase de planejamento é uma das fases mais importantes do processo de desenvolvimento, pois é nela que é decidido o que se fazer. Está fase acontece várias vezes durante o projeto em pequenas reuniões no inicio da semana, onde existe uma pessoa responsável por tomar decisões de negócio, está pessoa é chamada de cliente on-site, e a cada reunião mostra qual caminho o projeto deve seguir, criando cenários, desenvolvendo planos de versões, esclarecendo a visão do projeto e gerenciando os risco.

A partir dessas reuniões frequentes os programadores podem dar sugestões e fornecer estimativas essências que são analisadas e se necessário misturadas com os requisitos do cliente. Em conjunto toda a equipe se esforça para entregar pequenas funcionalidades do sistema com frequência, essas reuniões são chamadas de jogo do planejamento.

5.2.2. Análise

A análise dos requisitos é feita pelo cliente on-site, que pode ser um cliente real ou não, ele passa o tempo todo junto com a equipe e é ele que determina o que o software deve fazer fornecendo todos os requisitos necessários para os programadores. Os clientes on-site tem de ser pessoas extremamente qualificadas e que estejam totalmente inteiradas sobre quais funcionalidades o software deve fornecer.

Através de reuniões tradicionais de levantamento de requisitos, os clientes on-site devem esclarecer todos os requisitos propostos e quando algum programador precisar de informações ele deve fornecê-las.

5.2.3. Projeto

O Método de Desenvolvimento XP utiliza o projeto incremental, possibilitando uma continua melhora a partir de pequenos passos. Para que isso aconteça de forma concreta esse trabalho é guiado pelo desenvolvimento orientado a testes, onde constantemente são feitos testes no programa para verificar se tudo está correto,

normalmente os testes são feitos apenas ao final do projeto, e para que isso possa ser possível os programadores em XP trabalham em pares aumentando assim a capacidade mental para suportar cada tarefa.

5.2.4. Codificação

Trabalhando em pares os programadores são responsáveis por gerencia suas tarefas, normalmente trabalhando com um controlador de versão, para que cada par de programadores não trabalhe em versões diferentes do projeto, assim o desenvolvimento acaba se tornando automatizado garantindo a integridade do programa desenvolvido. Em pequenos espaços de tempo o programador submete seu código-fonte assim outros pares podem integrá-los de forma segura mantendo todos os trabalhos concentrados na mesma versão para que futuramente não haja problemas de integração de código. Para isso toda a equipe mantem um padrão de código e se algum erro for encontrado o programador tem de corrigi-lo não importando quem o fez.

5.2.5. Testes

Os testes encontrados em XP ocorrem em um espaço de tempo muito pequeno, assim os programadores, clientes e testadores envolvidos no projeto estão sempre testando as funcionalidades para assim garantir de fato a integridade do sistema.

Os programadores são responsáveis pelos primeiros testes, assim que acabam de fazer alguma funcionalidade eles entram fazendo testes que garante que o software faz exatamente o que eles tinham intenção de programar. Após feito os testes pelos programadores os clientes entram fazendo testes operacionais para garantir que foi implementado exatamente o que ele queriam, validando interfaces, funcionalidades do sistema e a regra de negócio aplicada. E por fim os testadores entram fazendo testes exploratórios para de fato garantir se a equipe desenvolveu códigos de qualidade . Estes testes exploratórios procuram falhas e lacunas não identificadas pelos programadores, e se algum bug é encontrado a equipe conduz seus esforços

para o causa-raiz considerando como melhorar o processo e corrigindo outras funcionalidades parecidas para que não ocorra o bug novamente.

5.2.6. Implementação

Ao final de cada iteração a equipe XP já tem alguma funcionalidade nova já produzida, assim a implementação desta funcionalidade é feita para o cliente podendo começar a utilização do software.

5.3. ÁREAS DE APLICAÇÃO

O XP foi criado para o desenvolvimento de projetos pequenos e médios com no máximo dez integrantes na equipe, podendo ser aplicado para qualquer tipo de de desenvolvimento de software. Por ser uma metodologia que visa prazos curtos com qualidade é uma ótima alternativa para o mercado de software atual, pois cada vez mais a procura por software está aumentando, porém não se pode deixar a qualidade de lado.

5.4. OBSTÁCULOS NO EXTREME PROGRAMMING

Por ter um limite de poucas pessoas envolvidas em seu desenvolvimento o XP fica limitado a projetos pequenos e médios o que não é muito atrativo para grandes empresas que desenvolves grandes softwares. E para se ter êxito na utilização do método é necessário que todas as pessoas envolvidas estejam completamente comprometidas e que saibam todos os valores propostos pelo XP, se não pelo contrário as pessoas acabaram se distanciando cada vez mais levando o projeto ao fracasso.

5.5. SCRUM

Scrum é uma metodologia de Desenvolvimento Ágil e Gerenciamento de Projetos utilizada pela primeira vez por Jeff Sutherland na empresa Easel Corporation em 1993, inicialmente era usado para a produção de automóveis e outros produtos, em

seguida foi aprimorado por Ken Schwaber sendo utilizado no processo de fabricação de softwares.

A palavra Scrum foi utilizada pois no jogo de rugby um scrum é quando uma equipe se une em um grupo de oito jogadores, quando ocorre alguma falta, para todos juntos tentarem ficar com a posse de bola, no entanto se todos os jogadores envolvidos no scrum não trabalharem de forma extremamente unidade e eficaz eles não conseguirão a posse da bola, favorecendo o outro time.

O principal objetivo do Scrum é aumentar a produtividade em um cenário muito imprevisível, onde não se tem totalmente definido todos os requisitos do sistema. Porém a metodologia Scrum não diz exatamente o que se deve fazer em cada etapa do desenvolvimento, mas propõem um conjunto de regras que devem ser seguidas para que ao final o projeto tenha sucesso.

Segundo Zanatta (ZANATTA, 2005, apud. SCHWABER, 2002, p. 2), o Scrum busca definir um processo para projeto e desenvolvimento de software orientado a objeto, que seja focado nas pessoas e que seja indicado para ambientes em que os requisitos surgem e mudam rapidamente.

De acordo com Ferreira (2005), a metodologia Scrum tem seis principais características:

- é um processo ágil para gerenciar e controlar o desenvolvimento de projetos.
- é uma base para outras práticas de engenharia de software.
- é um processo que controla o caos resultante de necessidades e interesses conflitantes.
- é uma forma de aumentar comunicação e maximizar a cooperação.
- é uma forma de detectar e remover qualquer impedimento que atrapalhe o desenvolvimento e um produto.
- é escalável desde projetos pequenos até grandes projetos em toda empresa.

Um projeto de software que utiliza a metologia Scrum inicia a partir do levantamento

de requisitos junto ao cliente, no entanto estes requisitos não são todas as funcionalidades que o sistema deve possuir quando pronto, e sim uma base para começar o desenvolvimento. Ao longo de todo o projeto irão surgindo novos requisitos para serem implementados ou requisitos antigos de devam ser mudados. Todos os requisitos levantados para o desenvolvimento do sistema são chamados de product backlog e são listados em ordem, onde o primeiro é o requisito mais importante e de mais urgência para a empresa que necessita do software e o ultimo o menos importante.

Scrum é dividido em três papeis a saber: Product Owner, Scrum Master e Scrum Team que são necessários para o desenvolvimento do projeto.

5.5.1. Product Owner

Product Owner é a pessoa responsável pelo projeto, podendo ser o cliente ou até mesmo um pessoa que conheça muito bem as necessidades do cliente, onde fica responsável pelo levantamento de todos os requisitos e o gerenciamento destes requisitos, é quem controla o product backlog estabelecendo a ordem de importância para a implementação com a visão geral de negócio.

É também responsável pela aprovação ou reprovação do produto ao final de cada sprint e deve sempre revisar o product backlog para se certificar que o projeto está seguindo o caminho certo podendo modificar ou não os requisitos anteriores.

5.5.2. Scrum Master

Scrum Master é o gerente do projeto, em um cenário onde as equipes não possuem líderes, o Scrum Master é responsável por assegurar que a equipe está seguindo no caminho correto utilizando todas as práticas do Scrum de forma correta, é também o responsável por resolver os problemas que surgem ao longo do desenvolvimento do projeto de maneira que não atrase a entrega de cada sprint.

Segundo Ramos (2011), o Scrum Master corresponde ao membro encarregado de realizar as atividades do projeto. Suas principais atividades são organizar e gerenciar suas próprias atividades e geralmente são dedicados integralmente ao

projeto.

5.5.3. Scrum Team

Scrum Team é a equipe envolvida com o desenvolvimento do software, onde existem profissionais de diferentes áreas. Está equipe é composta por 5 a 8 pessoas que trabalham em conjunto para ao fim de cada sprint conseguirem entregar uma funcionalidade do sistema pronto e testado para o cliente.

5.5.4. Sprint

Sprint é a fase mais importante do desenvolvimento utilizando a metodologia Scrum, mostrado na figura(12), é nele que as coisas de fato acontecem. A partir do product backlog estipulado pelo product owner, onde todos os requisitos estão organizados em ordem crescente do mais importante ao menos importante, é que o scrum team começa o desenvolvimento do projeto.

Antes de começar cada sprint o scrum team separa a primeira tarefa do product backlog em tarefas que devem ser realizadas durante o sprint formando o sprint backlog, e é a partir destas tarefas que a equipe começa a desenvolver a funcionalidade do sistema.

Cada sprint do projeto corresponde a um requisito do cliente, uma funcionalidade que o sistema deve conter, a partir desse requisito o scrum team tem um mês para desenvolver e testar está funcionalidade para colocar está funcionalidade em funcionamento junto ao cliente. Assim a cada término de um sprint o cliente pode estar em contato com o sistema que está sendo desenvolvido para ele podendo aceitar ou rejeitar o módulo entregue fazendo com que a dinâmica do desenvolvimento seja muito intensa.

A partir deste cenário de grande contato com o cliente os requisitos inicias são mudados com grande frequência durante o desenvolvimento do projeto fazendo com que o product owner esteja sempre atualizando o product backlog. Com o grande contato com o cliente e as frequentes mudanças nos requisitos o produto final acaba se tornando exatamente o que o cliente necessita.

Durante cada sprint as pessoas envolvidas com o projeto fazem pequenas reuniões para que discutam o desenvolvimento do projeto, o que já está feito e oque falta fazer, e se algum impasse surgir o scrum master é responsável por resolvê-lo para que o projeto continue de forma incremental.

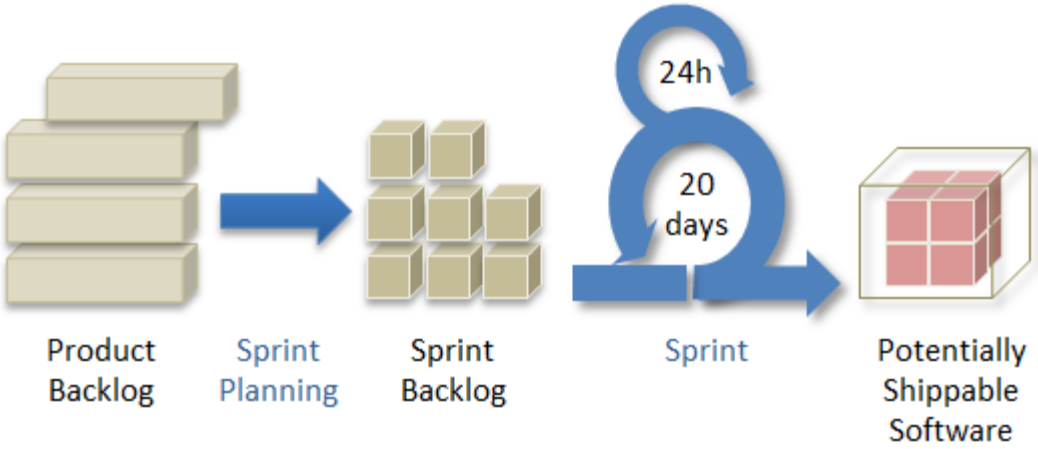

Figura 12 – *Agile Scrum Sprint Process Flow* (WALKER, 2010).

5.6. APLIAÇÕES DO SCRUM

O Scrum está sendo muito utilizado no mundo, não apenas para o desenvolvimento de software como também para o gerenciamento de diversos tipos de projetos. Por ser uma metodologia de Desenvolvimento Ágil muito simples e que possibilita a grande interação com o cliente ela propicia a entrega de um produto com grande qualidade em um curto espaço de tempo e que ao longo do desenvolvimento possa ocorrer diversas mudanças nos requisitos fazendo com que o sistema seja desenvolvido de forma incremental.

6. UMA NOVA ABORDAGEM

Ao longo dos anos as tecnologias de desenvolvimento evoluíram muito rapidamente, forçando o mercado a se adaptar para continuar oferecendo softwares de qualidade. No entanto, a Engenharia de Software evoluiu mais lentamente, fazendo com que softwares com alta tecnologia fossem desenvolvidos utilizando métodos de construção mais antigos.

A proposta deste trabalho é apresentar uma nova abordagem para a engenharia de software para assim possibilitar um desenvolvimento que:

- Possa evoluir de acordo com as necessidades da empresa.
- Possa ser utilizado em projetos grandes e complexos sem perder a qualidade final do produto.
- Tenha o código fonte totalmente organizado para facilitar a manutenção.
- Seja mais rápido, tornando o custo do projeto mais baixo.
- Seja mais flexível, podendo ocorrer mudanças inesperadas sem prejudicar o andamento do projeto.

A Engenharia de Software Tradicional prega que a documentação do projeto seja algo muito importante, e de fato a documentação é muito importante, pois um software desenvolvido não fica pronto ao término do projeto, e sim evolui de acordo com as necessidades da empresa ao longo dos anos. Assim, com o passar do tempo a empresa que comprou o software pode precisar de alguma mudança, uma melhoria, uma correção de erro ou uma nova funcionalidade.

Essas necessidades por parte da empresa podem surgir uma semana após o término do projeto ou até mesmo depois de anos, e a equipe de desenvolvimento tem que estar preparada para suprir essas necessidades. Com o passar dos anos funcionários entram e saem das empresas, fazendo com que a equipe que desenvolveu o sistema não seja mais a mesma.

Neste ponto a documentação do sistema se torna algo muito importante para a continuidade do sistema, pois é a partir dela que novas equipes podem entender o sistema e conseguir fazer as melhorias de forma correta sem afetar nenhuma outra parte do sistema.

Nesta nova abordagem a documentação é muito importante para o sistema, mas ela não será feita no início do projeto, pois pode haver muitas mudanças fazendo com que ao longo do desenvolvimento tenha que modificá-la muitas vezes causando um grande atraso no projeto. Ao final, quando o cliente já aprovou o sistema, a documentação é feita de forma que no futuro, quando necessário, ela seja esclarecedora. E assim, como o sistema, a documentação tem que evoluir, quando feita alguma mudança no projeto, e esta mudança for aprovada, é preciso acrescentar o recurso desenvolvido.

Hoje em dia o uso de sistemas é muito grande, pois nas empresas vários setores estão sendo automatizados com sistemas, e com a crescente demanda por sistemas o tamanho e a complexidade aumentam muito fazendo com que o desenvolvimento acabe se tornando algo muito difícil.

Para facilitar o desenvolvimento a Engenharia de Software Orientada a Agentes propõem dividir os grandes problemas a serem resolvidos pelo sistema em problemas menores. Quando todos os problemas menores forem resolvidos consequentemente o problema maior estará resolvido.

A resolução destes problemas se dá através de uma entidade autônoma e flexível chamada agente. Assim a partir do levantamento de requisitos iniciais são criados agentes onde cada um é responsável por resolver um problema. E através de sensores receptores esse agente capta as informações do ambiente para resolver um problema.

Quando um sistema é muito complexo e lida com diversos tipos de tarefas, o sistema não pode apenas esperar que o usuário faça algo em determinadas situações, por exemplo em um hospital, se o paciente está precisando que um determinado soro seja injetado em sua veia se o batimento cardíaco cair, ele não pode ficar esperando algum enfermeiro ver que os batimento estão caindo, assim o sistema capta as informações do batimento analisa e injeta o soro se necessário.

A abordagem de agentes é muito utilizada para sistemas que possuem inteligência artificial, principalmente em jogos, e como o desenvolvimento de softwares e aplicativos inteligentes estão cada vez mais presentes no mercado é necessário que o desenvolvimento seja rápido e que resolva todos os problemas impostos.

Assim para facilitar o desenvolvimento na abordagem proposta, o uso de agentes será essencial. Após o levantamento dos requisitos o engenheiro de software divide todas as principais funcionalidades em agentes, fazendo com que um ou mais agentes sejam responsáveis por um requisito do sistema formando um sistema multiagente.

A maioria dos sistemas desenvolvidos na atualidade estão em constante evolução, sempre é preciso fazer, correção de erros e até mesmo adicionar novas funcionalidades, no entanto para se dar uma manutenção de forma rápida e efetiva o código do sistema tem que estar totalmente organizado. Estas manutenções podem surgir muitos anos depois da entrega do sistema, e mesmo assim elas tem de ser cumpridas rapidamente.

A Engenharia de Software Orientada a Aspecto propõem dividir os requisitos em funcionais ou não funcionais, onde os funcionais descrevem uma tarefa do sistema e os não funcionais uma funcionalidade do sistema. Esses requisitos não funcionais são chamados de aspectos, que podem ser funcionalidades de log, persistência, segurança e auditoria que normalmente são usadas para todas as classes do sistema. Quando programamos essas classes sem utilizar o conceito de orientação a aspectos, temos que fazer cada um deles para cada uma das classes, fazendo com que fique um código muito extenso e repetitivo.

Futuramente se necessário alguma mudança em uma dessas funcionalidades é necessário que mude todas as partes que utilizam log, persistência, segurança e auditoria, tornando um trabalho repetitivo e demorado.

Se tratarmos estas funcionalidades como um aspecto precisaremos escrevê-las apenas uma vez, tornando o desenvolvimento mais rápido e consequentemente facilitando as manutenções futuras, pois ao modificar uma funcionalidade automaticamente estará modificando para todas as outras que a utilizam.

Para tornar a programação dos agentes mais eficiente, o engenheiro de software deverá identificar as funcionalidades que serão mais utilizadas por todos os agentes e assim separá-las em aspectos.

Para o desenvolvimento de todo o projeto é interessante o uso do Desenvolvimento Ágil que facilita a entrega em um espaço de tempo menor, um software constantemente testado e que possa sofrer mudanças bruscas ao longo do projeto sem comprometer o cronograma. Para todo o desenvolvimento ocorrer de forma concreta é preciso aplicar todos os doze princípios do Desenvolvimento Ágil.

Dentro do Desenvolvimento Ágil existem dois métodos, Extreme Programming e Scrum, que são muito bons para o desenvolvimento de projetos de forma rápida e consistente.

O Extreme Programming contém alguns valores, que são comunicação, simplicidade, feedback, coragem e respeito, que são extremamente importantes que sejam seguidos pela equipe de desenvolvimento, pois é a partir deles que se consegue ter um bom ambiente de trabalho facilitando a conclusão do projeto no tempo estipulado.

A equipe em Extreme Programming tem no máximo 10 pessoas, no entanto para esta nova abordagem usaremos uma ou mais equipes de no máximo 10 pessoas, dependendo do tamanho do projeto. Assim cada equipe pode ficar responsável pelo desenvolvimento de um agente dentro do sistema e uma equipe desenvolverá os aspectos que serão necessários para todos os agentes do sistema. Dentro de cada equipe a programação em par será necessária quando a complexidade de cada agente for muito grande, facilitando assim a resolução do problema.

Para que tudo isso de fato de certo e corra em seu tempo normal, é necessário a utilização do Scrum, que é de fato a gerência do projeto. O Scrum pode ser aplicado para qualquer desenvolvimento de algum produto, não necessariamente o desenvolvimento de software.

Na nova abordagem o Product Backlog será essencial para o levantamento de requisitos organizando-os em ordem crescente do mais importante ao menos importante. Esses requisitos são as funcionalidades, sendo assim cada funcionalidade é um agente do sistema. A primeira funcionalidade a ser desenvolvida serão os aspectos necessários para o sistema, elaborando a arquitetura do sistema.

Se o sistema não for muito grande e for necessário apenas uma equipe, usaremos apenas um sprint de cada vez, onde a cada sprint um agente deverá ficar pronto. No entanto se for um sistema muito grande e complexo ocorrerão vários sprints ao mesmo tempo, assim cada equipe do Extreme Programming fica responsável por um sprint.

Para gerenciar todas essas equipes e o desenvolvimento do sistema será utilizado a figura do Scrum Master, o gerente do projeto, onde ficará responsável por resolver os problemas e estará sempre em contato com o cliente para que o sistema evolua de acordo com o que o cliente precisa.

Com a mescla das melhores características de cada paradigma de engenharia de software temos as seguintes fases do desenvolvimento(Figura 13):

- Levantamento de requisitos, produção do Product Backlog.
- Levantamento dos aspectos necessários para o sistema.
- Divisão das funcionalidade do Product Backlog em agentes.
- Programação dos aspectos do sistema dentro de um sprint.

- Programação dos agentes do sistema dentro de cada sprint.

- A cada sprint para o desenvolvimento dos agentes, entregar uma funcionalidade do sistema para o cliente.

- A cada sprint testar constantemente o sistema.

- Montar a documentação do sistema.

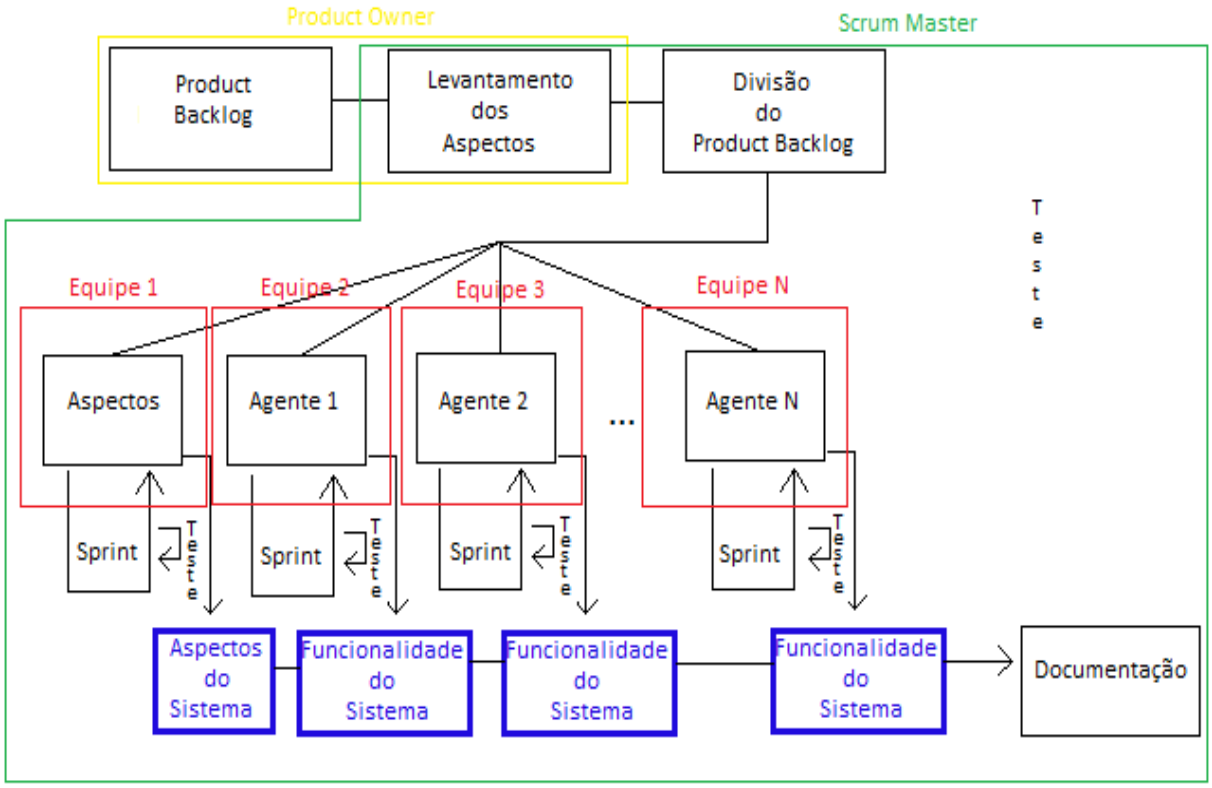

Figura 13 – Fases do Desenvolvimento utilizando a nova abordagem.

Está nova abordagem de engenharia de software pode ser utilizada tanto para sistemas grandes como para sistemas pequenos, o tamanho do sistema irá definir a quantidade de equipes necessárias. Define-se que cada equipe possuirá, no máximo, dez pessoas para atuar no desenvolvimento.

7. CONCLUSÃO

A Engenharia de Software vem sofrendo mudanças ao longo dos anos para cada vez mais melhorar o processo de desenvolvimento de um software, de forma rápida e com baixo custo. O paradigmas de engenharia de software surgem com a necessidade de melhorar este processo de desenvolvimento, assim não podemos dizer que um paradigma seja melhor que o outro e sim que determinado paradigma é melhor para determinada situação.

Ao usar está nova abordagem, o engenheiro de software terá todas as melhores características de cada um dos paradigmas apresentados trabalhando em conjunto para ao final apresentar um software de qualidade, terminado no prazo correto e com facilidades para implementar novas funcionalidades.

No entanto está abordagem ainda não foi testada em um ambiente real de desenvolvimento, podendo assim apresentar algumas falhas que poderão ser corrigidas em trabalhos futuros e com o amadurecimento da ideia.

Com este trabalho foi possível concluir que a Engenharia de Software não existe um padrão a ser seguido, assim o engenheiro pode mesclar diversas técnicas de diversos paradigmas para assim formar uma nova abordagem que se encaixe melhor para determinado software em desenvolvimento.

Está nova abordagem abre novos caminhos para o desenvolvimento de trabalhos futuros, onde poderá ser testada em ambientes reais de desenvolvimento, podendo concluir se está abordagem se encaixa melhor para softwares pequenos, médios ou de grande porte. Ou até mesmo ser mesclada com outros paradigmas para obter um novo paradigma que seja eficiente para determinado tipo de desenvolvimento.

REFERÊNCIAS

BISSI, Wilson. **Scrum - Metodologia De Desenvolvimento Ágil**, 4p. Centro Universitário de Maringá (CESUMAR), Tecnologia em Análise e Desenvolvimento de Sistemas, Maringá, PR, 2007.

BRITO, I. ; MOREIRA, A**.. Advanced Separation Of Concerns For Requirements Engineering.Viii**. Jornadas de Ingeniería de Software y Bases de Datos (JISBD), Alicante, Spain, 12-14 Novembro 2003.

CAFARATE, Liane Santiago. **Utilização Da Engenharia De Software Orientada A Agentes Na Modelagem De Um Sistema De Seleção De Pessoas .** 73p. Universidade Federal de Santa Maria – Centro de Tecnologia, Trabalho de Graduação , Santa Maria, RS, 2008.

CARENO, MARCHI, Fernando Foltre, Késsia R. C**.. Engenharia De Software Orientada A Agentes**. 5p. Universidade Paranaense (UNIPAR) , Paranavaí – PR.

CASACHI, Rafael Alessandro**. Aplicação Do Paradigma De Programação Orientada A Aspecto No Desenvolvimento De Sistemas De Informação**. 80p. Instituto Municipal do Ensino Superior de Assis – IMESA , Trabalho de Graduação , Assis, SP, 2011.

CASTRO, ALENCAR, SILVA, Jaelson, Fernanda, Carla. **Engenharia De Softwareorientada A Agentes**. 38p.

CASTRO, Vinicius Almeida. **Desenvolvimento Ágil Com Programação Extrema Eficácia E Disciplina Extrema No Desenvolvimento Orientado A Objetos De**

Software, 122p. Universidade de Sergipe - Ciência da Computação, Trabalho de Conclusão de Curso. São Cristóvão – SE, 2006.

CHITCHYAN, Ruzanna et al**..Survey Of Aspect-Oriented Analysis And Design Approaches**. Survey Lancaster University. May, 2005.

CLARKE, Siobhán; **Baniassad, Elisa. Aspect-Oriented Analysis And Design – The Theme Approach**. United States. AddisonWesley, March, 2005.

FERREINA, Décio, **SCRUM Um Modelo Ágil para Gestão de Projetos de Software.** 11P, 2005.

FININ, LABROU,MAYFIELD, Tim, Yannis James. **KQML as an agent communication**

language. Computer Science Department. University of Maryland Baltimore County.

Baltimore, USA, 1993.

GIRARDI, Steferson Lima Rosario. **Arquiteturas De Software Baseadas Em Agentes: Do Nível Global Ao Detalhado**. 19p. Universidade Federal do Maranhão Departamento de Informática Campus do Bacanga s/n , São Luis – MA, 2002.

GIRARDI, Rosario**. Engenharia De Software Baseada Em Agentes**. 25p. Universidade Federal do Maranhão DEINF - GESEC Campus do Bacanga s/n , São Luis – MA, 2004.

HUGO, GROTT, Marcel, Marcio Carlos**. Estudo De Caso Aplicando Programação Orientada A Aspecto**. 12p.

IMPROVE IT, site sobre XP. Disponível em: <http://www.improveit.com.br/xp/valores/respeito>. Acesso em: 31 mai. 2012.

IEEE, **Standard Glossary of Software Engineering Terminology**. 84p. 1990.

JR, S. M. S. **Concerns In Requirements Model - A Small Case Study. In Proceedings Of Early Aspects 2003**: Aspect-Oriented Requirements Engineering and Architecture Design,Workshop, March 17, Boston, USA, 17 Mar. 2003.

KLUGE, Monike Roberta. **Scrum Project: Ferramenta De Apoio Ao Gerenciamento De Projetos De Software Baseada Nos Princípios Da Metodologia Ágil Scrum** 81p. Universidade do Vale do Itajaí – Centro de Ciências Tecnológicas da Terra e do Mar Curso de Ciência da Computação. Itajaí– SC, 2009.

LUIZ, Ricardo Fernandes, **Sem boas práticas de engenharia não há agilidade**

Disponível em: <http://ricardofluiz.wordpress.com/2010/10/18/sem-boas-praticas-de-engenharia-nao-ha-agilidade/>. Acesso em: 03 out. 2012.

ODELL, J., PARUNAK, H. D., and BAUER, B.. **Extending Uml For Agents.** In Proceedings of the 2nd Int. Bi-ConferenceWorkshop on Agent-Oriented Information Systems (AOIS 2000),Austin, USA, pp. 3–17, 2000.

PARAISO, E. C. **Concepção e Implementação de um Sistema Multiagente para Monitoração e Controle de Processos Industriais,** Dissertação (Mestrado em Engenharia Elétrica e Informática Industrial), CEFET - PR, 1997.

PRESSMAN, R. S. **Engenharia de Software**. São Paulo: Makron Books, 1995.

RAMOS, Matheus Fernandes. **SCRUM E XP:**

Um comparativo no processo de desenvolvimento de software

. 46p.Trabalho de Conclusão de Curso - Universidade FUMEC, Belo Horizonte, MG, 2011.

SILVA, Luciana de Paiva**. A Engenharia De Requisitos Orientada A Aspectos: A Abordagem Daore** . 249p. UEM – Maringá, PR, 2007.

SILVA, TEDESCO, CASTRO, Carla, Patrícia, Jaelson**. Comparing Agent-**

Oriented Methodologies Using a NFR Approach. 9p. Proceedings of the 3rd Software Engineering for Large-Scale Multi-Agent Systems (SELMAS'04), Edinburgh, Scotland, 2004.

SOARES, Michel dos Santos. **Comparação Entre Metodologias Ágeis E Tradicionais Para O Desenvolvimento De Software**, 6p. Faculdade de Tecnologia e Ciências de Conselheiro Lafaiete. Conselheiro Lafaiete - MG, 2004.

SHORE, WARDEN, James, Shane; **A Arte do Desenvolvimento Ágil**. Tradução de Bianca Capitânio. 8. ed. Rio de Janeiro: Alta Books, 2008.

TAHARA, OHSUGA, HONIDEN, Yasuyki, Akihiko, Shinichi. **Agent System Development**
Method Based on Agent Patterns, p. 356. The 21st International Conference on Software Engineering (ICSE'99), Los Angeles, CA, USA, 1999.

TIRELO, BIGONHA, BIGONHA, VALENTE, Fabio, Roberto da Silva, Mariza Andrade da Silva, Marco Túlio de Oliveira . **Desenvolvimento De Software Orientado Por**

Aspecto. 42p. Pontifícia Universidade Católica de Minas Gerais - Departamento de Ciência da Computação – Belo Horizonte, MG, 2004.

TEKINERDOGAN, Bedir: **Asaam: Aspect Software Architecture Analysis Method**. WICSA 2004: 5-14.

WALKER, Mark , **Adopting the Agile methodology, a personal retrospective**

Disponível em: <http://www.g3it.com/blog/technical/adopting-the-agile-methodology-a-personal-retrospective/>. Acesso em: 28 set. 2012.

WOOLDRIDGE, M. **An Introduction To Multiagent Systems**. John Wiley and Sons, Ltd. 103p. 2002.

ZANATTA, Alexandre Lazaretti. **Uma análise do método ágil Scrum conforme abordagem nas áreas de processo Gerenciamento e Desenvolvimento de Requisitos do CMMI**. 12p. Universidade de Passo Fundo - UPF, Passo Fundo, RS, 2004.