UNIVERSIDADE FEDERAL DO PIAUÍ
CAMPUS SENADOR HELVÍDIO NUNES DE BARROS
CURSO DE BACHARELADO EM SISTEMAS DE INFORMAÇÃO

THAÍS KARINE SILVA SANTOS

APLICAÇÃO DO MODELO ÁGIL *SCRUM* COM EQUIPES DISTRIBUÍDAS NO DESENVOLVIMENTO DE JOGOS EDUCACIONAIS

PICOS – PI
2014

THAÍS KARINE SILVA SANTOS

APLICAÇÃO DO MODELO ÁGIL *SCRUM* COM EQUIPES DISTRIBUÍDAS NO DESENVOLVIMENTO DE JOGOS EDUCACIONAIS

Trabalho de Conclusão de Curso apresentado ao Curso de Bacharelado em Sistemas de Informação do Campus Senador Helvídio Nunes de Barros da Universidade Federal do Piauí – UFPI, como requisito final para obtenção do título de Bacharel em Sistemas de Informação.

ORIENTADOR: PROF. ESP. DENNIS SÁVIO MARTINS DA SILVA.

PICOS - PI

2014

THAÍS KARINE SILVA SANTOS

APLICAÇÃO DO MODELO ÁGIL *SCRUM* COM EQUIPES DIDTRIBUÍDAS NO DESENVOLVIMENTO DE JOGOS EDUCACIONAIS

Trabalho de Conclusão de Curso apresentado ao Curso de Bacharelado em Sistemas de Informação do Campus Senador Helvídio Nunes de Barros da Universidade Federal do Piauí – UFPI, como requisito final para obtenção do título de Bacharel em Sistemas de Informação.

Data de Aprovação: 04 de Agosto de 2014

Dennis Sávio Martins da Silva, Esp. (Orientador)

Patrícia Medyna Lauritzen de Lucena Drumond, MSc (Membro)

Alcilene Dalília de Sousa, MSc (Membro)

Eu, **Thaís Karine Silva Santos,** abaixo identificado(a) como autor(a), autorizo a biblioteca da Universidade Federal do Piauí a divulgar, gratuitamente, sem ressarcimento de direitos autorais, o texto integral da publicação abaixo discriminada, de minha autoria, em seu site, em formato PDF, para fins de leitura e/ou impressão, a partir da data de hoje.

Picos-PI, 08 de agosto de 2014.

Thaís Karine Silva Santos

Assinatura

FICHA CATALOGRÁFICA
Serviço de Processamento Técnico da Universidade Federal do Piauí
Biblioteca José Albano de Macêdo

S237a Santos, Thaís Karine Silva.
Aplicação do modelo ágil *scrum* com equipes distribuidas no desenvolvimento de jogos educacionais / Thaís Karine Silva Santos. - 2014.
CD-ROM : il. ; 4 ¾ pol. (60 p.)

Monografia(Bacharelado em Sistemas de Informação) – Universidade Federal do Piauí. Picos-PI, 2014.
Orientador(A): Prof. Esp. Dennis Sávio Martins da Silva

1. Jogos Educacionais. 2. Desenvolvimento de Software. 3. Scrum. I. Título.

CDD 005.1

Dedico este trabalho à minha família. Meus pais, meu marido, meu avô Valdemar Gonzaga Santos (*In Memoriam*), pela forma carinhosa que sempre me tratou e por não estar aqui para ver essa minha conquista que tanto esperava, e em especial quero agradecer aos meus filhos, Yasmim e Otávio, por existirem e por ser a minha fonte de coragem, determinação felicidade, meus filhos todas as minhas conquistas são motivadas por vocês.

AGRADECIMENTOS

À Deus primeiramente, por estar sempre presente na minha vida, me fortalecendo a cada amanhecer.

À Mãe Rainha, pela proteção maternal divina.

À minha Mãe Edilza, por ser meu porto seguro, por seu exemplo de grande mulher, pelas palavras de otimismo durante toda a minha jornada.

Ao meu Pai Manoel, por ter traduzido o orgulho de ser meu pai com seus gestos e atitudes para comigo.

Ao meu marido Edimilsom, por estar sempre ao meu lado, me apoiando, cuidando, durante toda essa fase da minha vida.

A minha irmã Thaila, e minha sogra Dacruz pela ajuda nos momentos que mais precisei.

Aos meus familiares que direto ou indiretamente contribuíram para essa conquista

Aos meus colegas de turma, professores, em especial o MSc. Ryan Ribeiro de Azevedo e meu orientador Dennis Sávio Martins da Silva, pela paciência, dedicação e auxilio.

“Nunca deixe que digam que não vale apena acreditar num sonho que se tem, ou que seus planos nunca vão dar certo, ou que você nunca vai ser alguém! Tem gente que machuca os outros tem gente que não sabe amar, mas eu sei que um dia a gente aprende, se você quiser alguém em quem confiar, confie em si mesmo. Quem acredita sempre alcança!”

(Renato Russo)

RESUMO

O desenvolvimento de *softwares* requer abordagens inovadoras, e atualmente vêm se destacando entre elas, o desenvolvimento distribuído de *software*, que apresenta diversas vantagens e desafios a serem vencidos. Nesse contexto notam-se as metodologias ágeis capazes de apresentar soluções de adaptações nesse ambiente distribuído de *software*. Dessa forma, este trabalho pretende descrever a experiência de aplicar o *scrum*, um modelo ágil de desenvolvimento de *software*, em um projeto de implementação de jogos educacionais com duas equipes distribuídas. A implementação desses jogos é apresentado como estudo de caso, de modo a ilustrar e validar a aplicação do modelo *scrum* com o desenvolvimento distribuído de *software*.

Palavras-chave: Jogos Educacionais. Desenvolvimento de *Software*. *Scrum*. Desenvolvimento Distribuído de *Software*.

ABSTRACT

Software development requires innovative approaches, and currently are distinguishing between them, distributed development of software, it has many advantages and challenges to be overcome. In this context notice is agile methodologies able to provide solutions that adaptations of distributed software environment. Thus, this study aims to describe the experience of applying Scrum, an agile software development model, in a implement project of educational games with a distributed team. The implement of this game will be presented as case study to illustrate and validate the model application scrum with the distributed development of software.

Keywords: Educational games. Software Development. Scrum. Distributed Development of Software.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

DAS	Desenvolvimento Adaptativo de Software
DDS	Desenvolvimento Distribuído de Software
DS	Desenvolvimento de Software
ES	Engenharia de Software
IDE	*Integreted Development Envoronment*
XP	*Extreme Programming*

SUMÁRIO

1. INTRODUÇÃO

Atualmente o ato de ensinar não se resume à transmissão de conteúdos. A sociedade passa por transformações, e estas podem influenciar no modelo antigo de educação (onde o aluno tornava-se o agente passivo e o professor o detentor do saber) a ser mais dinâmico, que leve desafio aos alunos, considerando os conhecimentos previamente adquiridos por eles, visto que, apenas absorver o conteúdo não é relevante, pois os alunos precisam dominar o processo de aprendizagem para desenvolver suas competências (LIMA; SILVA; SILVA, 2009). Nas novas técnicas pedagógicas o professor conduz e faz a mediação do processo pedagógico levando os desafios que estimulem o aprendizado do aluno.

Neste contexto, o jogo torna-se uma ótima ferramenta para o professor no processo de aprendizagem, por estimular o interesse do aluno, possibilitando a construção do conhecimento, pois possibilitam ao aluno aprender de forma natural, prazerosa e dinâmica. Sendo assim, o jogo, por proporcionar maior interação sociocultural entre os indivíduos, torna-se um grande aliado no processo de desenvolvimento educacional (LIMA; SILVA; SILVA, 2009), e uma técnica pedagógica atual de ensino. E isso se deve ao avanço da tecnologia que esta transformando a maneira de ensinar e aprender, por isso, o ritmo do mundo globalizado e a complexidade crescente de tarefas que envolvem informações e tecnologia fazem com que o processo educativo não possa ser considerado uma atividade primordial. Dessa forma, vemos que a demanda educativa deixou de ser exclusivamente de uma faixa etária que frequenta as escolas e passou a ser uma necessidade das pessoas em geral que necessitam estar continuamente atualizados para o competitivo mundo do trabalho.

A partir do avanço da tecnologia na sociedade, e a consequente exigência em oferecer produtos de *softwares* de qualidade, surge à necessidade de expansão da produção, onde os produtores (empresas ou instituições de ensino) desenvolvam os projetos de *software* de forma rápida e organizada. Com isso as organizações estão cada vez mais motivadas a aderir ao modo de desenvolver *software* de forma distribuída, e como consequência direta desse novo modo de desenvolvimento de *software* tem a diminuição de custos e uma maior agilidade e praticidade na hora de encontrar mão-de-obra (ALBUQUERQUE *et al.*, 2007).

O avanço das comunicações através da Internet e o crescente surgimento de ferramentas voltadas para a Engenharia de software, as quais visam organizar de forma mais eficiente tem influência relevante no desenvolvimento distribuído. Tais ferramentas (controladores de versões, ferramentas de triagem de bugs, fóruns de discussão, listas de e-mail, etc), são oferecidas gratuitamente por alguns *sites* como o *SourceForge.net* (ALBUQUERQUE *et al.*, 2007).

Os engenheiros de *software* têm reconhecido a grande influência dessa nova forma de trabalho no seu dia-a-dia e estão em busca de modelos que facilitem o desenvolvimento de *software* com equipes geograficamente distantes (AUDY; PRIKLDNICKI, 2008). E agregado a esse contexto, as metodologias ágeis determinam uma solução para o gerenciamento do desenvolvimento distribuído de *software*, devido a um ritmo acelerado de mudanças e inovações tecnológicas. Como exemplo tem-se o *Scrum*, que viabiliza um conjunto de práticas com foco na transparência do desenvolvimento através da comunicação continua flexibilidade, capacidade de oferecer produtos e serviços de valor ao mercado em curtos períodos de tempo.

Com essa base, este trabalho apresenta a aplicação de dois jogos educativos utilizando os conceitos da metodologia ágil *Scrum* com equipes distribuídas.

A importância da utilização de jogos como instrumentos educativos é fundamental, segundo (FREIRE, 2005, p.82-83)"O jogo ajuda a não deixar esquecer o que foi aprendido[...] faz a manutenção do que foi aprendido[...] aperfeiçoa o que foi aprendido[...] vai fazer com que o jogador se prepare para novos desafios...". Sendo assim, o jogador aprende brincando e de forma prazerosa, além de estimular na socialização, possibilitando o exercício dos conceitos sociais, culturais, éticos, econômicos, políticos, de solidariedade, de regras, de trabalho em grupo e de respeito mútuo (LIMA *et al.*, 2008, 2009).

Com todas essas características, o processo de desenvolvimento de jogos e *softwares* em geral, não é tarefa fácil, para tanto, é indispensável a utilização de metodologias de desenvolvimento que apoiem todo o processo.

Com esses atributos, e considerando o mercado de jogos cada vez mais competitivo, a agilidade na entrega do produto final se torna indispensável, quando a qualidade do produto é garantida. Sendo o desenvolvimento distribuído de *software* um processo que influência positivamente no tempo de conclusão de desenvolvimento de um *software*, a sua aplicação juntamente com a utilização de

metodologias ágeis para esses processos de desenvolvimento é uma solução que chama a atenção para uma serie de modelos, entre eles o *Scrum*, um Modelo Ágil que se torna adaptável aos mais diferentes ambientes, e quando associado ao desenvolvimento distribuído, torna-se desafiador.

Neste trabalho foi feito uma descrição da aplicação da Metodologia Ágil *Scrum* no desenvolvimento de dois jogos educacionais, por duas equipes geograficamente distribuídas, com a finalidade de colocar em prática os métodos desse modelo de desenvolvimento através da descrição das funcionalidades e características aplicadas no processo de desenvolvimento dos jogos, com a finalidade de acrescentar conhecimentos que fortalecem ainda mais os trabalhos na área. Para a concepção dos jogos, foi utilizada a linguagem de programação *Java* na IDE (*Integreted Development Envoronment*) *NetBeans* na plataforma *Windows*. Com o objetivo de descrever a realização de uma aplicação do desenvolvimento de um jogo educacional utilizando o Modelo Ágil *Scrum* com times distribuídos.

Este trabalho é composto por 5 Capítulos, organizados da seguinte forma. O Capítulo 2 traz o embasamento teórico da aplicação onde é abordado sobre os Jogos Educativos mostrando a importância de sua utilização no processo educacional. A seguir é apresentado os conceitos da Engenharia de Software e seus Processos de Desenvolvimento, os Tradicionais e Ágeis, onde é mostrado seus principais Modelos de Desenvolvimento de Software. Em seguida, o Desenvolvimento Distribuído de Software é apresentado, mostrando os fatores motivacionais para o uso dessa abordagem. E por fim, é feito uma comparação dos processos de Desenvolvimento Tradicional e Ágil de software.

O Capítulo 3 mostra como a aplicação foi desenvolvida, seu funcionamento, suas fases, algumas imagens do processo de desenvolvimento e da interface gráfica do jogo. Em seguida são apresentados os desafios obtidos durante o projeto e as soluções encontradas.

No Capítulo 4 é apresentado os resultados obtidos durante o processo de desenvolvimento. E no Capítulo 5, é apresentado a conclusão do trabalho, onde são descritos as considerações finais que influenciaram no desenvolvimento do projeto.

2. REFERENCIAL TEÓRICO

Nas próximas seções serão apresentados uma revisão bibliográfica sobre os jogos educacionais, a engenharia de *software* e seus modelos de processos de desenvolvimento de *software* e o desenvolvimento distribuído de *software*.

2.1 Jogos Educacionais

Sabemos que os jogos surgiram em ambientes não computadorizados, porém atualmente, podemos encontra-los em variados dispositivos móveis. Sua utilização no processo educacional tem sido de grande importância pelo fato de ser uma ferramenta que afeta a motivação, as funções cognitivas e a curiosidade do aprendiz, pois estes jogos permitem a experimentação e a exploração pelo jogador (BICUDO *et al.*, 2007). Além disso, obtém-se o aprofundamento dos conhecimentos referentes a áreas de conhecimento em que o jogador tem pouco interesse.

Os jogos educativos tem se destacado como uma importante ferramenta de aprendizagem para a sociedade em geral. Segundo Cunha *et al.*(2009, p.02):

> *"Por um longo tempo os jogos foram associados a atividades de entretenimento, limitados apenas à recreação, foram inicialmente utilizados no contexto educativo a partir do rompimento com o paradigma tradicional e o surgimento do construtivismo, que ressalta a participação e experimentação do sujeito na construção de seu próprio conhecimento através de suas interações".*

A ideia de utilizar os jogos como atividade lúdica estimula o interesse do jogador, visto que no momento de descontração a pessoa vai se envolvendo tornando a brincadeira uma forma de socialização que aprimora o conviver e consequentemente o aprender. Filho; Gomes(2008, p.6) ressalta:

> *"A importância de usar os jogos como instrumento educativo para superar os preconceitos e vícios adquiridos no dia a dia, e para acentuar as vivências positivas é porque através dele ultrapassamos os limites da nossa imaginação, superamos limitações pessoais ou sociais, e descobrimos caminhos diferentes para nos tornarmos uma pessoa melhor."*

Além disso, os jogos possibilitam o aprimoramento do individuo na capacidade de resolução de problemas, na tomada de decisões, na formação de uma postura ativa.

Um dos grandes desafios dos jogos educativos é apresentar para o jogador

um conteúdo educacional onde coleção de dados é vista de uma forma interessante. (BICUDO *et al.*, 2007) ressalta que esse problema referente à temática educativa acontece na sua elaboração, pois na maioria das vezes não tem o mesmo dinamismo, interatividade, estímulo e desafio de um jogo convencional. Portanto, para que jogo educacional seja útil, ao elabora-lo é necessário promover situações interessantes com desafios na resolução de problemas, fazendo com que os jogadores participem ativamente de todas as jogadas (ARAÚJO *et al.*, 2007). Com base nesses aspectos Cunha *et al.* (2009), sugerem um roteiro que deve ser observado durante o desenvolvimento de jogos educativos:

- Aspectos educacionais devem se apresentar no contexto, na ambientação ou no conhecimento prévio do usuário;
- O ambiente do jogo deve conter roteiros ricos e interativos, elaborados de forma a promover o jogo;
- Adaptar as características pedagógicas ao roteiro de forma a enfatizar o lúdico.

Dessa forma, se construídos com essas características podem apresentar pontos positivos para o processo de ensino-aprendizagem.

2.2 Engenharia de *Software* (ES)

Nos anos 60 houve uma crescente demanda de novos e modernos computadores, o *software* ganhou visibilidade, e por conta disso, iniciou uma serie de problemas relacionados ao “amadorismo” para seu desenvolvimento, surgindo assim à Crise do *Software* que emergiu no final dos anos 60. Para (NETO, 2007, p. 9):

> *“O termo expressava as dificuldades do desenvolvimento de software frente ao rápido crescimento da demanda por software, da complexidade dos problemas a serem resolvidos e da inexistência de técnicas estabelecidas para o desenvolvimento de sistemas que funcionassem adequadamente ou pudessem ser validados.”*

A Crise do *Software* refletia problemas no desenvolvimento, tais como o estouro de prazo e orçamento, os requisitos não eram atendidos, tornando o *software* de baixa qualidade. Porém, esses problemas também são refletidos na atualidade, visto que, mesmo com a evolução dos computadores e técnicas para o

Desenvolvimento de *Software* (DS) “ainda hoje diversos projetos de desenvolvimento de *software* fracassam” (NETO, 2007). Com isso, fica a pergunta: será que a crise do *software* se perpetua até hoje?

O aumento da demanda tecnológica, e a dependência das organizações em relação a *software* são visíveis, pois muitas vezes atender aos requisitos exigidos pelos clientes para que esses realizem suas atividades de modo a não haver falhas, tem sido um desafio, principalmente quando o Desenvolvimento do *Software* é feito a partir de uma abordagem informal. A partir dessa premissa, surge a necessidade de um processo em que o desenvolvimento de *software* seja bem definido. Para Sommerville (2011) “a ES se ocupa de todos os aspectos da produção de *software*, desde os estágios iniciais de especificação do sistema, até a manutenção desse sistema, depois que ele entrou em operação”. Por tanto, “a crise acabou para aqueles que utilizam a Engenharia de *Software* como base para a construção de produtos de *software*” (NETO, 2007).

Assim como nas demais engenharias, existem processos adequados á condução das atividades de desenvolvimento dos produtos, neste caso, o *software*, que como todo produto possui um ciclo de vida, ou seja, etapas que são relacionadas à sua existência. Dessa forma o processo de *software* guia a construção de um produto de *software* através de um modelo de ciclo de vida.

Existem vários Modelos de Ciclo de Vida ou Modelos de Processo de *Software*, tais como: Modelo em Cascata; Modelo Incremental; Modelo RAD (Desenvolvimento Rápido de Aplicações); Modelo Espiral; Modelo de Desenvolvimento Concorrente; RUP (*Rational Unified Process*).

Nas próximas sessões será detalhado o processo de desenvolvimento de *software* e alguns dos modelos de ciclo de vida existentes.

2.3 Processos de Desenvolvimento de *Software*

Para Filho (2003, p.11) na ES, Processos de *Software* pode ser definidos para:

> *“Atividades como desenvolvimento, manutenção, aquisição e contratação de software. Pode-se também definir subprocessos para cada um desses; por exemplo, um processo de desenvolvimento abrange subprocessos de determinação dos requisitos, análise, desenho, implementação e testes. Em*

> *um processo de desenvolvimento de software, o ponto de partida para a arquitetura de um processo é a escolha de um modelo de ciclo de vida."*

Os processos de DS são as atividades destinadas à produção de um produto de *software* ou evolução de um já existente organizado de maneira sequencial e iterativa.

As funcionalidades de um *software* são diferentes das de outro, com isso não existe um processo comum para todo desenvolvimento. E embora existam muitos processos de *software* diferentes, há atividades fundamentais comuns a todos eles (SOMMERVILLE, 2011).

Figura 1 - Processo de *software*

Fonte: Adaptado de (SOMMERVILLE, 2011).

A Figura 1 mostra quatro atividades do processo, que podem ser organizadas de forma diferente conforme o modelo de processo, o tipo de *software* e a estrutura organizacional envolvida.

A Especificação do *Software* ou engenharia de requisitos define a funcionalidade do *software* e identifica as restrições do desenvolvimento do sistema. Esse processo é crítico, visto que, erros nessa fase geram problemas na implementação do *software*. Esse processo tem como objetivo produzir um documento que especifique os requisitos exigidos pelo cliente. Segundo Sommerville

(2011), existem quatro atividades principais do processo de engenharia de requisitos:

- Estudo de viabilidade. É um estudo onde é feita uma estimativa da possibilidade de satisfazer as necessidades do usuário, considerando se o sistema será rentável do ponto de vista de negócio e se ele cabe dentro dos orçamentos. O resultado desse estudo de viabilidade deve indicar a decisão de avançar ou não;
- Elicitação e análise de requisitos. É o processo que faz a derivação dos requisitos através da observação dos sistemas existentes, da observação de tarefas e do diálogo com os possíveis usuários. Através dessa derivação de requisitos podem surgir mais de um modelo de sistema, o qual ajuda a entender melhor o sistemas a ser especificado;
- Especificação de requisitos. “É a atividade de traduzir as informações obtidas durante na atividade de análise em um documento que defina um conjunto de requisitos” (SOMMERVILLE, 2011, p.25). Nesse documento é incluso os requisitos do usuário, com descrições abstratas do sistema, e os requisitos de sistema que descreve de forma mais detalhada como o sistema deve funcionar;
- A validação de requisitos. Nessa atividade é feita a verificação dos requisitos. Erros na especificação dos requisitos podem ser identificados no documento de requisitos, portanto deve ser modificado para correção dos problemas.

Essa atividade de especificação de software emerge durante todo o processo sendo então intercalada de acordo com as prioridades do usuário.

A atividade de Projeto e Implementação de Software é onde é feita a execução ou desenvolvimento do software de modo que cumpra o que foi especificado. Nessa etapa pode haver variações nas atividades do projeto dependendo do tipo de sistema que será desenvolvido. Sommerville (2011) ressalta quatro atividades que podem ser parte do processo de projeto de sistemas:

- Projeto de arquitetura. Identifica a estrutura do sistema, seus componentes, os relacionamentos e como eles se distribuem;
- Projeto de interface. Define as interfaces de cada componente do sistema;
- Projeto de componente. Define as funcionalidades de cada componente;

- Projeto de banco de dados. Define a estrutura de como os dados serão representados no banco de dados do sistema.

A Validação de *Software* é a atividade que mostra se o sistema atende aos requisitos especificados, inclusive do cliente. Neste processo testes de programa são feitos simultaneamente com dados simulados, sendo essa a principal técnica de validação. Com relação aos estágios de testes, Sommerville (2011) ressalta que os componentes do sistema são testados, em seguida o sistema integrado é testado e, finalmente, o sistema é testado com os dados do cliente. Idealmente os defeitos de componentes são descobertos no inicio do processo, e os problemas de interface são encontrados quando o sistema é integrado. No entanto, quando os defeitos são descobertos, o programa deve ser depurado e isso pode requerer que outros estágios do processo de teste sejam repetidos. Erros em componentes de programa podem vir a luz durante os testes de sistema. O processo é, portanto, interativo, com informações realimentadas de estágios posteriores para partes anteriores do processo.

A atividade de Evolução do *Software* é um processo onde mudanças podem ser feitas durante ou após o desenvolvimento do sistema, sendo assim um processo evolutivo, onde o *software* é alterado durante seu período de vida, pelas mudanças de requisitos e pelas necessidades do cliente.

As Modificações ou Manutenção de *Software* acontecem quando erros são corrigidos, quando há necessidade de novos requisitos ou quando o cliente solicita novas funções (PRESSMAN, 2006).

2.4 Modelos de Processos de *Software* (ou Modelos de Ciclo de Vida)

Para (SOMMERVILLE, 2011, p.19), Modelos de Processo de *Software* e definido como:

> *"Um representação simplificada de um processo de software. Onde cada modelo representa uma perspectiva particular de um processo e, portanto, fornece informações parciais sobre ele. Por exemplo, um modelo de atividade do processo pode mostrar as atividades e sua sequência, mas não pode mostrar os papéis das pessoas envolvidas."*

Os modelos ditam a ordem em que as atividades relacionadas a métodos e ferramentas devem ser utilizadas.

É importante ressaltar que não existe um modelo de processo que se encaixe perfeitamente para uma organização. Segundo (PRIKLADNICKI, 2003, p.23) "a escolha depende do tamanho da organização, da experiência da equipe, da natureza e da complexidade da aplicação, do prazo de entrega, entre outros fatores".

A seguir abordo alguns dos modelos de processos, como: Modelo em Cascata; Modelo Incremental; e Modelo em Espiral. Vale frisar que esses modelos não são exclusivos, e podem ser usados em conjunto para o desenvolvimento de sistemas, ou seja, podem ser ampliados ou adaptados para criar processos de Desenvolvimento de *Software*.

2.4.1 Modelo em Cascata

Algumas vezes chamado de ciclo de vida clássico, o modelo em cascata sugere uma abordagem sistemática e sequencial para o DS (PRESSMAN, 2006). Esse modelo considera as atividades de Especificação de Requisitos, Projeto, Implementação e Teste, como mostra a Figura 2.

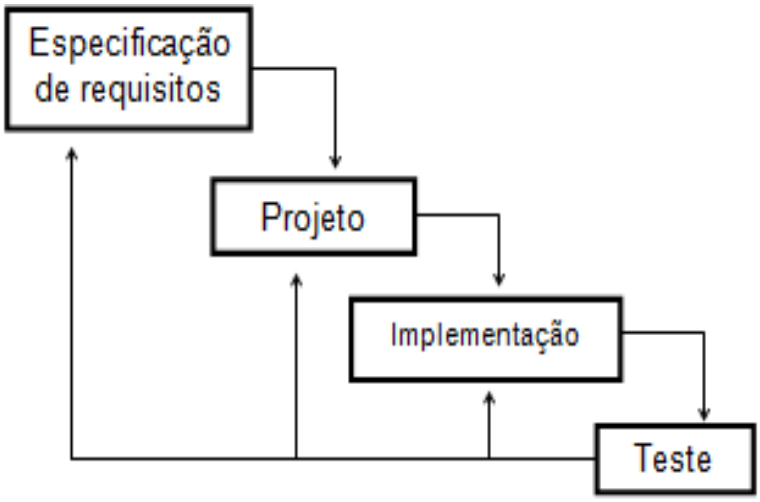

Figura 2 - Modelo em Cascata

Fonte: Adaptado de (PRESSMAN, 2006).

1. Especificação de Requisitos. Levantamento das necessidades do cliente incluindo os recursos e as funcionalidades do sistema, depois de elencados os requisitos, devem ser documentados para possíveis consultas e revisões

do cliente.

2. Projeto. O processo de projeto de sistemas agrupa os requisitos em sistemas de hardware ou de *software* (SOMMERVILLE, 2011).
3. Implementação. Essa implementação acontece de acordo com as especificações, e pode acontecer por unidade de programa, e ao final da implementação essas unidades são integradas para formar todo o sistema.
4. Teste. É a fase em que as unidades de programas e o seu conjunto de unidades são testadas para garantir que o sistema atende aos requisitos exigidos pelo cliente.

A Tabela 1 descreve alguns pontos importantes referentes a esse modelo em questão.

Tabela 1 - Vantagens e Desvantagens do Modelo em Cascata

VANTAGENS	DESVANTAGENS
Esse modelo foi um dos mais utilizados no desenvolvimento de software.	Os projetos reais raramente seguem o fluxo que o modelo propõe.
Chamou a atenção para outros tipos de ciclo de vida.	E difícil para o cliente especificar todos os requisitos do sistema no inicio do projeto.
	E necessário que o cliente tenha paciência, pois um erro grosseiro pode ser desastro se não for detectado até que o programa executável seja revisto.

Fonte: (PRESSMAN, 2006. SOMMERVILE, 2011).

O Modelo em Cascata foi muito utilizado no passado, hoje em dia ele vem sendo descartado, pois segundo Neto (2007, p.31) “... na teoria é uma maravilha. Na prática é rígido e burocrático, tendo sido considerável inviável pelas organizações que o utilizam!”.

2.4.2 Modelo Espiral

Também conhecido como paradigma de *Boehm* (1991), foi desenvolvido para englobar as melhores características dos ciclos de vida Clássico e Incremental, ao mesmo tempo em que adiciona um novo elemento, a analise de risco, que não

existe nos modelos anteriores(AUDY; PRIKLADNICKI, 2008).

A Figura 3 mostra os *loops* do espiral que são divididos em quatro atividades onde cada *loop* representa uma fase do processo (SOMMERVILLE, 2011):

Figura 3 - Modelo em Espiral

Fonte: (SOMMERVILLE, 2011).

1. Definição de Objetivos. São definidos quais os objetivos, as restrições os riscos e as estratégias para cada fase do processo;
2. Avaliação e Redução dos Riscos. Consiste na análise, identificação dos riscos de uma forma detalhada onde se procura a resolução e possíveis diminuições dos riscos;
3. Desenvolvimento e Validação. É a fase em que o desenvolvimento do produto se inicia, onde é selecionado um modelo e desenvolvimento;
4. Planejamento. É feita a revisão do projeto e a elaboração de planos para a próxima fase.

Baseado principalmente em decisões de prosseguir ou não, de acordo com a avaliação, seja do cliente ou da equipe responsável pelo projeto, o modelo espiral tende a uma trajetória que ruma para o modelo mais completo do sistema (AUDY; PRIKLADNICKI, 2008). As principais vantagens e desvantagens desse modelo são

mostradas na Tabela 2

Tabela 2 - Vantagens e Desvantagens do Modelo em Espiral

VANTAGENS	DESVANTAGENS
Possibilita a evolução do software	Pode ser difícil convencer os clientes que a abordagem evolucionária é controlável.
E um modelo interativo que permite tanto a avaliação de interação quanto a interação com o usuário.	Se um risco importante não for descoberto e gerenciado, fatalmente ocorrerão problemas.
Utiliza a analise de risco.	Por ter um custo e uma complexidade mais elevada que os demais modelos, a relação custo/beneficio pode não ser satisfatórias.

Fonte: (AUDY; PRIKLADNICKI, 2008. PRESSMAN, 2006).

2.4.3 Modelo Incremental

O Modelo Incremental como o nome sugere, é uma abordagem intermediaria que tem relação tanto com o Desenvolvimento Clássico quanto o Evolucionário.

O Modelo Incremental tem o objetivo de apresentar um produto operacional a cada incremento, onde os primeiros incrementos são versões simplificadas do produto final, mas oferecem capacidades que servem ao usuário, além de uma plataforma para avaliação do usuário (PRESSMAN, 2006).

As atividades do Modelo Incremental são semelhantes à de outros ciclos de desenvolvimento, pois para iniciar um processo de desenvolvimento são necessários os requisitos e a elaboração de um projeto baseado nesses requisitos, com isso pode-se iniciar a construção do protótipo para que esse seja avaliado pelo cliente e pelo usuário.

É apresentada na Figura 4 a abordagem do Desenvolvimento Incremental que é usada como um meio de reduzir o ‘retrabalho’ no processo de desenvolvimento e de proporcionar aos clientes algumas oportunidades de adiar decisões sobre seus requisitos detalhados, até que eles tenham alguma experiência com o sistema (SOMMERVILLE, 2011).

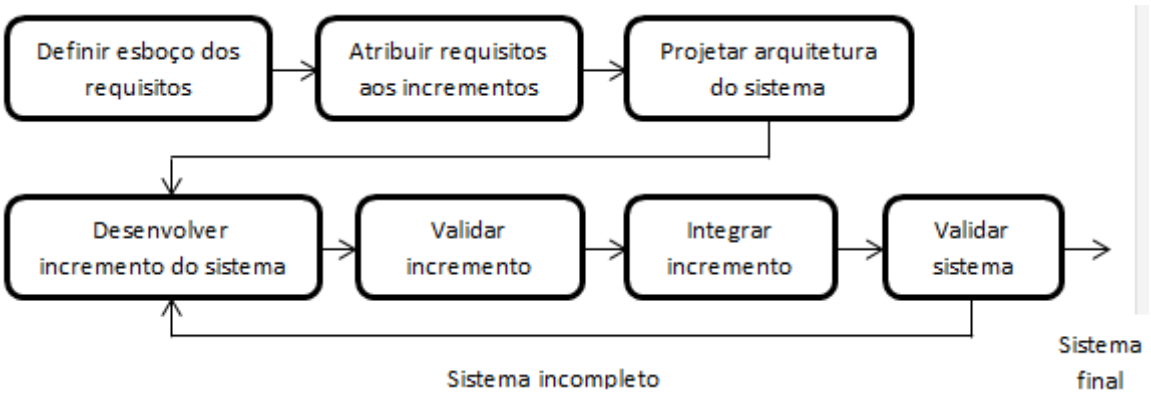

Figura 4 - Modelo Incremental

Fonte: Adaptado de (SOMMERVILLE, 2011).

A Tabela 3 mostra algumas vantagens e desvantagens na utilização do Modelo Incremental:

Tabela 3 - Vantagens e Desvantagens do Modelo Incremental

VANTAGENS	DESVANTAGENS
Menor risco de perda total do sistema	Dificuldade de mapear os requisitos dos clientes dentro de incrementos de tamanho correto.
A avaliação do cliente faz com que sua participação contribua para a modelagem do sistema.	O fato de acelerar as entregas de protótipos aos clientes pode fazer com que o desenvolvedor escolha uma linguagem ou um sistema operacional que seja mais simples de utilizar, levando o produto muitas vezes a ineficiência.

Fonte: (PRESSMAN, 2006. SOMMERVILE, 2011).

2.5 Desenvolvimento Ágil de *Software*

O Desenvolvimento de *Software* possui vários estados ao longo do seu ciclo de vida, pelo fato de sofrer mudanças como manutenção, retrabalho, descumprimento dos prazos que geram aumento dos custos e consequentemente a insatisfação do cliente. Com isso, o projeto de desenvolvimento pode ter seu

desempenho comprometido ou terminar fracassado.

Neste cenário, o Desenvolvimento Ágil surgiu como uma nova alternativa em resposta a esses problemas relacionados ao Desenvolvimento de *Software*. Em fevereiro de 2001 um grupo de pesquisadores autodenominados Aliança Ágil se reuniram na estação de Esqui *Snowbird* nos Estados Unidos iniciaram uma discussão sobre como desenvolver *software* de forma mais rápida, simples e eficaz. A partir dessa discussão surgiu o Manifesto Ágil, uma declaração das características que fundamentam o processo ágil (CHAVES, 2010. ZANATTA, 2004).

Conforme (PRESSMAN, 2006) o Manifesto para o Desenvolvimento Ágil de Software declara:

- Indivíduos e interações em vez de processos e ferramentas;
- Softwares funcionando em vez de documentação abrangente;
- Colaboração do cliente em vez de negociação de contratos;
- Resposta à modificação em vez de seguir um plano.

O Desenvolvimento Ágil determina uma nova tendência de processo com características focadas principalmente na adaptação a mudanças, pouca documentação e maior qualidade de código, ciclo de vida iterativo e incremental e fundamentado na participação do cliente no decorrer do projeto.(PRESSMAN, 2006. SOMMERVILLE, 2011).

A aliança ágil define 12 princípios para alcançar a agilidade (PRESSMAN, 2006):

1. Prioridade em satisfazer o cliente através da entrega continua do *software*.
2. As modificações dos requisitos são aceitas durante todo o desenvolvimento.
3. Entrega frequente de *software* no espaço de tempo de duas semanas a no máximo dois meses.
4. Trabalho em conjunto e diário entre os desenvolvedores e o pessoal de negocio.
5. O projeto deve ser construído em torno de indivíduos motivados.
6. A conversa face a face é o melhor método de levar informação para uma equipe de desenvolvimento.
7. O progresso é baseado pelo funcionamento do *software*.
8. Processos ágeis promovem desenvolvimento sustentável. E o ritmo deve ser constante por parte dos patrocinadores, desenvolvedores e usuários.
9. A agilidade necessita de uma atenção continua à excelência técnica.

10. A simplicidade é essencial.
11. Equipes auto organizadas geram as melhores arquiteturas, requisitos e projetos.
12. Para se tronar efetiva, a equipe reflete em intervalos regulares o seu comportamento.

Mesmo com tais princípios, o manifesto ágil não rejeita os processos e ferramentas, a documentação, a negociação de contratos ou o planejamento, mas simplesmente mostra que eles têm importância secundária quando comparados aos indivíduos e interações, à colaboração do cliente e às respostas rápidas a mudanças e alterações (ARAUJO, 2009). Cabe ressaltar, que o manifesto ágil não é contra os modelos tradicionais de desenvolvimento, ele é apenas uma proposta que busca uma melhoria no processo de desenvolvimento de *software*, tornando-o mais rápido e menos burocrático.

Existem vários modelos ágeis, entre os quais destacam-se: *Extreme Programming* (XP); DAS – Desenvolvimento Adaptativo de Software (*Adaptative Software Development* - ASD); DSDM (*Dynamic Systems Development Method* – Método de Desenvolvimento Dinâmico de Sistemas); *Scrum*; Crystal; FDD (*Feature Driven Development* – Desenvolvimento Guiado por Características). Esses modelos satisfazem ao Manifesto Ágil, possuindo assim muitas semelhanças em suas abordagens. A seguir serão apresentados alguns desses Modelos Ágeis.

2.5.1 *Extreme Programming* (XP)

Esse modelo teve inicio na década de 1980, tendo como *fundadores Kente Beck e Ward Cunninghan*, mas somente foi publicado em 1999.

O XP usa uma abordagem orientada a objetos, e inclui um conjunto de regras e práticas que ocorrem no contexto de quatro atividades de arcabouço como mostra a Figura 5 (PRESSMAN, 2006):

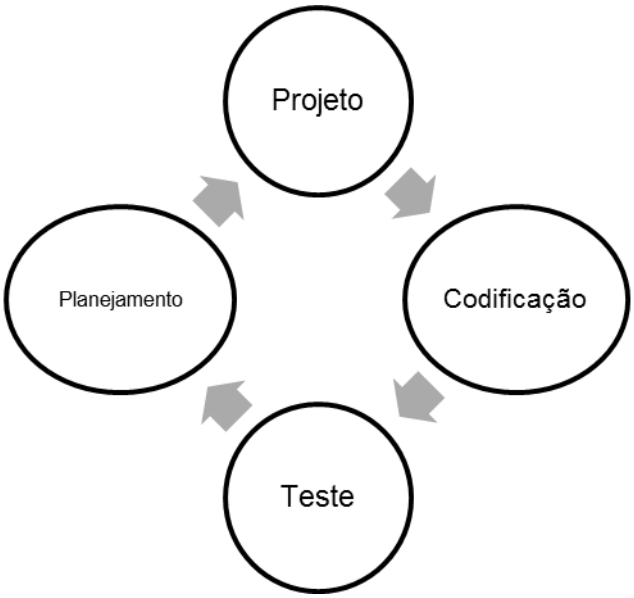

Figura 5 - *Extreme Programming* (XP)
Fonte: Adaptado de (PRESSMAN, 2006).

- Planejamento. Essa atividade se inicia com à criação de varias histórias descritas pelo cliente na intenção de determinar os requisitos necessários para a construção do *software*. Cliente e equipe trabalham juntas. A quantidade de historias implementadas na primeira versão determina a velocidade de projeto;
- Projeto: Encoraja o uso de cartões CRC (*Class-Responsibility-Colaborator*). Esses cartões identificam e organizam as classes orientadas a objetos que são relevantes para o incremento do *software* atual. O projeto XP é um artefato provisório que é modificado à medida que a construção continua;
- Codificação: uma vez completado o código, ele pode ser submetido imediatamente ao teste unitário, fornecendo assim *feedback* instantâneo para os desenvolvedores (PRESSMAN, 2006). Nessa atividade o XP recomenda a programação em pares, ou seja, duas pessoas, mais especificamente programadores, trabalham juntos para criar o código baseado na historia do cliente;
- Teste: quando os testes unitários são validados, esses são integrados em uma sequencia universal de testes. Os testes devem acontecer sempre que o código for modificado. Nesta atividade o cliente também participa dos testes, sendo esses revisados de uma forma global, ou seja, o cliente foca nas características e funcionalidades do sistema, sendo esses aceitos ou não, a

depender das historias do usuário se foram implementadas da forma especificada pelo cliente, sendo estes testes chamados testes de aceitação ou testes do cliente (PRESSMAN, 2006).

2.5.2 Desenvolvimento Adaptativo de *Software* (DAS)

Proposto por *Jim Highsmith* como uma técnica para construção de sistemas e *softwares* complexos (PRESSMAN, 2006), o Desenvolvimento Adaptativo de *Software* é centrado a partir da colaboração e organização da equipe, sendo essa responsável por criar resultados emergentes, ou seja, um resultado além da capacidade de um único agente.

Ele define um “Ciclo de Vida” que incorpora três fases como mostra a Figura 6:

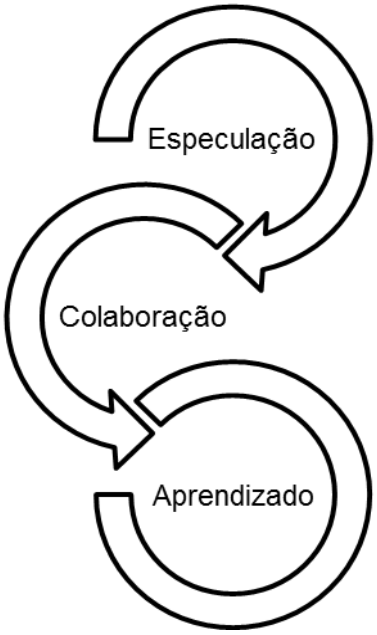

Figura 6 – DAS

Fonte: Adaptado de (PRESSMAN, 2006).

- Especulação: fase de inicio do projeto de adaptação onde são feitas as declarações e restrições do projeto pelo cliente, além da definição dos requisitos básicos para os possíveis incrementos de *software* necessários ao projeto (PRESSMAN, 2006);

- Colaboração: é a forma de se trabalhar em grupos de forma a multiplicar os resultados onde a confiança deve prevalecer;
- Aprendizado: a medida que os componentes da equipe DAS desenvolve os componentes do ciclo adaptativo, a ênfase está tanto no aprendizado quanto no progresso em direção a um ciclo completo (PRESSMAN, 2006).

O DAS produz resultados com rapidez e pode ser utilizado em projetos que necessitem de avaliação constante dos clientes. O planejamento pode ser adaptado em qualquer fase do projeto. Porém, ambos, os desenvolvedores e clientes devem estar comprometidos com o passo rápido do projeto, ou seja, se algum grupo do projeto não estiver acompanhando, o DAS pode falhar. E quando os riscos do projeto gerados por essas falhas forem altos, o DAS não pode ser adaptado (PRESSMAN, 2006).

2.5.3 *Scrum*

O *Scrum* é um modelo muito conhecido e sem duvida o mais utilizado atualmente. É um modelo ágil desenvolvido por *Ken Schwaber* e *Jeff Sutherland* e sua equipe na década de 1990 (PRESSMAN, 2006). Seus princípios incorporam os requisitos, análise, projeto, evolução e entrega como atividades de desenvolvimento de um processo em que cada tarefa ocorre dentro de um padrão chamado *Sprint*.

O *Scrum* é um modelo que gerencia um processo de desenvolvimento de software não definindo técnicas nem ferramentas, ficando a critério do desenvolvedor como será este processo, definindo apenas como equipes devem trabalhar, com um ambiente em que os requisitos necessitam de mudanças e alterações constantes (ZANATTA, 2004).

O funcionamento do processo *Scrum* é mostrado na Figura 7:

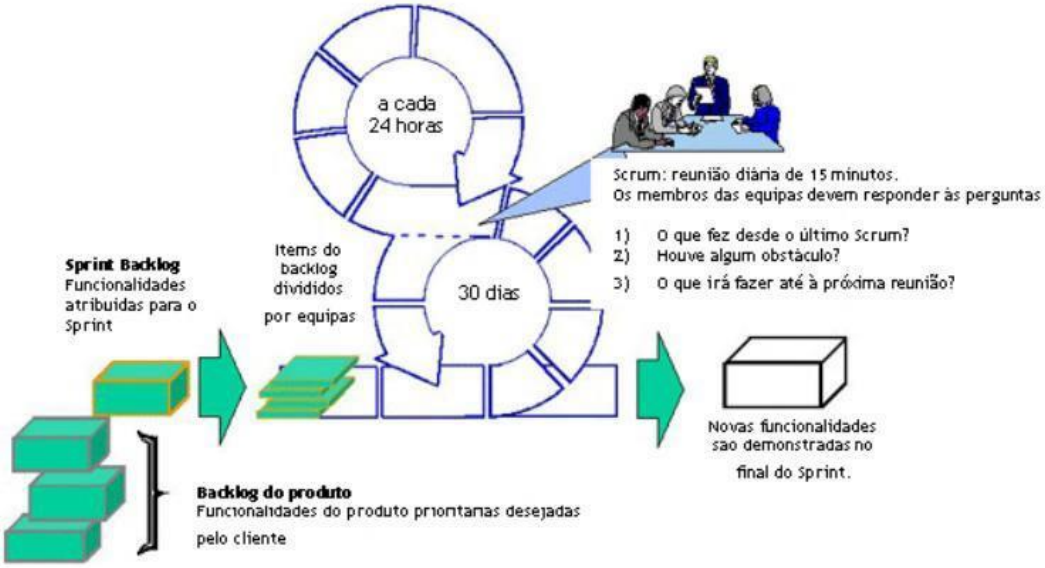

Figura 7 - Processo *Scrum*

Fonte: (BARBAN; BERTHOLDO, 2010, p.15).

- O *Product Owner*, juntamente com o *Scrum Master* são responsável por manter a lista priorizada de requisitos que devem ser implementados do *Product Backlog*.
- A cada *Sprint*, o time decide quais itens do *Product Backlog* serão implementados *Sprint Backlog* em uma reunião especial denominada *Sprint Planning Meeting*.
- O time é responsável por implementar a funcionalidade escolhida e demostrar ao *Product Owner* do *Sprint – Sprint Review*.
- No final do *Sprint* o time faz uma revisão do trabalho – Retrospectiva do *Sprint*.
- Uma reunião diária de 15 minutos denominada *Daily Scrum* é realizada para avaliar os possíveis impedimentos do projeto.

Papeis *Scrum*

No processo *Scrum*, existe diferentes papéis e suas respectivas responsabilidades:

- *Scrum Master*: é um “líder facilitador” que trabalha próximo ao *Product Owner*.

Seu papel é garantir que a equipe esteja funcionando e produzindo totalmente. Deve possibilitar o trabalho cooperativo entre pessoas de diferentes papeis e funções, remover os impedimentos que surgirem e proteger a equipe de interferências externas. O *Scrum Master* também deve assegurar que o processo definido esta sendo seguido (CHAVES, 2010).

- *Product Owner*: este papel é representado por uma pessoa, que é responsável, por manter e priorizar o desenvolvimento dos itens do *Product Baclog* desempenha um papel formal para assumir as responsabilidades do projeto, sendo responsável pela liberação e escolha dos itens que passarão à outra fase, para o desenvolvimento do produto de *software* (ZANNATA, 2004).
- Time *Scrum*: é responsável pelas ações de cada *Sprint*. Ela determina a criação da *Sprint Baclog*, além de revisar o *Product Backlog* para detectar impedimentos e retira-los para alcançar o sucesso do projeto (ZANNATA, 2004).

Práticas do *Scrum*

O *Scrum* possui um conjunto de regras que devem ser seguidas pelo fato de não fornecer um método especifico para o desenvolvimento de *software*. Algumas práticas são apresentadas a seguir:

Product Backlog. Inicio do *Scrum*, através da coleta dos requisitos. É uma lista de atividades que provavelmente serão desenvolvidas durante o projeto (ZANNATA, 2004).

A principal regra é não adicionar nenhum item ao *Product Backlog* fora da reunião, garantindo o comum acordo entre os envolvidos. A Figura 8 mostra a composição do *Product backlog.*

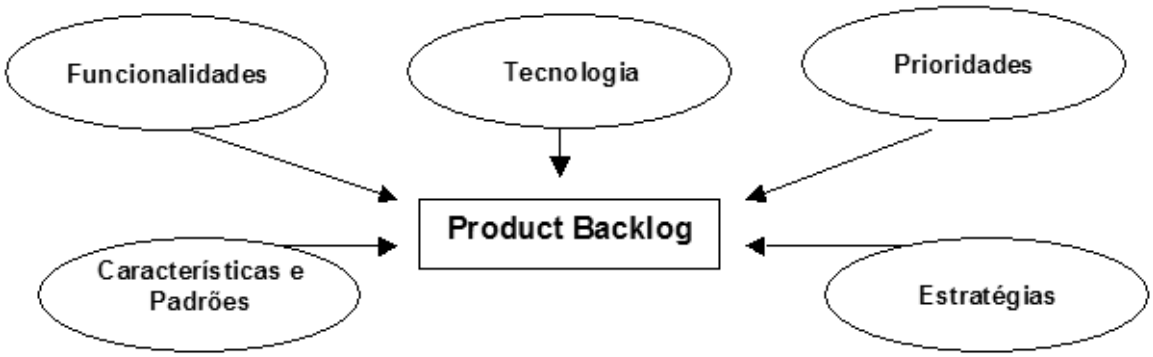

Figura 8 - Origens para a Confecção do *Product Backlog*

Fonte: (ZANNATA, 2004, p. 51)

Daily Scrum. Reuniões diárias ou em dias alternados com duração de 15 à 30 minutos cujo objetivo é a identificação de possíveis problemas encontrados pela equipe.

Nessa reunião o *Scrum Master* levanta três questões (PRESSMAN, 2006)

1. O que eu fiz desse a ultima reunião?
2. O que vou fazer antes da próxima reunião?
3. O que me impede de realizar o meu trabalho da forma mais eficiente possível?

Portanto, cada membro da equipe tem a oportunidade de anunciar os obstáculos durante a reunião, onde o *Scrum Master* se responsabiliza em resolver os impedimentos.

Sprint. É a principal fase do *Scrum* por ser o momento de executar o que foi definido no *Product Backlog*. Cada *Sprint* pode durar de uma a quatro semanas, sendo trinta dias o prazo máximo (BLOMER, 2005).

As práticas de cada *Sprint* do *Scrum* apresentam características especificas do ciclo de desenvolvimento tradicional, onde cada ciclo corresponde a uma *Sprint* que é planejado para durar de uma semana a um mês. A *Sprint* é composta pelas fases:

- Analise: é a ideia, a descrição do jogo, a especificação dos requisitos;
- Projeto: o estudo, a definição de como as ações serão implementadas;
- Implementação: codificar de acordo com o desenho;
- Teste: busca a ocorrência de erros nas implementações, procurando corrigi-los.

Sprint Planning Meeting. É a reunião de planejamento do *Sprint* que analisa os itens no *Product Backlog* com o *Product Owner* com a finalidade de priorizar os itens

a serem desenvolvidos na *Sprint* (ZANNATA, 2004).

Sprint Backlog. Define o trabalho para um *Sprint*, representado pelo conjunto de tarefas que devem ser cumpridas para realizar os objetivos da *Sprint*, e um conjunto de itens do *backlog* do produto selecionado (BLOMER, 2005).

Sprint Review. No último dia do *Sprint*, a equipe e o *Scrum Master* apresentam os resultados em uma reunião informal ao cliente, ao gerente e ao *Product Owner*, que farão a análise destes resultados verificando as novas atividades que poderão integrar com *Product Backlog*. O *Scrum* não exige todos os pré-requisitos logo no início do projeto, pois estes são “descobertos” à medida que o projeto evolui (ZANATTA, 2004).

Estrutura do Funcionamento do Scrum

A estrutura de funcionamento do *Scrum* é dividida em três fases: *PreGame, Game e PostGame* (BLOMER, 2005). A fase *PreGame* (Pré-Planejamento), fase inicial que se divide em dois estágios: Planejamento e arquitetura. O planejamento consiste em desenvolver os requisitos conforme o *Product Backlog* para o sistema, define a entrega das versões e as datas para realização das tarefas. Na arquitetura os itens do *Product Backlog* são revisados, posteriormente são priorizados para o desenvolvimento de cada requisito da aplicação (ZANNATA, 2004).

A fase *Game*, é a parte do desenvolvimento, é um ciclo interativo onde as equipes se encontram para revisar os planos das versões. É realizado o desenvolvimento incremental, e inseridos novos itens no *Product Backlog* (ZANNATA 2004).

Na etapa final, *PostGame* é feita a preparação para entrega do resultado final, onde são feitas as etapas de integração, documentação ao usuário e preparação de material para treinamento (BLOMER, 2005).

2.6 Principais Problemas e Desafios do Desenvolvimento de *Software*

O desenvolvimento de *software* é uma atividade que possui diversos desafios, esses são relativos ao próprio desenvolvimento e também ocasionados por

problemas esses que são relatados na literatura.

Entre os vários problemas e desafios sofridos pelo DS pode-se identificar e caracterizar os principais nas próximas seções.

2.6.1 Capacitação de pessoal

Para obter sucesso, uma organização de DS necessita de profissionais capacitados, ou seja, que tenham o domínio da aplicação. Porém na prática, gerentes de nível médio e superior sem nenhum conhecimento em softwares recebem a responsabilidades pelo seu desenvolvimento (PRIKLADNICKI, 2003). Além disso, nota-se que na maioria das organizações os profissionais em tecnologia da informação têm recebido pouco treinamento sobre as novas tecnologias lançadas.

Com isso podemos notar que é necessário que as organizações adotem uma politica de seleção de pessoal. Algumas empresas utilizam vários tipos de testes para avaliar candidatos, incluindo testes de aptidão em programação e testes psicométricos (SOMMERVILLE, 2011). Esses testes são importantes para descobrir a real posição desse pessoal em relação à resistência a mudanças e suas atitudes em relação à execução de certos tipos de tarefas.

2.6.2 Trabalho em equipe

As equipes de DS são formadas por profissionais que trabalham na solução de um problema. E é importante que o grupo tenha o equilíbrio certo entre as habilidades técnicas, a experiência e as personalidades individuais (SOMMERVILLE, 2011).

De fato, muitos problemas surgem pela falta de comunicação e quando os envolvidos possuem seus próprios objetivos pessoais.

Para tanto, uma estrutura de equipe bem sucedida depende do estilo de gestão de sua organização, da quantidade de pessoas que formarão a equipe e seus níveis de aptidão, e da dificuldade geral do problema (PRESSMAN, 2006).

2.6.3 Produtividade

O grau de produtividade em um processo de DS, esta ligada a facilidade de produzir. Isso acontece quando existem fatores colaboradores para esse rendimento, tais como o alcance da demanda de serviços, o acompanhamento das necessidades do cliente.

Porém os problemas relacionados à produtividade acontecem quando os projetos são normalmente executados de qualquer maneira, com um vago indício das necessidades do cliente, e somente a comunicação entre o cliente e o desenvolvedor é insuficiente (PRIKLADNICKI, 2003).

2.6.4 Desenvolvimento Distribuído de *Software* (DDS)

Devido ao grande aumento da demanda de produção de *softwares*, empresas voltadas para a área tecnológica principalmente para o Desenvolvimento de *Software* estão expandindo as fronteiras e abrindo uma nova forma de negócio, o DDS, este se apresenta de forma que os membros das equipes trabalham de forma distribuída, ou seja, em diferentes localidades, estados ou países.

Os engenheiros de *software* têm reconhecido a grande influência dessa nova forma de trabalho no seu dia-a-dia e estão em busca de modelos que facilitem o DS com equipes geograficamente distantes (AUDY; PRIKLADNICKI, 2008). É importante ressaltar que a complexidade e os desafios enfrentados pelos gerentes são frequentes e desafiadores nesse ambiente distribuído

2.6.5 Especificação de Requisitos

Um dos maiores desafios do DS esta na especificação de requisitos. Uma especificação pode ser um documento escrito, um modelo gráfico, um modelo matemático formal, uma coleção de cenários de uso, um protótipo ou qualquer combinação desses elementos (PRESSMAN, 2006). E para que estes requisitos sejam especificados de forma suficiente, o tempo de levantamento pode ser maior que o esperado, além disso, durante o processo de desenvolvimento pode ser

introduzidos novos requisitos, e consequentemente o aumento da dificuldade de implementação do sistema.

2.6.6 Custo

Para serem definidos os custos é necessária uma análise das estimativas do processo, o tempo e os esforços gastos no processo de desenvolvimento. Mas da mesma forma que ocorre com a definição de prazos, muitas vezes não existem dados suficientes para se determinar um custo adequado (PRIKLADNICKI, 2003). Devido a esses fatores a análise de custos pode ser avaliada de forma errada, de modo a não considerar os benefícios envolvidos.

2.7 Desenvolvimento Distribuído de *Software*

Nos últimos anos o Desenvolvimento Distribuído de *Software* (DDS) vem se tornando um atrativo notável e crescendo em ritmo acelerado para empresas voltadas para o desenvolvimento de *software*, segundo os autores (AUDY *et al.*, 2003) observou-se na última década um grande investimento na conversão de mercados nacionais em mercados globais, criando novas formas de competição e colaboração entre os países.

> *"O DDS é um modelo de desenvolvimento de software onde os envolvidos em um determinado projeto estão dispersos". (JÚNIOR, 2011, p.18)*

Vários fatores colaboraram para que o DDS se tornasse uma solução para empresas que procuram por mão de obra qualificada e melhores custos na produção, (AUDY *et al.*, 2003, p.) apresenta os fatores que tem contribuído para o crescimento do DDS:

- Custo mais baixo e disponibilidade de mão de obra;
- Evolução e maior disponibilidade de recursos de telecomunicações
- Evolução das ferramentas de desenvolvimento;
- A necessidade de possuir recursos globais para utilizar a qualquer hora;
- A vantagem de estar perto do mercado local;
- A formação de equipes virtuais para explorar as oportunidades de mercado;

- A pressão para o desenvolvimento *time-to-market*, utilizando as vantagens proporcionadas pelo fuso horário diferente, no desenvolvimento conhecido como *round-the-clock*, ou seja, o desenvolvimento quase que continuo.

No mercado de Desenvolvimento de *Software* é importante ressaltar que existem dificuldades tanto no desenvolvimento centralizado, ou seja, que possuem a mesma localização física, quanto no distribuído, e que são geradas principalmente por falhas em projetos, pela falta de recursos, pessoas capacitadas e um aumento significativo na demanda de produção. Porém existem algumas diferenças no DDS que são adicionadas as dificuldades de um processo de desenvolvimento que são citadas por Audy; Prikladnicki (2008, p.86), tais como: “dispersão geográfica (distância física), dispersão temporal (diferenças de fuso horário) e diferenças sócio culturais (idioma, tradições, costumes, normas e comportamento).”

Apesar das dificuldades, varias vantagens são identificadas, pois muitas organizações encontraram no DDS uma alternativa, experimentando o desenvolvimento em locais remotos (PRIKLADNICKI, 2003). Como em todo processo de desenvolvimento, se for planejado de forma correta o DDS pode ser um colaborador considerável para as operações que envolvem de uma forma geral o desenvolvimento de *software*, e para que isso aconteça se faz necessária a utilização de algumas ferramentas que auxiliam as equipes e que são focadas principalmente nos processos de comunicação que podem ser através de documentos e canais não interativos.

2.7.1 Motivações para a Prática de DDS

Apesar do acréscimo das dificuldades que são desafiadoras em um projeto de desenvolvimento utilizando as práticas de DDS, não podemos deixar de nos referir a esta nova aplicação como uma crescente e vantajosa forma de melhorar o desenvolvimento de *software*. Para tanto existe várias razões que motivam as organizações a utiliza-se dessa prática, de acordo com a Figura 9 são identificadas algumas delas envolvidas no DDS:

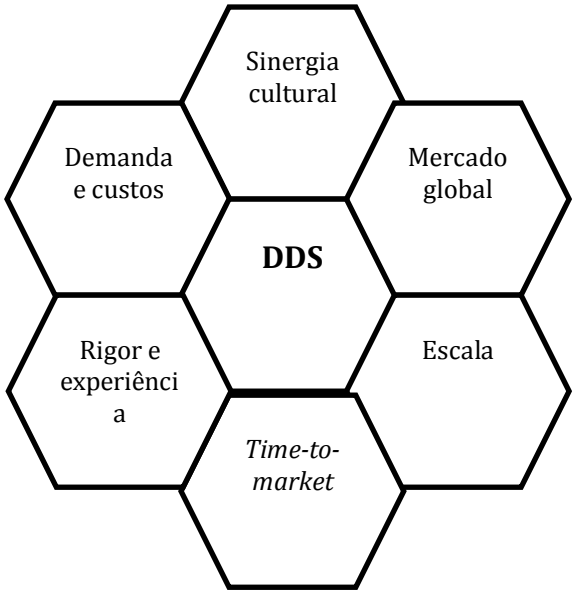

Figura 9 - Principais Razões Envolvidas no DDS

Fonte: Adaptado de (PRIKLADNICKI, 2003).

Sinergia Cultural: durante a execução de um processo de desenvolvimento de *software* muitas questões são analisadas e uma das primordiais é o fato de "quem" vai produzir, espera-se que as pessoas envolvidas sejam profissionais especializados e com uma vasta experiência, no entanto às vezes se torna difícil encontrar pessoas aptas para determinadas tarefas, e para esta questão as práticas de DDS torna-se uma solução, pois o fator dispersão possibilita o encontro desse pessoal através da sinergia cultural, pois a diversidade amplia a criatividade e a inspiração na organização de desenvolvimento de *software* (AUDY; PRIKLDNICKI, 2008).

Mercado Global: com o crescente potencial da distribuição de *softwares*, o mercado da TI aumenta e consequentemente o mercado consumidor que necessita de suporte e manutenção mais próxima. Para essa questão, (CARMEL, 1999) muitas empresas optam pelo DDS para atingir o mercado global e ficar próximas de seus consumidores. Além disso, obtêm o diferencial de se tornar empresas globais, um bom atrativo de *marketing*.

Escala: esse fator também influi na escala de crescimento das empresas de desenvolvimento que conforme Carmel (1999), à medida que aumentam, chegam ao ponto em que ficam muito grandes e difíceis de gerenciar. Com isso torna-se necessário distribuir o desenvolvimento para atender a demanda necessária.

Time-to-market: um dos mais interessantes fatores influenciadores do desenvolvimento distribuído é a rapidez da entrega do produto no mercado (t*ime-to-market*). Isso se deve ao desenvolvimento *follow-the-sun*, ou seja, equipes distribuídas ao redor do globo onde pelo menos uma estará trabalhando.

Rigor e Experiência no Desenvolvimento: a rigorosidade em seguir uma metodologia e práticas de qualidade. E na tentativa de melhorar a comunicação entre os locais de desenvolvimento, tendem a melhorar significativamente a documentação utilizada (CARMEL, 1999).

Demanda de Produção: o aumento da demanda de produção que influência diretamente na contratação de profissionais qualificados em alguns casos, não atinge a suficiência de mercado nacional. Dessa forma, com um número de funcionários insuficiente e custos de contratação altíssimos, a disponibilidade de recursos equivalentes em outras localidades a um custo mais baixo tornou-se um grande atrativo para as empresas de *software*, motivando o DDS (PRIKLADNICKI, 2003).

2.7.2 Níveis de Dispersão dos Tipos de Atores (*Stakeholders*)

Em um projeto de DDS é essencial conhecer as limitações impostas por dimensões num processo distribuído. Para tanto é necessário entender os níveis de dispersão para minimizar os riscos de dificuldades e de caracterização da distribuição.

No desenvolvimento de projetos vários atores podem estar envolvidos, e em diferentes atividades no processo, tais atores são conhecidos como *Stakholders*, e de acordo com Prikladnicki (2003), estes podem ser de diferentes tipos: a equipe de projeto que representa todos os envolvidos no desenvolvimento de um projeto; o cliente que é a pessoa física ou jurídica que solicita e contrata o desenvolvimento de um determinado projeto; e o usuário que representa os responsáveis por fornecer os requisitos para o correto desenvolvimento do projeto e responsáveis por utilizar o produto gerado.

Estes atores podem estar dispersos em diferentes níveis, ou seja, podem estar em diferentes distâncias física. Um estudo realizado por Prikladnicki (2003) classifica os níveis de dispersão entre os *Staksholders*:

- Mesma localização física: é a situação em que a empresa possui todos os atores presentes fisicamente em um mesmo local.
- Distância nacional: os atores são localizados em um mesmo país.
- Distância continental: os atores são localizados em países diferentes.
- Distância global: atores localizados em países diferentes ou em continentes diferentes, formando uma distribuição global.

Vale ressaltar que a característica fundamental que destaca o DDS esta no fator da distância física entre os atores.

2.7.3 Unificações dos Termos de Operacionalização do DDS

Com o mercado de *software* crescendo cada vez mais, gerando com isso estudos para que se possam adaptar as melhores práticas num ambiente distribuído, a diversidade de termos gerados a partir das operações analisadas nesse cenário são cada vez mais diversas e muitas vezes confusas.

A operacionalização em questão é tratada na forma de caracterização de processos de uma empresa em que são atribuídos termos unificados de acordo com a dispersão dos processos e pela dispersão geográfica.

Dois termos principais são direcionados a distância geográfica:

• *Ofshore* se caracteriza pela equipe de projeto estar distante continental ou globalmente de seus clientes e usuários (AUDY; LOPES; MAJDENBAUM, 2002).

• *Onshore* é estar no mesmo país onde estão localizados o cliente e a matriz da empresa (AUDY; PRIKLDNICKI, 2008).

A partir dessas duas formas, surge uma unificação desses termos que geram modelos relacionados a empresas, tais formas podem ser definidas (AUDY; PRIKLDNICKI, 2008):

• *Onshore Insourcing*, um departamento dentro da própria empresa ou uma subsidiária da empresa no mesmo país que provê serviços de desenvolvimento de *software* através de projetos internos.

• *Onshore Outsourcing*, indica a contratação de uma empresa terceirizada (outsourcing), para o desenvolvimento de determinados serviços ou produtos de

software para a empresa.

• *Offshore Outsourcing*, indica a contratação de uma empresa terceirizada para o desenvolvimento de determinados serviços ou produtos de *software*, sendo que ela esta necessariamente localizada em um país diferente do contratante.

• *Offshore Insourcing*, indica a criação de uma subsidiária da própria empresa, que esta necessariamente localizada em um país diferente da matriz da empresa ou empresa contratante.

Contudo, torna-se necessário salientar que estas são as formas mais utilizadas pelas empresas, apesar de existir novos tipos e evoluções dos já existentes, porém devido à complexidade que segue as formas de operacionalização de um ambiente distribuído coube salientar somente as principais.

2.7.4 Desafios do DDS

Apesar do desafio do aumento de complexidade no desenvolvimento de software, o DDS apresenta alguns ainda maiores, como: distância física, fusos horários diferentes e as diferenças culturais. Além disso, esse ambiente apresenta grande impacto na forma de como os produtos são concebidos, desenvolvidos, testados e entregues aos clientes. A seguir são descritos os principais desafios encontrados pelo DDS (AUDY; PRIKLDNICKI, 2008):

Culturais

- Liderança: em um ambiente de DDS, a questão relacionada à liderança torna-se complexo pelo fato das contradições existentes em cada cultura, ou seja, enquanto alguns lideres tomam suas próprias decisões, seguindo uma linha hierárquica, outros incentivam a participação dos subordinados propondo situações para a tomada de decisões, tais fatores costumam fazer com que o processo decisório em equipes distribuídas seja mais demorado ou complexo KAROLAK (apud Lopes, 2004).
- Formação de grupos: a formação dos grupos podem gerar alguns problemas, visto que enquanto algumas culturas valorizam a independência e os direitos individuais, outras são coletivistas, subordinando os interesses do individuo para o bem do grupo HORVATH; MARQUARDT(apud LOPES, 2004). A

questão do idioma também é um causador de problemas, pois quando o idioma não é o nativo, acentua-se a falta de atenção, expressão e leitura lenta.

- Estilos de comunicação: a comunicação entre os membros do grupo pode variar de acordo com a cultura, para tanto existe dois tipos de estilos de comunicação pessoal, o expressivo que estabelece uma comunicação baseada em emoções, que são transmitidos por gestos como abraços e toques, e o instrumental onde a base para a comunicação esta na forma mais comum dela, a fala. Este estilo de comunicação é mais impessoal, centrado em problemas e orientado a objetivos (LOPES, 2004).
- Resolução de problemas: pode ser vista de forma diferente de acordo com a cultura, pode ser linear, onde os problemas são dissecados em pequenas partes ligadas em cadeias de causa e efeito, ou sistêmico, onde problemas são vistos de forma mais holística, com relacionamentos e caos como componentes comuns HORVATH; MARQUARDT (apud LOPES, 2004).
- Perspectiva de tempo: essa questão afeta o processo de desenvolvimento pelo fato das diferenças dos horários de trabalho, o que torna difícil a coordenação das equipes, isso se relaciona com questões físicas e psicológicas dos membros das equipes, tal situação resulta em uma dificuldade de comunicação entre elas.

Dispersão geográfica

- Gerenciamento: o gerenciamento de equipes distribuídas é um grande desafio, visto que existe uma grande dificuldade em relação ao gerenciamento de decisões, estas que são um fator determinante em um processo de desenvolvimento, para tanto é indispensável à utilização de técnicas de gerenciamento, estas que em sua maioria necessita de algumas adaptações para serem impostas em um ambiente de DDS. Apesar disso, algumas pesquisas indicam que os problemas de gerência de projetos em ambientes DDS não diferem significativamente dos problemas no desenvolvimento co-localizado (AUDY; PRIKLDNICKI, 2008).
- Fusos horários: outro desafio que implica principalmente na comunicação são os horários de trabalhos das equipes, tal questão faz a diferença tanto em

equipes globais, quanto em equipes co-localizadas, pois existem além dos fusos horários os diferentes turnos de trabalho. Isso se deve aos estados físicos e mentais dos participantes, pelo fato de que em um local podem estar iniciando o dia, enquanto outros estão no final do expediente (LOPES, 2004).

- Legislação: é uma questão que envolve uma serie de diferenças legais que são cuidadosamente estabelecidos em contratos que envolvem a área de propriedade intelectual principalmente quando se trata de DDS. A área de propriedade intelectual, envolvendo registro e comercialização de patentes, titularidade e direitos sobre *royalles*, registro de *software*. Licenciamento e direito de uso são tema em fase de estabilização no mundo todo, em especial nos países em desenvolvimento, como Brasil, Índia, Rússia e China (AUDY; PRIKLDNICKI, 2008).

Coordenação e controle

- Interdependência: coordenação diz respeito à união de tarefas e equipes para a realização de um determinado problema, e controle se refere aos fatores que contribuem coma a coordenação, que são controle das metas, politicas e padrões a serem seguidos. Para tanto o grau de dependência das tarefas exerce um papel fundamental na coordenação, por exemplo, quando dois integrantes de uma equipe, com tarefas fortemente relacionadas, colaboram, a diferença de fuso horário pode dificultar a coordenação (LOPES, 2004).
- Gerenciamento de configuração: fazer o controle das configurações tanto nos produtos quanto nas localidades em que se encontram os mesmos, além de toda uma documentação, é uma tarefa extremamente complicada principalmente em ambientes distribuídos.
- Consciência: ter a consciência do que esta sendo desenvolvido é crucial no DDS, pois em um processo de desenvolvimento feito em equipes é importante ressaltar que a colaboração de uns depende do que foi feito por outros, por isso a importância de documentação e abertura de código.

Comunicação

- Contexto: devido a distância a forma de comunicação trona-se um desafio, pois se sabe que esta é de fato essencial para equipes de desenvolvimento.

Para tanto, o contexto de comunicação pode variar de alto ou baixo, a depender da cultura. Em culturas de alto contexto possuem a habilidade de compartilhar experiências e tornar inteligíveis determinadas mensagens sem precisar declarar explicitamente, por outro lado, culturas de baixo contexto enfatizam a troca de fatos e informações, nessas culturas, vídeo conferência e e-mails são usualmente aceitos como substitutos eficientes para comunicações face a face (LOPES, 2004).

- Meio: as interações nos meios de comunicação também podem variar entre rica: reunião face a face, vídeo conferencia, etc.; e pobre: correspondência regular, fax, etc.

Alguns dos impactos da dispersão na comunicação, é a necessidade de excessiva comunicação, a falta de comprometimento e o desconforto ao utilizar alguns meios (AUDY; PRIKLDNICKI, 2008).

Espirito de equipe

- Coesão: a palavra equipe lembra uma serie de características, tais como, confiança, comunicação, poucos membros que trabalham em prol de um objetivo em comum, cooperação, enfim, esses fatores podem se tornar frágeis em um ambiente distribuído. Mesmo com a presença da coesão que segundo Lopes (2004) é uma característica fundamental para o sucesso das equipes, pelo fato de ter maior motivação, moral e produtividade, além de melhor comunicação e satisfação com o trabalho.
- Confiança: a obtenção de confiança entre as equipes é outro grande desafio encontrado no DDS, pois se sabe que ela é conquistada através da convivência face a face entre os membros da equipe. Como esse tipo de encontro não pode ser realizado devido a distancia, muitas organizações investem em viagens para integração das equipes (LOPES, 2004).
- Tamanho: é importante que se tenha um controle do número de membros de uma equipe, pois quanto maior for a equipe, maior o numero de canais de comunicação, e mais difícil fica de gerenciar principalmente quando se trata de um cenário distribuído.

2.8 Comparação dos Métodos Tradicionais e Ágeis de Desenvolvimento de *Software*

O Desenvolvimento Tradicional de *Software* possui como característica principal sua orientação a documentos, ou seja, todos os processos são documentados antes da implementação. Seu fluxo é em maioria unidirecional, ou seja, a fase atual é pré-requisito da fase seguinte. O desenvolvimento tradicional é indicado para projetos onde os requisitos são estáveis, o que é inviável atualmente, pelo fato das organizações possuírem ambientes dinâmicos (CHAVES, 2010).

Neste sentido, o Desenvolvimento Ágil é apontado como alternativa ao contexto de requisitos mutáveis. Na tabela 4, é possível observar um comparativo entre os métodos tradicionais e ágeis.

Tabela 4 - Comparação dos Métodos Tradicionais com os Ágeis

	Métodos Tradicionais	Métodos Ágeis
Pressupostos Fundamentais	Sistemas são totalmente especificáveis, previsíveis e podem ser construídos através de um planejamento minucioso e extensivo.	Softwares de elevada qualidade podem ser desenvolvidos por pequenos times usando princípios de melhoras contínuas e testes baseados em feedback rápido e mudanças.
Controle	Centrado no processo	Centrado nas pessoas
Estilo de Gerenciamento	Comando e controle	Liderança e colaboração
Gerenciamento de Conhecimento	Explícito	Tácito
Atribuição de Funções	Individuais - favorece a especialização	Times auto-organizáveis – encoraja troca de funções
Comunicação	Formal	Informal
Função dos Clientes	Importante	Crítica
Ciclo do Projeto	Guiado por tarefas e atividades	Guiados pela produção de funcionalidades
Modelo de Desenvolvimento	Modelo cíclico	Modelo de entrega evolutiva
Forma e Estrutura Organizacional Almejada	Mecânica (burocrática com grande formalidade)	Orgânica (flexível e com ações que encorajam a cooperação participativa)

Fonte: adaptada de MAHAPATRA; MANGALARAJ; NERUR (apud CHAVES, 2010).

3. ESTUDO DE CASO

O mercado de *software* busca processos mais produtivos com o objetivo de melhorar a qualidade, agilidade e flexibilidade durante o desenvolvimento do *software*. Sendo o DDS um termo abrangente, onde diversas situações são possíveis, cada uma com diferentes problemas que podem ser gerenciados, os benefícios aproveitados e as soluções aplicadas (AUDY *et al.*, 2003), e o *Scrum* formado por um conjunto de práticas de gestão que admite ajustes rápidos, acompanhamento, visibilidades constante e planos realistas (ANTUNES *et al.*, 2011,), propõe-se a utilização desses termos no desenvolvimento de jogos educacionais.

Este estudo de caso consiste no desenvolvimento dois jogos educacionais (Torneio *Math* e *Game Pet*), idealizados por um aluno de graduação da Universidade Federal do Piauí (UFPI). O processo de desenvolvimento teve a participação de uma equipe de alunos de graduação do Centro de Informática da Universidade Federal do Pernambuco (UFPE) e uma equipe de alunos de graduação da Universidade Federal do Piauí (UFPI). Com o objetivo de permitir a experiência de utilização do desenvolvimento ágil com o desenvolvimento distribuído de *software*.

A proposta segue algumas diretrizes utilizadas pelo *Scrum*, com algumas adaptações para o desenvolvimento distribuído de software. O desenvolvimento do projeto foi dividido em três fases: *PreGame*, *Game* e *PostGame*, o equivalente as práticas *Scrum* e ocorreu de forma distribuída, ou seja, os integrantes do projeto residiam em cidades diferentes, apesar de o *Scrum* enfatizar a comunicação de forma presencial.

3.1 Sobre o Jogo

O Torneio *Math* é um jogo de matemática em que o jogador escolhe um entre seis personagens para competir. Possui três níveis: fácil, médio e difícil, onde esses implicam no tempo em que o jogador tem para responder e na complexidade dos das questões matemáticas. A cada acerto o jogador marca 10 pontos e a cada erro o jogador perde 3 pontos. Em caso de empate a partida é repetida até que um dos competidores ganhe.

O *Game Pet* é um jogo onde o animal apresenta sintomas comuns e o usuário escolhe entre as opções uma correta para o tratamento, em caso de acerto o jogador passa para a próxima fase, se a opção for errada a fase é reiniciada.

3.2 Ferramentas Utilizadas no Desenvolvimento do Projeto

O Torneio *Math* e o *Game Pet* foram desenvolvidos utilizando a linguagem de programação Java com a ferramenta *NetBeans* IDE (*Integrated Development Environment*) 7.2.1, o *CorelDraw* e o *Blender* para o design gráfico.

Além dessas ferramentas de desenvolvimento dos jogos, foram utilizadas algumas ferramentas para a comunicação entre as equipes distribuídas. Sendo então utilizadas ferramentas de comunicação como *Skype* e e-mail, onde os membros descreviam as atividades realizadas, os problemas e o as próximas tarefas a serem realizadas.

3.5 Times

O projeto envolveu oito alunos de graduação residentes na cidade de Picos e dois alunos de graduação residentes em Recife, sendo esse projeto caracterizado como uma operacionalização *Onshore* de dispersão geográfica. Participaram também, um professor mestre e um aluno da UFPI que auxiliavam durante todo o processo.

As equipes foram divididas em duas. Cada equipe possuía um *Scrum Master*, responsável por gerenciar conflitos dentro de sua equipe, remover os impedimentos durante o processo e interagir com o *Product Owner* e com o Time auxiliar.

Baseado Modelo *Scrum*, foram definidos os papeis de cada participante do projeto que são apresentados na Figura 10.

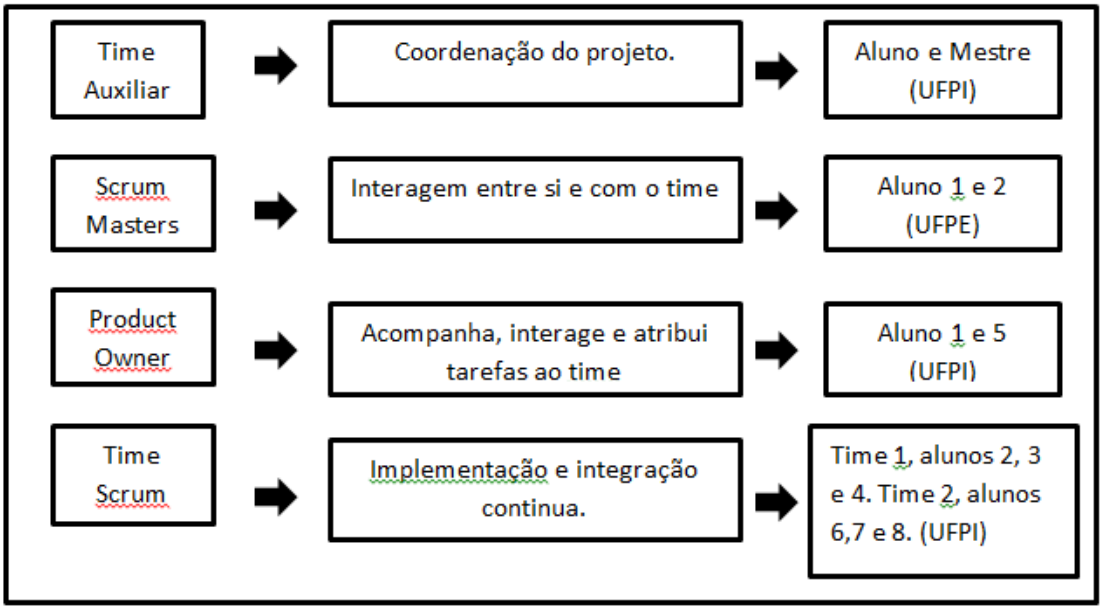

Figura 10 - Participantes do Projeto

3.6 Fases de Desenvolvimento

O projeto teve a duração de dois meses, sendo que a primeira fase durou uma *Sprint Backlog* de duas semanas o que correspondeu a confecção do *Product Backlog*. A segunda fase durou um *Sprint* de quatro semanas, nessa fase o jogo foi desenvolvido baseado nos requisitos do *Product Backlog*. A terceira fase durou uma Sprint de duas semanas, onde foram validadas as funcionalidades do jogo baseado nos requisitos do *Product Backlog*.

O projeto foi desenvolvido seguindo as práticas do Modelo *Scrum*, que mesmo sugerindo uma comunicação face a face, não foi possível pela dispersão dos membros do time e pelos horários incompatíveis entre os envolvidos.

O desenvolvimento do jogo foi dividido em três fases de acordo com a estrutura de funcionamento do processo *Scrum*: *PreGame, Game e PostGame*. Como mostra a Figura 11.

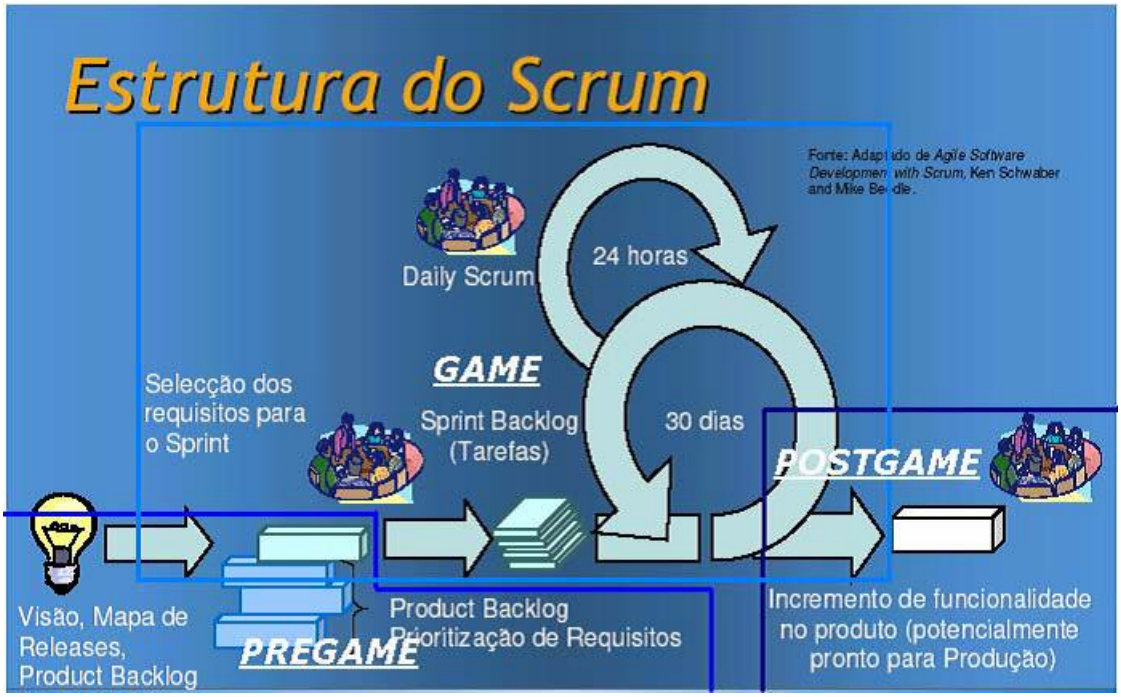

Figura 11 - Estrutura do Processo *Scrum* – Fases do Desenvolvimento

Fonte: (BLOMER, 2005, p.49).

Antes de iniciar as fases do desenvolvimento do jogo, foi realizado uma reunião presencial com o Time auxiliar, para definir os objetivos do projeto:

- Desenvolvimento de dois jogos educacionais;
- Desenvolver as aplicações baseado no modelo ágil *Scrum* de desenvolvimento de software;
- Desenvolver aas aplicações com dois times distribuídos;
- Fazer a descrição de desenvolvimento das aplicações.

Foram definidos também as atribuições de papeis dos participantes do projeto, como mencionado na Seção 3.4. Vale ressaltar que o time auxiliar teve participação ativa do decorrer de todo o projeto. A seguir serão apresentadas as fases de todo o desenvolvimento das aplicações

PreGame

Nesta fase o Time Auxiliar, os *Scrum Master* e os *Product Owner*, definiram alguns dos requisitos e os responsáveis pela execução, estes que foram adicionados ao *Product Backlog* e organizados de acordo com as prioridades.

Tabela 5: Lista de Requisitos do *Product Backlog*

Prioridade	Item	Descrição	Responsavél
Alta	1	Definir o escopo do jogo	Alunos 1 e 2 (UFPE) e Alunos 1 e 5 (UFPI)
Alta	2	Ferramentas de desenvolvimento do jogo	Time 1 e 2 (UFPI)
Média	3	Selecionar imagens atrativas ao usuário	Alunos 1 e 5 (UFPI)
Média	4	Não utilizar girias ou regionalismo em nenhuma atividade escrita	Todos

A tabela 5 mostra a lista do *Product Backlog* elaborada com as prioridades, dando inicio assim as informações sobre o escopo educativo dos jogos, visando que sua funcionalidade engloba a participação de pessoas de todas as idades. O tipo de ferramenta mais adequada para o desenvolvimento dos jogos onde essas podem utilizar imagens que sejam atrativas em um ambiente educativo, e que faz uso de uma escrita comum a jogos, podendo utilizar palavras em Inglês para a animação do jogo. Os requisitos dessa lista serão atendidos na fase *Game* de desenvolvimento.

A partir dessa primeira lista, as listas do *Product Backlog* foram confeccionadas de acordo com as necessidades do jogo, atendendo a prática de que os itens só podem ser inseridos nas reuniões.

Esse *Sprint* de confecção do *Product Backlog* teve uma duração de duas semanas, sendo que a prioridade do item 1 da lista de *Product Backlog* durou uma semana, e as prioridades dos itens 2,3 e 4 durou uma semana. O levantamento desses requisitos é necessário para iniciar o desenvolvimento do projeto.

Game

Nessa fase os time se reúnem através do *Sprint Planning Meeting*, onde foram adicionados novas funcionalidades ao *Product Backlog*, formando o primeiro *Sprint Backlog*, com regras dos jogos e modelos de interface. A *Sprint Planning* era realizada presencialmente entre o time da UFPI e através de uma ferramenta

síncrona (*Skype*) com os participantes da UFPE.

A partir dessas definições iniciou o primeiro *Sprint* que durou quatro semanas. Nesse período as *Daily Meeting* foram substituídas por reuniões semanais, onde os Times, os *Scrum Masters* e os *Product Owners*, juntamente com o Time auxiliar, apresentam os pontos positivos e negativos e as atividades desenvolvidas durante a semana.

Durante essa *Sprint* os Times *Scrum* programaram protótipos executáveis das interfaces, para que esses demostrassem assim às funcionalidades descritas no *Product Backlog.* As tarefas de programação duraram três semanas, visto que eram feitos testes de funcionamento a cada parte que constitui o jogo, bem como a aceitação pelos dos membros dos times, e a criação das interfaces duraram uma semanas. As figuras 12 e 13, exibem as principais telas dos jogos Torneio *Math* e *Game Pet*.

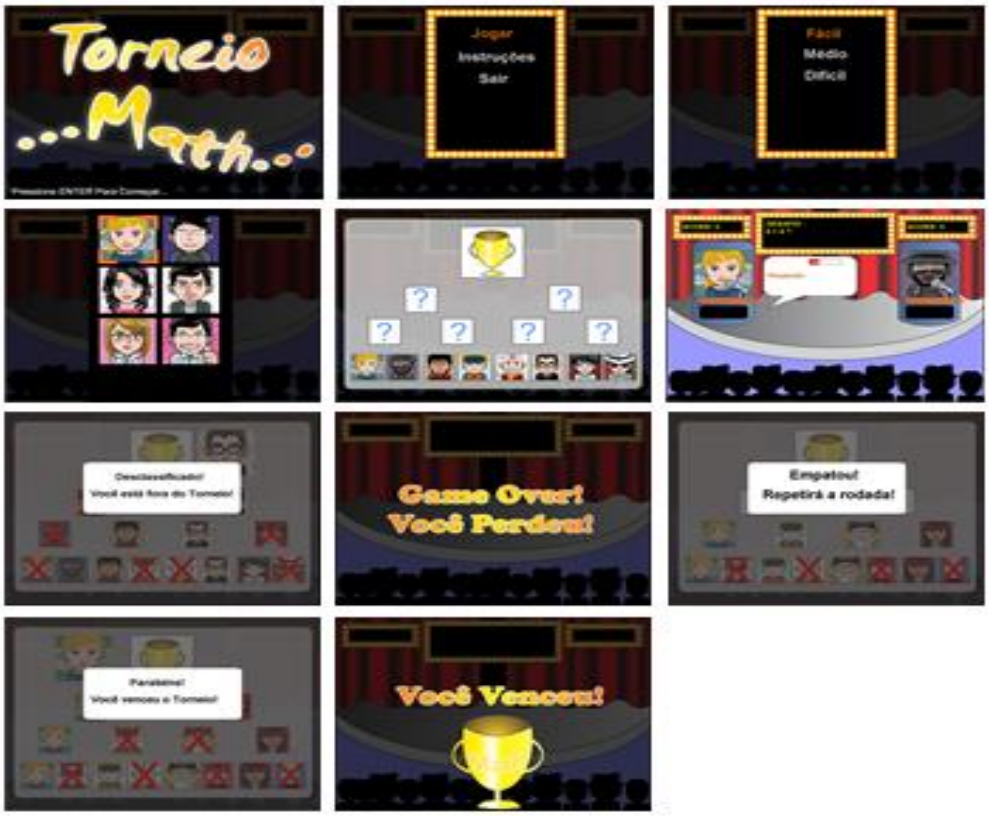

Figura 12 - Telas de Abertura, Escolha de Níveis, Escolha de Personagens e Competição do Jogo Torneio *Math*

Fonte: Time *Scrum*.

Figura 13 - Tela de Abertura, de perguntas, e dica de respostas do Game Pet.
Fonte: Time *Scrum*

Terminado a fase de desenvolvimento, a próxima etapa consistiu nos testes e validação dos jogos como um todo antes de denominar concluído o primeiro *Sprint*.

PostGame

Essa fase durou duas semanas, onde foi feita a revisão dos requisitos do *Product Backlog*, onde as funcionalidades desenvolvidas pelo Time *Scrum* eram testadas, para que pudesse identificar os defeitos, corrigi-los e depois valida-los, essas etapas eram feitas pelo *Product Owner*, pois ele se responsabilizava em avaliar as funcionalidades do produto. Esse por sua vez comunicava-se com o *Scrum Master* no objetivo de repassar as tarefas já executadas para que na reunião seguinte pudesse ser planejado o novo ciclo de desenvolvimento. Essa comunicação acontecia ao final de cada Sprint e se realizava de forma assíncrona

(e-mail).

Nesta fase, além da entrega do produto, seria entregue a documentação do mesmo, para tanto neste caso, não houve a necessidade pelo fato de ser um jogo simples, que foi desenvolvido somente para validar uma aplicação de desenvolvimento de dois jogos utilizando o modelo *Scrum* no contexto de DDS

3.7 Desafios e Soluções Encontrados

O maior desafio de desenvolvimento desse projeto foi o fato de os participantes de dos times, exceto o Mestre, terem sua primeira experiência de desenvolvimento de *software* seguindo um modelo de DS em um ambiente de DDS.

Durante todo o projeto houve algumas dificuldades nas equipes, e de uma forma geral, estava relacionadas à falta de compromisso com o projeto, tais como:

- O *Scrum* defende a centralização do time de desenvolvimento, ou seja, as iterações diárias devem ser presenciais. Nas experiências aqui relatadas houve uma dificuldade em encontrar um horário em comum para que houvesse essa iteração, porém, os times geograficamente distribuídos conseguiu superar esse desafio de comunicação através de ferramentas de chat e e-mail, mesmo que em horários diferentes entre os integrantes do time;
- A falta de entrosamento dos participantes gerou dificuldade para delegar tarefas para os participantes desconhecidos;
- A demora na execução de desenvolvimento e testes do processo. Visto que, o *Scrum* enfatiza alguns aspectos motivacionais ao time, tais aspectos podem corresponder a escolha de uma atividade pelo participante, o que pode ocasionar no atraso de uma *Sprint*;
- Outro ponto que ocasionou atraso no desenvolvimento, foi em relação a liderança no ambiente distribuído, visto que, o projeto atendia a uma linha hierárquica, sendo então o *Scrum Master* o responsável pela demora no retorno dos resultados;

O trabalho em grupo é complexo, pelo fato de cada participante possuir uma habilidade técnica e objetivos individuais.

As fases que mais apresentaram problemas foram as de desenvolvimento e testes, pelo próprio contexto de desenvolvimento de *software* e por ser uma tarefa

executada não somente pelo *Product Owner*, mas também pelo *Scrum Master*.

Porém, a utilização do modelo *Scrum*, contribuiu para amenizar alguns problemas encontrados:

- Um time pequeno aumentou a concordância de horários;
- A comunicação era feita também através de ferramentas síncronas e assíncronas (e-mail, Skype);
- Em relação a produtividade do projeto, obteve-se sucesso, pois não foi executado de forma vaga;
- Para manter o comprometimento dos times com as atividades, cada participante teve consciência do seu papel no projeto e o realizou de forma colaborativa, pois o trabalho de uns dependia do que era realizado por outros. Daí a importância das reuniões, mesmo que semanais;
- Durante a segunda fase foram feitos alguns testes de funcionalidade, visando diminuir o conjunto de erros na fase seguinte. Nessa fase de desenvolvimento também fazia parte os *Scrum Masters*.
- Os *Product Owners* eram responsáveis pela execução dos testes, para tanto a última fase *Scrum* foi delegada para que essa tarefa fosse executada com mais cautela.

4. RESULTADOS

O resultado final do projeto foi à aplicação de dois jogos educacionais, que foi desenvolvido seguindo o Modelo *Scrum.* Apesar de algumas situações de dificuldades devido ao cenário distribuído em que as aplicações eram desenvolvidas e pelo fato do trabalho em grupo de uma forma geral fomentar essas dificuldades, os participantes do projeto concluiu que os jogos foram desenvolvidos de forma satisfatória.

O projeto desenvolvido atendeu as expectativas iniciais idealizadas por um aluno e um mestre da UFPI, tendo como principal objetivo aplicar e descrever as experiências obtidas com o desenvolvimento de *softwares* de uma forma distribuída, mesmo que com equipes pequenas, características que enfatiza o Modelo *Scrum*.

Em alguns momentos fizeram-se necessárias algumas adaptações no contexto do Modelo *Scrum*, como a inserção de duas pessoas (aluno e mestre da UFPI) com a criação de um Time Auxiliar. Essa foi à forma encontrada para que a observação fosse feita constantemente durante o desenvolvimento do projeto, para que este fosse descrito.

O Time Auxiliar ditou as regras para o desenvolvimento do projeto, apesar de não fazer parte diretamente do desenvolvimento dos jogos, o Time Auxiliar foi fundamental para controlar o processo e os conflitos gerados.

As experiências vivenciadas durante o projeto, também resultaram em um maior conhecimento em relação ao desenvolvimento e gerenciamento de projetos de software, dando uma maior importância em seguir características de metodologias de desenvolvimento, para que os resultados sejam satisfatórios ao final do processo.

Durante o projeto notou-se que trabalhar com o Modelo *Scrum* proporciona uma maior dinâmica entre a equipe, isso ocasionado pela necessidade de uma continua comunicação, mesmo se tratando de um ambiente distribuído.

5. CONSIDERAÇÕES FINAIS

No mercado de software, a busca por ferramentas e métodos que possam auxiliar os processos de desenvolvimento tem aumentado significativamente, pois a maior preocupação dos desenvolvedores é melhorar a qualidade do produto e reduzir os prazos de desenvolvimentos.

Tendo em vista esses aspectos, o desenvolvimento de aplicações baseados nas formas tradicionais ou ágeis para o desenvolvimento de *software* é uma realidade onde equipes de projetos buscam criar novos processos com diferentes abordagens.

Este trabalho apresentou um levantamento bibliográfico da Engenharia de *Software* e suas formas de Desenvolvimento de *Software*, mostrando os Modelos Tradicionais de desenvolvimento e os principais Modelos Ágeis, em especial o *Scrum*, e o Desenvolvimento Distribuído de *Software*. Também foi apresentado um comparativo entre o desenvolvimento tradicional e ágil. A partir desse levantamento foi apresentada uma experiência de desenvolvimento passo-a-passo de dois jogos educacionais utilizando as características do Modelo *Scrum* em um projeto distribuído de *software*.

O objetivo do projeto foi alcançado, pois o processo de desenvolvimento do jogo foi realizado de acordo com a estrutura do processo *Scrum*, com sua divisão de papeis e suas fases.

Os trabalhos futuros consistem na realização de uma pesquisa maior sobre o desenvolvimento distribuído e sua aplicação em um ambiente global de *software*, fazendo uso dos Métodos ágeis, contribuindo para um estudo não somente do desenvolvimento de *software*, mas também em outros ramos do trabalho em equipe.

REFERÊNCIAS

ALBUQUERQUE *et al.* **ADOÇÃO DE *SCRUM* EM UMA FÁBRICA DE DESENVOLVIMENTO DISTRIBUÍDO DE *SOFTWARE***. 2007. 8f. Monografia (Bacharelado em Tecnologia da Informação) - Universidade Federal do Pernambuco, Recife, 2007.

ANTUNES *et al.* **ADOÇÃO DE METODOLOGIAS ÁGEIS PARA PRODUÇÃO DE JOGOS SOCIAIS COM TIMES DISTRIBUÍDOS**. 2011. 4f. Dissertação (Mestrado em Ciência da Computação) - Universidade Federal do Pernambuco, Recife, 2011.

ARAÚJO *et al.* **USO DOS JOGOS DIDÁTICOS EM SALA DE AULA**. 2007. 10f. Monografia (Bacharelado em Letras) - Universidade Luterana do Brasil – Campus Guaiba/RS, Guaiba - RS, 2007.

ARAUJO, L. B. P. **ESTUDO COMPARATIVO DA COMPATIBILIDADE ENTRE AS MELHORES PRÁTICAS DO PMI® E *SCRUM***. 2009. 89f. Monografia (Especialização em Gestão de Projetos) - FACULDADE DE INFORMÁTICA E ADMINISTRAÇÃO PAULISTA, São Paulo, 2009.

AUDY *et al.* **DESENVOLVIMENTO DISTRIBUÍDO DE *SOFTWARE*: UM MODELO DE CLASSIFICAÇÃO DOS NÍVEIS DE DISPERSÃO DOS *STAKEHOLDERS***. 2003. 10f. Dissertação (Mestrado em Ciências da Computação) - PUCRS; University of Illinois at Chicago, Porto Alegre - RS, 2003.

AUDY, J. L. N.; LOPES, L. T.; MAJDENBAUM, A. **UMA PROPOSTA PARA PROCESSO DE REQUISITOS EM AMBIENTES DE DESENVOLVIMENTO DISTRIBUÍDO DE *SOFTWARE***. 2002. 14f. Dissertação (Mestrado em Ciência da Computação) - Pontifícia Universidade Católica do Rio Grande do Sul, Rio Grande do Sul, 2002.

AUDY, J.; PRIKLDNICKI, R. **DESENVOLVIMENTO DISTRIBUÍDO DE *SOFTWARE***. 1. ed. Rio de Janeiro: Elsevier, 2008. 211p.

BARBAN, L. R. C. F.; BERTHOLDO, L. **ADAPTAÇÃO DO *SCRUM* AO MODELO INCREMENTAL**. 2010. 21f. Monografia (Bacharelado em Ciência da Computação) - Universidade Estadual de Campinas, Limeira, 2010

BICUDO *et al.* **PROJETO E DESENVOLVIMENTO DE JOGOS EDUCATIVOS EM 3 DIMENSÕES: A EXPERIÊNCIA DA UNIVAP VIRTUAL**. 2007. 15f. Monografia (Tecnologia em Tecnologia da Informação) - Universidade Vale do Paraíba, Vale do Paraíba, 2007.

BLOMER, V. **DESENVOLVIMENTO DE *SOFTWARE* EDUCACIONAL UTILIZANDO O MÉTODO ÁGIL *SCRUM* VISANDO A QUALIDADE DE REQUISITOS**. 2005. 72f. Monografia (Bacharelado em Informática) - UNIVERSIDADE DO PLANALTO CATARINENSE, Lages, 2005.

CARMEL, E. ***GLOBAL SOFTWARE TEAMS – COLLABORATING ACROSS***. EUA: Prentice Hall, 1999. 269p.

CHAVES, F. R. C. **MELHORANDO A USABILIDADE EM PROJETOS DE DESENVOLVIMENTO ÁGIL: UM ESTUDO DE CASO**. 2010. 63f. Monografia (Bacharelado em Ciência da Computação) - Universidade Federal de Pernambuco, Recife, 2010.

CUNHA *et al.* **O USO DE JOGOS ELETRÔNICOS NO PROCESSO EDUCACIONAL**. 2009. 6f. Monografia (Tecnologia em Ciência e Tecnologia) - Instituto Federal de Educação de Sergipe, IV Congresso de Pesquisa e Inovação da Rede Norte e Nordeste de Educação Tecnologica - Belém do Pará, 2009.

FILHO, W. P. P. **ENGENHARIA DE *SOFTWARE*: FUNDAMENTOS, MÉTODOS E PADRÕES**. 2. ed. Rio de Janeiro: LTC , 2003. 601p.

FILHO, N. A. S.; GOMES, J. L. **JOGOS: A IMPORTÂNCIA NO PROCESSO EDUCACIONAL**. 2008. 32f. Monografia (Especialização em Pedagogia) - Universidade Estadual de Ponta Grossa, Ponta Grossa - PR, 2008.

FREIRE, J. B. **O JOGO: ENTRE O RISO E O CHORO**. 2. ed. Campinas - São Paulo: Autores Associados, 2005. 132p.

HORVATH, L.; MARQUARDT, M. J. ***GLOBAL TEAMS: HOW TOP GLOBAL TEAMS: HOW TOP TEAMWORK***. Palo Alto, EUA: Davies-Black Publishing, 2001. 246p.

JÚNIOR, A. N. B. **O PROCESSO DE ENGENHARIA DE REQUISITOS EM AMBIENTES DE DESENVOLVIMENTO DISTRIBUÍDO DE *SOFTWARE***. 2011. 38f. Monografia (Especialização em Ciências da Computação) - UNIVERSIDADE FEDERAL DE PERNAMBUCO, Recife, 2011.

KAROLAK, D. W. ***GLOBAL SOFTWARE DEVELOPMENT - MANAGING VIRTUAL TEAMS AND ENVIRONMENTS***. Los Alamitos, EUA: IEEE Computer Society, 1998. 159p.

LIMA, M. C. F.; SILVA, M. E. L. E.; SILVA, V. V. S. **JOGOS EDUCATIVOS NO ÂMBITO EDUCACIONAL: UM ESTUDO SOBRE O USO DOS JOGOS NO PROJETO MAIS DA REDE MUNICIPAL DO RECIFE**. 2009. 25f. Monografia (Bacharelado em Pedagogia) - Universidade Federal do Pernambuco, Recife, 2009.

LOPES, L. T. **UM MODELO DE PROCESSO DE ENGENHARIA DE REQUISITOS PARA AMBIENTES DE DESENVOLVIMENTO DISTRIBUÍDO DE SOFTWARE**. 2004. 142f. Dissertação (Mestrado em CIÊNCIA DA COMPUTAÇÃO) - PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL, Porto Alegre, 2004.

MAHAPATRA, R.; MANGALARAJ, G.; NERUR, S. ***CHALLENGES OF MIGRATING TO AGILE METHODOLOGIES*.**. Communications of the ACM, EUA, v.48, n.5, p.6-32, 2005.

NETO, P. A. S. **INTRODUÇÃO À ENGENHARIA DE *SOFTWARE***. Piauí: Autoria Própria, 2007. 110p.

PRESSMAN, R. S. **ENGENHARIA DE *SOFTWARE***. 6. ed. São Paulo: McGraw-Hill, 2006. 711p.

PRIKLADNICKI, R. **MuNDDoS: UM MODELO DE REFERÊNCIA PARA DESENVOLVIMENTO DISTRIBUÍDO DE *SOFTWARE***. 2003. 144f. Dissertação (Mestrado em Ciências da Computação) - Faculdade de Informática da Pontifícia Universidade Católica do Rio Grande do Sul, PORTO ALEGRE, 2003.

SOMMERVILLE, I. **ENGENHARIA DE *SOFTWARE***. 9. ed. São Paulo: Pearson, 2011. 521p.

ZANATTA, A. L. ***XSCRUM*: UMA PROPOSTA DE EXTENSÃO DE UM MÉTODO ÁGIL PARA GERÊNCIA E DESENVOLVIMENTO DE REQUISITOS VISANDO ADEQUAÇÃO AO CMMI**. 2004. 180f. Monografia (Especialização em Ciência da Computação) - UNIVERSIDADE FEDERAL DE SANTA CATARINA, Florianópolis, 2004.