Universidade do Minho
Escola de Engenharia

Cristiana Sofia da Silva Lopes

Scrum para Ambientes de *Software* Distribuído: Análise Crítica e Estudo de Casos

Dissertação de Mestrado

Mestrado Integrado em Engenharia e Gestão de Sistemas de Informação

Trabalho efetuado sob a orientação de:

Professor Doutor Ricardo J.Machado

Outubro de 2014

DECLARAÇÃO

Nome: Cristiana Sofia da Silva Lopes

Endereço Eletrónico: crislopes@outlook.pt **Telefone:** +351 914861785

Nº. do Bilhete de Identidade: 13562303

Título da Dissertação de Mestrado: *Scrum* para ambientes de *software* distribuído: análise crítica e estudo de casos

Orientador: Professor Doutor Ricardo J. Machado

Ano de conclusão: 2014

Designação do Mestrado: Mestrado em Engenharia e Gestão de Sistemas de Informação

Nos exemplares das teses de doutoramento ou de mestrado ou de outros trabalhos entregues para prestação de provas públicas nas universidades ou outros estabelecimentos de ensino, e dos quais é obrigatoriamente enviado um exemplar para depósito legal na Biblioteca Nacional e, pelo menos outro para a biblioteca da universidade respectiva, deve constar uma das seguintes declarações:

1. É AUTORIZADA A REPRODUÇÃO INTEGRAL DESTA TESE/TRABALHO APENAS PARA EFEITOS DE INVESTIGAÇÃO, MEDIANTE DECLARAÇÃO ESCRITA DO INTERESSADO, QUE A TAL SE COMPROMETE;

2. É AUTORIZADA A REPRODUÇÃO PARCIAL DESTA TESE/TRABALHO (indicar, caso tal seja necessário, n'' máximo de páginas, ilustrações, gráficos, etc), APENAS PARA EFEITOS DE INVESTIGAÇÃO, MEDIANTE DECLARAÇÃO ESCRITA DO INTERESSADO, QUE A TAL SE COMPROMETE;

3. DE ACORDO COM A LEGISLAÇÃO EM VIGOR, NÃO É PERMITIDA A REPRODUÇÃO DE QUALQUER PARTE DESTA TESE/TRABALHO

Universidade do Minho, 31/10/2014

Assinatura:_____

Mais uma etapa difícil foi concretizada com muito esforço e dedicação. Contudo, este resultado só foi conseguido graças a algumas pessoas que estiveram sempre ao meu lado mesmo nos momentos mais complicados. Agradeço:

Aos meus pais, pelo esforço que toda a vida fizeram para me proporcionarem uma boa educação na introdução dos melhores valores.

À minha família pela motivação, carinho e apoio incondicional, nesta fase e durante toda a minha vida.

Aos meus amigos, pelo apoio e pelos momentos de desconcentração. Pelo facto de me acompanharem sempre de forma positiva em todos as minhas fases da vida académica.

Agradeço ao Dr. Prof. Ricardo Machado, pela motivação, pela disponibilidade e apoio durante a orientação deste trabalho de dissertação.

Agradeço à Engª. Ana Lima, pelo apoio e disponibilidade prestada durante a elaboração deste trabalho de dissertação.

RESUMO

A abordagem para o desenvolvimento de *software* tem despertado grande interesse entre as organizações de todo o mundo, uma vez que apresentam características de uma abordagem simples e flexível num ambiente em constante mudança.

Quando o assunto é o desenvolvimento de *software,* pensamos logo nos modelos que podem ser utilizados, a fim de que o processo de construção seja o mais eficiente possível.

Diversas metodologias estão a ser utilizadas no desenvolvimento de *software* com o propósito de satisfazer o cliente da melhor forma possível. Existem, no entanto, vários riscos associados a este desenvolvimento, destacando-se a possibilidade do produto não cumprir os requisitos pretendidos.

Para combater este risco, umas das soluções é a utilização das metodologias ágeis, que são uma alternativa às metodologias tradicionais.

Este trabalho, tem por objetivo explicar o que são as metodologias ágeis, através das suas práticas, princípios e métodos, sendo assim, a metodologia *Scrum* assume-se extremamente ágil e flexível.

Da mesma forma, relata-se aqui experiência de adaptar as práticas ágeis, do *scrum*, no contexto de um projeto distribuído de *software*, que envolve uma equipa

Palavras-chave: Desenvolvimento de *Software*, Métodos ágeis, *Scrum*, Desenvolvimento Distribuído de *Software*

ABSTRACT

The approach to software development has given rise to a great interest among several organizations around the world, since it features a simple and flexible approach in a changing environment.

As far as software development is concerned, we normally think about which models can be used in order to make the process the most effective we can get.

Several methodologies are being used in software development, in order to satisfy the customer in the best possible way. There are several risks associated with this development. One of them is that the product may not meet the desired requirements.

To fight this risk, one of the solutions is to use agile methodologies that are alternative to traditional methodologies.

The purpose of this paper is to explain what agile methodologies are, such as their practices, principles and methods. The Scrum is assumed as an agile and extremely flexible methodology.

It is also about an experience of adapting scrum agile practices in a distributed software project that involves a team.

Key-Words: Software Development, Agile Methods, Scrum, distributed software development

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABELAS

SIGLAS

Neste documento, encontram-se acrónimos alusivos às áreas de conhecimento em estudo que, pela elevada frequência com que aparecem, ao longo dos textos, sugerem a sua utilização. Seguidamente, apresenta-se a lista de siglas:

RUP - Rational Unified Process

AUP - *Agile Unified Process*

CCG - Centro de Computação Gráfica

PMEs. - Pequenas e Médias Empresas

EPMQ - *Engineering Process Maturity & Quality*

UMC - Urban and Mobile Company

I&DT - Investigação e Desenvolvimento tecnológico

DDS - Desenvolvimento Distribuído de *Software*

CAPÍTULO 1- INTRODUÇÃO

1.1 Contextualização

Desenvolvimento é, definido como sendo, a ação ou o efeito de desenvolver (algo) ou de se desenvolver. Porém, é necessário explorar o significado do verbo "desenvolver", em que este trata de acrescentar ou de melhorar algo, podendo ser de ordem física, intelectual ou moral.

O desenvolvimento de *software* é uma atividade intensamente humana, em que a preocupação com as pessoas, o facto de o produto ser o objetivo alcançado, o processo prever as linhas gerais para a criação de um plano para o desenvolvimento de *software* e o projeto ser o empreendimento realizado para atingir os objetivos. Se não existir uma preocupação com esses aspetos, dificilmente o projeto obterá sucesso.

Todavia, as empresas de desenvolvimento de *software* encontram dificuldades em responder às dificuldades encontradas na utilização de métodos de desenvolvimento tradicionais, problemas como: a extrapolação dos prazos da entrega de *software*, custos acima do previsto e uma insatisfação elevada do cliente. Estes problemas, enfrentados pelas empresas com as metodologias, motivaram a adoção de metodologias de desenvolvimento mais flexíveis, como as metodologias ágeis, de modo a promover uma rapidez e qualidade maior no *software* produzido (Boehm,2006).

Em paralelo ao crescimento dos métodos ágeis, o desenvolvimento distribuído de *software* tem conseguido cativar cada vez mais adeptos motivados pelos problemas organizacionais enfrentados pelas empresas e pelas vantagens que este tipo de desenvolvimento traz. Como vantagem, o desenvolvimento distribuído de *software* possibilita o aumento na velocidade de desenvolvimento através de equipas distribuídas que trabalham em paralelo e em diferentes fusos horários (Herlsleb and Moita, 2001).

Devido ao aumento da adoção destas duas formas de desenvolvimento e a sua procura elevada, as empresas foram motivadas a utilizar metodologias ágeis no desenvolvimento de *software* em equipas distribuídas. Através desta junção, espera-se reduzir mais o tempo de desenvolvimento do *software*.

O novo ambiente em que vivemos, cria novos e grandes desafios para o processo de desenvolvimento de *software,* que se torna cada vez mais distribuído e global, por isso, a motivação

deste trabalho surgiu da necessidade de adaptar o método *Scrum* a projetos que utilizam o desenvolvimento distribuído de software.

Neste trabalho de investigação pretende-se, numa primeira faz, analisar as metodologias tradicionais e ágeis com o objetivo de justificar a escolha das abordagens ágeis. Também será efetuado uma análise ao *Scrum*, com o objetivo de estudar as suas vertentes e explicar o porquê da sua utilização neste trabalho de dissertação.

Numa segunda fase, será analisado o desenvolvimento distribuído de *software* e, através desta análise profunda, será sistematizado um conjunto de características e de práticas correspondentes a este tipo de desenvolvimento.

Para uma análise critica mais fundamentada, também serão demonstrados alguns problemas que surgem neste tipo de desenvolvimento.

Por conseguinte, através da informação retida no levantamento de informação sobre o desenvolvimento distribuído, pretende-se analisar a junção do *Scrum* com este tipo de desenvolvimento através da adaptação das características do desenvolvimento distribuído de *software* com as práticas e papéis do *Scrum.* Ou seja, esta dissertação tem como objetivo analisar o *Scrum* em ambientes de *software* distribuídos através de uma análise critica e no estudo de dois casos.

1.2 Objetivos

Este subcapítulo, tem como função apresentar os objetivos que se pretende alcançar, com a realização deste trabalho de dissertação-

Esta tem como objetivo principal a análise do *Scrum* em ambientes de desenvolvimento distribuídos de *software*, através do estudo de dois casos do CCG e através de uma análise crítica.

Por conseguinte, os objetivos são os seguintes:

1. Sistematizar as características gerais das metodologias tradicionais e ágeis no desenvolvimento de *software* para justificar a utilização de métodos ágeis neste trabalho de dissertação.

2. Analisar de forma cuidada, o *Scrum* em todas as suas vertentes, na identificação das suas práticas, dos seus benefícios, atributos e nos fatores de qualidade aplicados a este método.

3. Propor um conjunto de características, que permite entender melhor o desenvolvimento distribuído de *software,* assim como os problemas que, com este tipo de desenvolvimento, podem surgir.

4. Sistematizar as características dos dois projetos desenvolvidos no laboratório EPMQ do CCG.

5. Criar recomendações básicas que possam auxiliar as tarefas de gestão de projetos no *Scrum* para ambientes distribuídos.

1.3 Abordagem Metodológica

Nos projetos de investigação é necessário utilizar uma abordagem metodológica adequada para a concretização dos objetivos definidos, para este trabalho de dissertação.

Hegenberg (1976) indica que, a metodologia é o estudo de métodos científicos e que o seu objetivo é o aperfeiçoamento dos procedimentos e critérios utilizados na pesquisa. Por sua vez, método é o caminho para chegar a um determinado fim ou objetivo.

As abordagens metodológicas são organizadas a partir de diversas propostas de classificação. Do conjunto de abordagens existentes, as escolhidas para este trabalho são o *Design Science Research* e a *Action Research*

O primeiro refere-se a um processo de investigação para a criação de um artefacto, com o objetivo de garantir disciplina, rigor e transparência à investigação, para que o conhecimento obtido seja também científico. (Isabel Ferreira *et al.,* 2012).

Para Vaishnavi e Kuechler (2009), a *Design Science Research* é um conjunto de técnicas analíticas que permitem o desenvolvimento de pesquisas nas diversas áreas, em particular na engenharia.

Por sua vez, tem como objetivo estudar, pesquisar e investigar o seu comportamento, tanto do ponto de vista académico como da organização (Bayazit, 2006)

Ao utilizar esta metodologia é importante entender o seu significado, a sua organização e os seus possíveis resultados e como se pode fazer investigação de alta qualidade em sistemas de informação.

Este tipo de sistemas são uma atividade de *design* onde os resultados são artefactos, e que visam responder à melhoria de condições do mundo real

Relativamente à organização do *design science research*, a metodologia é constituída por um conjunto de cinco fases sendo elas as seguintes:

A primeira fase diz respeito à **Consciencialização do problema,** que diz respeito à compreensão e definição do problema e o seu resultado final é a definição e a formalização do problema a ser resolvido. O output, desta fase, é a proposta para um novo esforço de investigação (Vaishnavi & Jr., 2008).

A segunda fase é referente à **Sugestão.** Esta etapa, segundo *Manson* (2006) está ligada às atividades de desenvolver artefatos para a solução do problema. Por isso, o resultado da sugestão é um conjunto de artefatos a desenvolver.

Esta fase é um processo criativo em que, diferentes investigadores podem criar diferentes teorias para um conjunto de observações iguais.

A terceira fase**,** o **Desenvolvimento,** consiste no desenvolvimento e implementação do processo de construção dos artefactos. Estes são objetos artificiais que podem ser caraterizados em termos de objetivos, funções e adaptações. (*Simon*, 1996).Existem quatro tipos de artefactos os constructos, os modelos, os métodos e as instanciações. O principal resultado do desenvolvimento é o artefacto em estado funcional.

A quarta fase é a **Avaliação,** que diz respeito a um processo rigoroso de verificação do comportamento do artefacto, no ambiente para qual foi criado.

Por fim, a quinta fase é a **Conclusão** sendo a fase final de *Design Science Research*, consiste na formalização geral do processo. Normalmente diz respeito aos resultados obtidos, assim como aos conhecimentos adquiridos ao longo do esforço de investigação.

Para os objetivos desta dissertação serem cumpridos, serão adotadas as cincos fases desta metodologia que servirá de guia para a resolução do problema apresentado.

Este trabalho pretende analisar um problema real que ocorre durante o desenvolvimento das soluções.

Por sua vez, apresenta como principal problema as constantes mudanças nos projetos e o facto de a maioria dos artefactos se relacionarem e se afetarem mutuamente. A sugestão apresentada, para resolver o problema mencionado consiste em efetuar uma análise crítica e o estudo de casos relativamente ao *Scrum* em ambientes distribuídos.

O facto da criação de artefatos se basear em teorias existentes já aplicadas faz com que a concretização dos objetivos deste trabalho sejam muito importantes, ou seja a concretização do segundo objetivo do mesmo, seja bastante importante, uma vez que é através deste que é reunido grande parte do conhecimento necessário para criar uma análise crítica pretendida neste estudo.

Para minimizar os riscos, diversas pesquisas são efetuadas, sendo elas realizadas através de artigos científicos, teses de dissertação e livros. Contudo, ao longo da pesquisa, existiu sempre a necessidade de verificar a credibilidade dos documentos, verificando o número de citações e o local onde foram publicados.

Para a concretização do segundo objetivo, deste trabalho de dissertação, foi efetuada uma sistematização de todas as vertentes do *Scrum*, com base nas diversas análises efetuadas ao mesmo.

Com os resultados obtidos neste objetivo, procede-se à proposta de um conjunto de características, referentes ao desenvolvimento distribuído de *software,* que permite entender melhor este tipo de desenvolvimento em que são criadas recomendações que possam auxiliar as tarefas de gestão de projetos no *Scrum* e no desenvolvimento distribuído de software, realizando assim o terceiro e o quarto objetivo desta dissertação.

Por conseguinte, os resultados obtidos com a concretização dos referidos objetivos, serão discutidas com as seguintes conclusões, de forma a demonstrar que é importante possuir conhecimento relativamente aos diversos elementos que constituem o *Scrum* assim como a análise das características e práticas proposta para o desenvolvimento distribuído de *software*.

A *Action Research* consiste no desenvolvimento de uma colaboração, entre o investigador e os elementos da organização designados, no sentido de diagnosticar e resolver um problema de especial relevância para a organização (Bryman & Bell, 2007; Eden & Huxham, 1996). A *Action Research* é uma metodologia qualitativa multifacetada, existindo um debate considerável entre académicos sobre a sua natureza (Bryman & Bell, 2007).

A metodologia adotada neste estudo é a descrita em Eden & Huxham (1996), que refere as seguintes caraterísticas adicionais:

(a) Utilização de um processo iterativo de identificação do problema, planeamento, ação e avaliação, que conduz à emergência de novos avanços teóricos em pequenos passos incrementais;

(b) Os resultados da investigação devem ter implicações teóricas, que vão para além da resolução do problema concreto da organização;

(c) A investigação deve conduzir a generalizações, que possam ser expressas através de ferramentas, técnicas, modelos e métodos aplicáveis noutras situações.

A *Action Research* é indicada para indivíduos que escolhem um problema da organização onde se encontram inseridos como foco do seu estudo.

Usualmente, a investigação é conduzida numa única organização e, por esse motivo, muitos dos procedimentos utilizados são semelhantes ao estudo de caso. A recolha de dados pode basear-se em documentação sobre o problema, questionários e entrevistas (Bryman & Bell, 2007).

Neste trabalho de dissertação, as principais fontes utilizadas foram conversas informais com um leque alargado de colaboradores da empresa.

1.4 Estrutura do Documento

De forma a atingir todos os objetivos definidos neste capítulo, este trabalho será dividido em cinco capítulos.

No primeiro, é apresentada uma breve contextualização sobre o problema que motiva a realização deste estudo, assim como a definição dos objetivos que se pretendem alcançar, com este trabalho de dissertação, a abordagem metodológica adotada e a organização do documento.

O segundo capítulo é referente à revisão da literatura, onde são introduzidos os conceitos que serão abordados.

Do mesmo modo, serão expostos os conceitos relacionados com o *Scrum* e a realização de uma análise critica

No terceiro capítulo, inicialmente é efetuado a análise do desenvolvimento distribuído de *software*. Em seguida, é apresentado um conjunto de características e práticas deste desenvolvimento que depois serão adaptadas ao *Scrum*.

No quarto capítulo, é realizado a análise de dois casos e feita uma comparação entre ambos para analisar a viabilidade do *Scrum* em ambiente de *software* distribuído.

Por fim, o último capítulo refere-se à conclusão deste trabalho e às tarefas futuras, capazes de enriquecer o processo criado no âmbito deste trabalho de dissertação

CAPÍTULO 2 - SINOPSE DO *SCRUM*

2.1 Introdução

Ao longo dos anos, cada vez mais se percebe que o desenvolvimento de *software* nunca foi uma tarefa simples.

Segundo Pressman (2001), o processo de *software* é um conjunto de tarefas solicitadas para a produção de um *software* com alta qualidade. O resultado deste é um produto (*software*) que reflete a forma como o mesmo é realizado.

A qualidade final, do produto criado, são métodos propostos que procuram padronizar as atividades do ciclo de vida de um processo de *software*.

A metodologia do desenvolvimento de *software* é a representação simplificada do processo (Sommerville, 2004).É um conjunto de práticas recomendadas para o seu desenvolvimento, em que a primeiro é a construção de sistemas da forma mais eficaz e eficiente possível e que satisfaça às necessidades dos seus utilizadores (Ambler, 2005).

A fase de desenvolvimento do *software* tem como propósito construir o produto (*software*) e é constituída pelo *design*, mesmo, tanto ao nível do utilizador (funcionalidades e interface gráfica), quanto ao do desenvolvedor (arquitetura de software). A implementação é a etapa da codificação do produto. A verificação do *software* é a etapa que garante que o mesmo esteja a funcionar corretamente como foi proposto. Por fim, a validação visa garantir que o *software* esteja a realizar corretamente o que foi definido na especificação (Sommerville, 2007).

A fase de operação do *software* é responsável pela distribuição e entrega do produto ao cliente, pela instalação configuração e utilização do software e a manutenção do mesmo (Sommerville, 2007).

Como suporte a esta atividade, a engenharia de *software* oferece metodologias que auxiliam os desenvolvedores durante o processo de desenvolvimento, das quais, as metodologias tradicionais e as metodologias ágeis.

Na década de 70,a atividade do desenvolvimento de *software* era executada de uma forma desorganizada e sem planeamento, o que resultava num produto com péssima qualidade e que não correspondia às necessidades reais do cliente (Pressman, 2001).

Devido a esta situação, surge a necessidade de optimizar o desenvolvimento num processo mais estruturado e planeado através do uso de metodologias tradicionais.

A metodologia tradicional é muito utilizada nos dias de hoje, principalmente o seu modelo clássico, também conhecido pelo modelo em "cascata". Este, foi o primeiro processo publicado do desenvolvimento de *software* (Pressman*,* 2001). É um modelo que tem uma sequência de etapas. Cada etapa tem associada ao seu término uma documentação que tem deve ser aprovada para que se inicie a etapa posterior.

Por conseguinte, este modelo apresenta um problema relativamente à divisão do projeto em fases distintas, o que dificulta as possíveis alterações dos requisitos por parte do cliente. Assim sendo, este modelo é apropriado apenas, quando os requisitos são bem compreendidos e estruturados (Sommerville,2004).

Apesar de ser o mais usado não é o mais eficaz, pois são raros os projetos de *software* que utilizam este fluxo linear. Os processos tradicionais são baseados numa produção com uma elevada quantidade de documentação e de modelos, para seguir durante a programação, em que o tempo de desenvolvimento é elevado o que dificulta o controlo do projeto.

Na década de 90, surge um novo método que sugere uma abordagem de desenvolvimento ágil, em que os processos adotados tentaram adaptar-se às mudanças. A constante comunicação e negociação com o cliente deixou de ser vista como algo prejudicial, passando a ser vista como uma oportunidade para melhorar o produto final, utilizando os mesmos recursos, aumentando assim a eficácia na criação dos produtos de *software.*

O termo metodologias ágeis tornou-se popular quando um grupo de 17 pessoas ligadas à indústria de *software* reuniram, em Fevereiro de 2001, nos Estados Unidos, para discutirem os chamados processos leves. Foi então criada a Aliança Ágil e estabelecido o Manifesto Ágil (Agile Manifesto, 2004).

O manifesto ágil, baseia-se nos seguintes valores fundamentais (Fowler,2001):

- Indivíduos e interações, ao invés de processos e ferramentas.
- *Software* executável, ao invés de documentação
- Colaboração do cliente, ao invés da negociação de contratos.
- Respostas rápidas à mudança, ao invés de seguir planos.

Segundo Ambler (2004), "o interessante dessas declarações de valores é que todos irão concordar com elas imediatamente, mas poucos irão aderir a elas na prática, uma vez que as pessoas dizem uma coisa e fazem outra".

Todas estas metodologias baseiam-se nos princípios descritos pelo manifesto, mas cada um tem uma visão diferente na sua aplicação.Com o objetivo de ajudar na compreensão do que é o desenvolvimento ágil, os membros da Aliança Ágil refinaram as ideias do Manifesto Ágil em doze princípios citados abaixo nas palavras de Pressman, (2011):

1- A principal prioridade é satisfazer o cliente através da entrega adiantada e contínua do *software* requerido.

2 - Tolerância a mudanças de requisitos repentinos, mesmo no fim do processo. Um processo ágil para que tenhamos uma vantagem competitiva em relação aos concorrentes.

3 - Entrega de várias versões do *software* a funcionar e com frequência. Tentar encurtar o prazo de entrega.

4 - A principal medida de progresso é o *software* estar a funcionar.

5 - Os clientes e a equipa de desenvolvimento do *software* devem trabalhar juntos, deve existir uma cooperação entre os mesmos.

6 - Desenvolver os projetos em torno de indivíduos motivados. É necessário que eles tenham um ambiente propício para o desenvolvimento e que sintam que existe confiança por parte de outrem.

7 - A forma mais eficiente e eficaz de se transmitir qualquer tipo de informação é uma conversa cara a cara entre os indivíduos da equipa.

8 - Um processo ágil move o desenvolvimento sustentável. Todos os trabalhadores devem manter um ritmo constante de trabalho.

9 - Atenção redobrada em relação à técnica e a um bom *design*.

10 - Simplicidade, ou seja, a arte de maximizar a quantidade de trabalho que não foi preciso fazer, é essencial.

11 - Auto-organizar as equipas, pois é daí que saem as melhores arquiteturas, requisitos e projetos.

12 - Adaptar-se à evolução das circunstâncias. Ajustar o comportamento a ideias que possam tornar o desenvolvimento do projeto mais eficaz.

Os objetivos fundamentais destes princípios são: promover uma melhor compreensão do que são métodos ágeis, orientar as equipas e verificar se eles estão realmente a utilizar um método ágil. Eles oferecem uma base, para o sucesso no trabalho do desenvolvimento do *software* e para, além disso, definem requisitos de alto nível para uma metodologia eficaz (Ambler, 2005).

Cockburn (2000) define desenvolvimento ágil de *software* como uma abordagem de desenvolvimento, que trata os problemas que surgem com mudanças rápidas. Por isso, é importante entender que as metodologias ágeis são uma atitude e não um processo descritivo. É uma forma efetiva de se trabalhar em conjunto, para atingir as necessidades das partes interessadas do projeto. Reúne boas práticas de programação e de gestão de projetos com o objetivo de produzir um *software* com qualidade. Propondo assim uma alternativa que procura a melhoria do processo tornando-o mais ágil e menos burocrático.

Contudo, por trás dos métodos ágeis, existem duas ideias principais que diferenciam as metodologias ágeis das metodologias tradicionais sendo elas as seguintes (Fowler 2006):

Os métodos ágeis são mais adaptativos do que prescritivos, eles adaptam-se à mudança a qualquer momento, ao ponto de adaptar a sua própria metodologia para serem bem-sucedidos, enquanto que, os métodos tradicionais procuram planear em detalhes e faz parte da sua natureza a resistência à mudança, uma vez que o planeamento esta estabelecido.

Os métodos ágeis são orientados a pessoas ao invés dos processos. Já, os métodos tradicionais procuram definir processos que vão funcionar com qualquer um que os vai executar, os métodos ágeis asseguram que nenhum processo pode superar as habilidades de uma equipa. Desta forma, o papel do processo é dar suporte à equipa de desenvolvimento no seu trabalho.

Nos dias de hoje, a literatura demonstra que diversas experiências e casos práticos levaram ao surgimento de algumas metodologias ágeis de desenvolvimento de *software*::

eXtremme Programming (Beck, 1999), *Agile Unified Process* (Ambler, 2005), *Crystal Methodologies* (Cockburn, 2000); *Dynamic System Development Model* (DSDMC, 1997) e o *Scrum* (Sutherland (1990); Swhwaber (1995)).

Destas várias técnicas, destaca-se o Scrum pela rápida aceitação que teve na indústria de produção de *software*, devido á sua fácil compreensão e utilização.

Segundo *7th Annual State of Agile Development Survey* (Versionne,2012), que trata-se de uma pesquisa realizada em dezembro de 2012 com 4048 profissionais de desenvolvimento de *software* sobre o uso de metodologias ágeis, em que 60% dos entrevistados foram da América do Norte, 27% da Europa e 13% do resto do mundo, em que o tamanho medio das organizações entrevistadas foi de 100 colaboradores.

Esta pesquisa mostra que *Scrum* é a metodologia mais usada com 52% dos participantes e os restantes 48% ficaram divididos entre as 12 outras metodologias, como se pode verificar na figura 1.

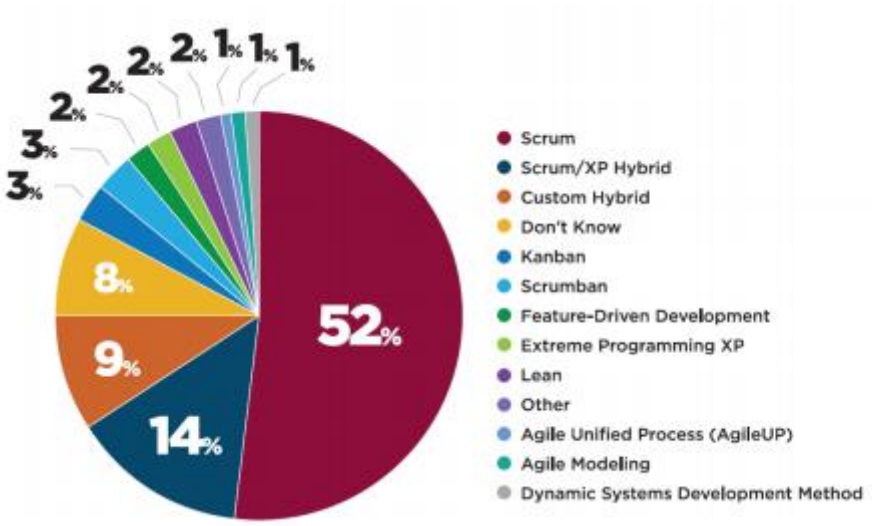


Figura 1– Metodologias ágeis utilizadas, (Fonte: Versionone, 2012)

A realização desta pesquisa permitiu verificar os resultados positivos da aplicação de metodologias ágeis. A importância dada pelas metodologias ágeis às pessoas envolvidas nos projetos já esta clara nos seus valores constantes no manifesto ágil, no qual se destaca o primeiro destes que cita que metodologias ágeis valorizam "indivíduos e iteração entre eles mais que processos e ferramentas (Beck et al., 2001).

Com este valor como principal guia da metodologia *Scrum*, este método destaca-se pelas suas caraterísticas de procurar manter a visão na gestão do projeto e não em processos e

controlos. Por outras palavras, ao aplicar *Scrum* será possível descobrir o que há de errado nos processos e o que pode ser feito para corrigir esses problemas. (Schwaber & Sutherland, 2011)

Por conseguinte, este capítulo vai ter como objetivo demonstrar todas as vertentes do *Scrum*, assim como, uma análise critica na conclusão para a explicação dos restantes capítulos deste trabalho de dissertação.

2.2 O *Scrum*

Mike Cohn definiu o *Scrum* em 100 palavras:

> *Scrum* é um processo ágil que permite manter o foco na entrega do maior valor do negócio, em menor tempo possível.
>
> Isto permite a rápida e contínua inspeção do *software* em produção (em intervalos de duas a quatro semanas).
>
> As necessidades do negócio é que determinam as prioridades do desenvolvimento de um sistema. As equipas auto-organizam se para definir a melhor maneira de entregar as funcionalidades com maior prioridade.
>
> Entre cada duas a quatro semanas todos podem ver o *software* real em produção, decidindo-se o mesmo deve ser entregue ou continuar a ser aperfeiçoado por mais um "Sprint". (*Scrum Training & Agile Training from Scrum Master Mike Cohn*, 2009)

Origem: É uma abordagem simples criada por Jeff Sutherland, a partir do trabalho de Nonaka e Takeuchi, pela sua equipa no início de 1990 e formalizada por Ken Schwaber em 1995. (Schwaber,1997; Sutherland, 2004).

A palavra *Scrum* surgiu de uma atividade que ocorre durante uma partida de *rugby* em que a bola é colocada no jogo e toda a equipa reúne esforços para alcançar o objetivo: a posse da bola. Tal como no *rugby*, o *Scrum* depende de um trabalho de equipa constante, disponibilidade e entrega total para conseguir ter sucesso e cumprir os objetivos.

Schwaber e Sutherland são autores de diversos artigos que explicam as motivações que originaram o *Scrum*. Segundo estes autores, é uma metodologia baseada num processo empírico, iterativo e incremental que é orientada por três aspetos base: transparência, inspeção e adaptação.

Sendo o *Scrum*, um processo iterativo e incremental, ele tem como objetivo aprimorar prazos e controlar riscos. Ou seja, o seu papel é fazer transparecer a eficácia das práticas de desenvolvimento de *software* e fornecer uma framework dentro das quais os produtos complexos podem ser desenvolvidos.(Schwaber & Sutherland, 2011)

Ele atua, principalmente, na gestão de projetos sem determinar como a equipa vai executar as tarefas de programação. Este método favorece a auto-organização da equipa e permite a integração com outras metodologias ágeis que focam na programação, como por exemplo *XP*.

Outra das principais características das equipas geridas por esta metodologia, é a autonomia. Esta tal como a autogestão é uma característica fundamental para o êxito das equipas (Moe et al.,2008).

Contudo, o seu foco principal é encontrar uma forma de trabalhar os elementos da equipa para produzir um *software* de forma flexível num ambiente em constante mudança. Por isso, a gestão de topo das empresas deve-se comprometer com as equipas *Scrum*, dando-lhes apoio e espaço de decisão, atribuindo assim um voto de confiança que deve ser confirmado no final de cada ciclo iterativo bem-sucedido (Beavers, 2007).

Transparência, Inspeção e adaptação: O Scrum é fundamentado em teorias empíricas de controlo de processos, o empirismo. Este afirma que o conhecimento vem da experiência. Segundo Schwaber, estes são os três pilares que apoiam a implementação do controlo do processo empírico.

A transparência garante que todos os aspetos relevantes ao processo se mantenham visíveis e conhecidos, de modo a garantir que o resultado obtido seja equivalente à definição de "pronto"[1] utilizada.

A inspeção é realizada com a finalidade de se detetar qualquer inconformidade que possa vir a prejudicar os resultados da equipa.

A adaptação indica que, a partir da identificação de irregularidades, são feitas as adaptações no processo, reduzindo a probabilidade de um resultado insatisfatório.

Qualidade: O *Scrum* possui iterações rápidas e a utilização de padrões de qualidade, demonstra uma rápida adaptação as mudanças, entre outras características. No entanto, ele possui ênfase no controlo do projeto, pois não diz como fazer, apenas acompanha os resultados. Essa constante verificação/validação no *Scrum* é o que representa a dimensão da qualidade, segundo Highsmith.

[1] Quando o item do *product backlog* ou o incremento é descrito como "pronto" todos devem entender o que o produto, embora varie significativamente de um extremo ao outro para cada equipa *Scrum* pois os integrantes devem ter um entendimento compartilhado do que significa o trabalho estar completo assegurando sempre a transparência. De acordo com Schwaber e Sutherland (2010) " "Pronto" define o que a equipa quer dizer quando se compromete a "aprontar" um item de product backlog em uma *sprint*".

Embora o *Scrum* não seja explícito no procedimento para garantir a qualidade, o compartilhamento de informações ao longo de todo o processo favorece a adoção de boas práticas e da constante verificação se os incrementos estão a corresponder às necessidades do cliente.

A definição de "pronto" também resulta em uma etapa de qualidade, pois os incrementos devem ser testados e deve funcionar com todas as outras funcionalidades já implementadas. A seguir, os fatores de qualidade em métodos ágeis, para Mnkandla*,* aplicados ao *Scrum*:

Compatibilidade: Cada parte do produto deve ser testado. Neste sentido as iterações rápidas e o desenvolvimento em pequenas equipas favorecem a criação de componentes compatíveis;

Correção: Utilização de boas práticas desde o início. O cliente está sempre presente e tem como função verificar se desenvolvimento do projeto esta acordo com os requisitos declarados no backlog.

Efetividade de Custo: Os gráficos *burndown* auxiliam no acompanhamento dos gastos ao longo do tempo do projeto;

Eficiência: O compartilhamento contínuo de informações entre os membros da equipe favorecem o uso de artefatos que irão otimizar os recursos de *hardware* e *software.* A capacidade do produto demonstrar o desempenho apropriado relativamente à quantidade de recursos usados sob condições específicas.

Extensibilidade: Na reunião os objetivos são definidos e caso surja um item novo no backlog, as prioridades são alteradas e o âmbito do *sprint* redefinido.

Integridade: A cada reunião de planeamento cada integrante da equipa compartilha as lições aprendidas – de sucesso ou não. Seguindo a característica empírica do Scrum, o grupo adota novas práticas ao longo do processo;

Manutenibilidade: Capacidade do *software* ser modificado. As modificações podem incluir correções, adaptações ao *software* devido às mudanças no ambiente, ou seja, à medida que há

uma interação maior da equipa em cada *sprint*, os erros são detetados rapidamente, assim como é possível projetar software com maior atenção a integração da modularidade;

Oportunidade: Ao fim de cada *sprint* é realizada a entrega de uma funcionalidade do produto. À medida que ocorrem os *sprints*, a lista do *backlog* é reduzida e pode ser modificada a prioridade da lista de acordo com as necessidades do cliente;

Portabilidade: Capacidade do software se transferido de um ambiente para outro. O Scrum adota padrões em todas as fases de desenvolvimento do produto, isso permite a criação de arquiteturas independentes de *hardware*;

Robustez: Enquanto o produto existir, o backlog também existirá. O dono do produto acompanha a execução de todas as iterações e oferece sempre feedback sobre o atendimento dos requisitos. As situações não previstas são tratadas rapidamente;

Verificação e validação: Reuniões diárias permitem conferir o andamento do trabalho realizado por cada integrante da equipa. Cada iteração significa uma entrega ao cliente, o produto é verificado, validado e testado constantemente.

Casos Práticos: O *Scrum* não emergiu em laboratórios mas na própria indústria com o objetivo de tentar combater as causas que proporcionam uma grande taxa de cancelamento dos processos. Por conseguinte, é natural que apareçam vários casos de estudo reais onde o *Scrum* foi implementado nas organizações.

Grandes organizações como a Microsoft, Primavera, Oracle, Yahoo, Motorola, IBM, HP, Philips entre outras possuem projetos que utilizam *Scrum*.

Apesar destas gigantes tecnológicas, o Scrum pode ser utilizado em pequenas e médias empresas (PMEs).

Relativamente as pequenas e médias empresas (PMEs) da área de desenvolvimento de *software*, estas apresentam uma serie de limitações como: limitações de custo, falta de tempo e recursos, falta de profissionais nas equipas e falta de equipamentos, o que leva as PMEs a procurar um método com custo mais baixo e eficaz para o desenvolvimento de *software*. Uma das soluções é a utilização do *Scrum*.

Uma das vantagens da sua utilização nas PMEs é o custo de desenvolvimento ser menor, haver uma maior produtividade, maior satisfação do cliente e maior oportunidade de negócio no desenvolvimento do produto.

Segundo Beck 2001, o *Scrum* disponibiliza a informação através de uma comunicação em tempo real e em ciclos iterativos. Pois uma das suas caraterísticas fundamentais é a disponibilidade da informação através da comunicação em tempo real e em ciclo iterativos, ou seja, as PMEs baseadas neste processo apresentam as suas etapas divididas em ciclos de pequena duração.

Contudo, no processo de desenvolvimento em que é utilizado o *Scrum* verifica-se que as PMEs podem ser mais flexíveis e adaptam-se mais facilmente ao processo porque o ele tem uma maior flexibilidade para responder de forma dinâmica às mudanças no mercado de acordo com as necessidades do cliente. (C.K Prahalad & Hamel, 1994)

No entanto, foi aprovado que esta abordagem pode ser utilizada em qualquer área de produção, até mesmo em disciplinas universitárias de programação, que pode trazer vantagens relativamente à produtividade e qualidade nos produtos (Scharff & Verma, 2010).

Benefícios e Atributos: Nas metodologias existem benefícios e atributos na sua adoção. Os principais benefícios obtidos do *Scrum* nas organizações são os seguintes:

- Melhor adaptação na gestão de projetos com constantes mudanças;
- Aumento do retorno do investimento em projetos de novos produtos;
- Melhoria na comunicação entre os elementos da equipa e um maior espirito colaborativo. Assim, a oportunidade de os projetos serem um sucesso são maiores, pois há uma boa comunicação, entendimento e cooperação mútua;
- Por ser iterativo e incremental permite a entrega antecipada de uma parte funcional do *software* desenvolvido;
- A entrega do *software* usando o *Scrum* é mais rápida do que se utilizasse outra metodologia tradicional.

Quanto aos atributos que esta abordagem apresenta, eles são os seguintes:

- Processo ágil para gerir e controlar o trabalho desenvolvido;
- Uma abordagem baseada em equipas para desenvolvimento de sistemas quando os requisitos estão a passar pelo processo de mudança;

- Aprimorar a comunicação e maximizar a cooperação e a produtividade;
- Uma forma de fazer com que todo o mundo se sentir bem com o seu trabalho, contribuições e saber que deram o seu melhor;

Vantagens e desvantagens: Todas as metodologias possuem vantagens e desvantagens e o *Scrum* não é diferente.

Uma das grandes vantagens desta metodologia é a comunicação que existe entre todas as pessoas envolvidas na equipa, pois todos participam nas decisões, fazem com que a comunicação aconteça de forma clara e objetiva. Outro ponto positivo é a forma de se trabalhar com divisões de tarefas, isso faz com que todas as pessoas envolvidas, participem em todas as tomadas de decisões e nas resoluções dos problemas.

Vale ressaltar que, além da equipa de desenvolvimento, o cliente participa também de forma ativa desde o início do projeto, o que colabora ainda mais com a propagação da informação.

A participação do cliente traz fortes benefícios e de certa forma também se destaca no *Scrum*, pois o contato com os desenvolvedores, além de permitir a divulgação do que será de facto produzido, colabora com o feedback do cliente junto à empresa, que acontece de forma rápida e eficaz, colaborando, e muito, com o seguimento do processo. Desta forma, a organização estabelece uma relação de confiança e credibilidade junto ao seu cliente, pois, se todos participam com empenho, o resultado deste esforço com certeza cria produtos funcionais e com excelente qualidade, e a equipa sincronizada cria uma aceleração no tempo de desenvolvimento, ganhando-se com isso mais tempo; e tempo é dinheiro, quando falamos de desenvolvimento de *software*.

O *Scrum* também se destaca por ter uma forma de trabalhar bem flexível e de fácil adaptação, podendo ser aplicada em diversos ambientes. Além de trabalhar de forma transparente e eficiente, trabalha com constantes alterações do âmbito

Para além destas vantagens, é uma metodologia de fácil automatização, isso influência diretamente nos custos e prazos, e colabora de forma efetiva no controlo das atividades a serem desenvolvidas.

Devido ao facto do *Scrum* suportar diversas mudanças, essa vantagem, pode ser vista de certa forma, como uma desvantagem, isto porque, para suportar mudanças constantes, é necessário trabalhar de forma bem exigente.

Como no *Scrum* se permite mudanças de âmbito, isso pode fazer com que, crie projetos duradouros, já que no início do ciclo de desenvolvimento não temos a visibilidade concreta da finalização do projeto.

A definição das tarefas, quando não definidas adequadamente, criam dificuldades para estimar o custo e o andamento do projeto, já que o tempo não é o fator principal mas sim a conclusão da tarefa.

2.2.1 Os Atores do *Scrum*

Product owner: É o "dono do produto". É o responsável por maximizar o valor do produto e do trabalho da equipa de desenvolvimento. Ele deve possuir a visão do produto em vários níveis.

Além de priorizar os requisitos do projeto é o responsável pelo ROI (*return of investement*) é ele quem decide quais os requisitos a implementar, assim como a configuração da equipa.

O *product owner* tem algumas responsabilidades que, nos métodos tradicionais de gestão de projetos de *software* se podem mapear com as responsabilidades do *project manager,* nomeadamente em relação aos requisitos e à relação com os *stakeholders* (Collaris et al.,2010). Na relação com o produto e com a equipa de desenvolvimento, o *product owner* é o único com autoridade para agir e tomar decisões.

Contudo, é essencial que ele tenha um conhecimento do negócio e consiga ter uma boa comunicação com todas as pessoas envolvidas no projeto. Além disso, ele deve ter autoridade para tomar e manter decisões e ser responsabilizado por elas (Cohn, 2011).

***Scrum Master*:** É a pessoa responsável por fazer o *Scrum* funcionar, ou seja, ele deve possuir o conhecimento de todo o processo para garantir que ele seja seguido. É ele o responsável pela implementação e utilização da metodologia a todas as pessoas envolvidas no processo, assim como assegurar que todos sigam as suas regras e práticas. (Schwaber & Sutherland, 2010).

O *scrum master* é diferente do tradicional gestor de projeto uma vez que não tem poder decisório sobre o trabalho a executar ou o produto que está a implementar. A função do *scrum master* é orientar todos os eventos do *Scrum*, a manutenção do *product backlog* e do *sprint backlog,* juntamente em comunicação com o *product owner* e a respetiva equipa. Ele também deve ser capaz de fazer um diagnóstico da equipa em cada *sprint*.

Equipa de desenvolvimento: É responsável pelo desenvolvimento do produto e é formada por desenvolvedores. Consiste em profissionais que realizam o trabalho de entregar uma versão usável que potencialmente incrementa o produto "Pronto "ao fim de cada *sprint*.

As equipas são estruturadas e auto-organizáveis, com um alto nível de autonomia e responsabilidade. Num projeto gerido pela metodologia *Scrum,* o número de equipas de desenvolvimento deve conter entre 3 a 10 elementos.

A equipa de desenvolvimento é a que desenvolve as funcionalidades do produto e são responsáveis pelo sucesso da iteração e consequentemente pelo projeto todo (Marçal; Freitas; Soares; Maciel; Belchior, 2007).

2.2.2 Os Artefactos do *Scrum*

***Product Backlog*:** Lista priorizada de todos os requisitos funcionais que o produto deve possuir e que ainda não foram implementados. Em cada item desta lista temos um valor de negócio associado, onde se pode medir o retorno do projeto e a priorização dos itens.

Este artefacto é da responsabilidade do *product owner* e gerido com a ajuda do *scrum master*.

Os requisitos são representados em formato *user stories* (frase que especifica o que se quer implementar e que ator se relaciona com esse requisito).

O *product backlog* lista todas as características, funções, requisitos, melhorias e correções que formam as mudanças a serem feitas no produto nas versões futuras. Os itens do *product backlog* possuem os atributos da descrição, ordem e estimativa e geralmente é ordenado por valor, risco, prioridade e necessidade.

***Sprint Backlog*:** É uma lista de tarefas (conjunto de itens do *product backlog* selecionados para a sprint), onde temos o trabalho de equipa em cada *sprint* do processo. A lista é criada durante o planeamento do *sprint*. As tarefas do *sprint* backlog são as que a equipa definiu como sendo necessárias para a influência da realização dos itens do *product backlog* nas funcionalidades do *software*.

***Sprint Burndown Charts* (Gráficos de acompanhamento):** Este artefacto está associado ao *sprint backlog* e é a visualização de um gráfico, como ilustra a figura 2, que tem a função de monitorizar o desenvolvimento da equipa.

Esta figura, que representa um gráfico é usado como uma das principais ferramentas para acompanhar o percurso da *sprint* e tem como função relacionar as horas de trabalho disponíveis como o somatório das horas das tarefas que ainda estas por realizar. (Kniberg; Skarin, 2009). Este gráfico é visualizado pela equipa em *Daily Scrum*.

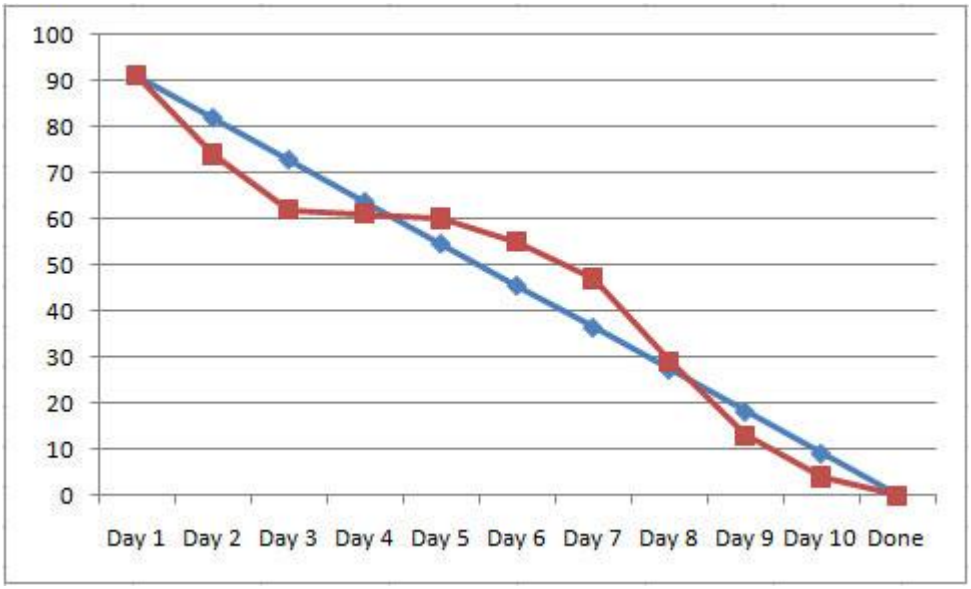


Figura 2- *Burndown Charts* (Fonte: http://certschool.com/blog/agile-burn-down-charts/)

User Stories: São pequenas descrições sobre as funções do sistema, cujos detalhes devem ser negociados entre o cliente e a equipa de desenvolvimento (Cohn, 2004). Após serem elaboradas pelo cliente ou *product owner*, as *user stories* podem ser divididas em tarefas de menor granularidade pelos elementos da equipa *Scrum*.

Uma *user story* contém três variáveis, ilustradas na figura 3, que dependem umas das outras. São elas: âmbito, importância e estimativa. O âmbito e a importância são definidos pelo *product owner*, enquanto a estimativa é definida pela equipa (Kniberg, 2007).

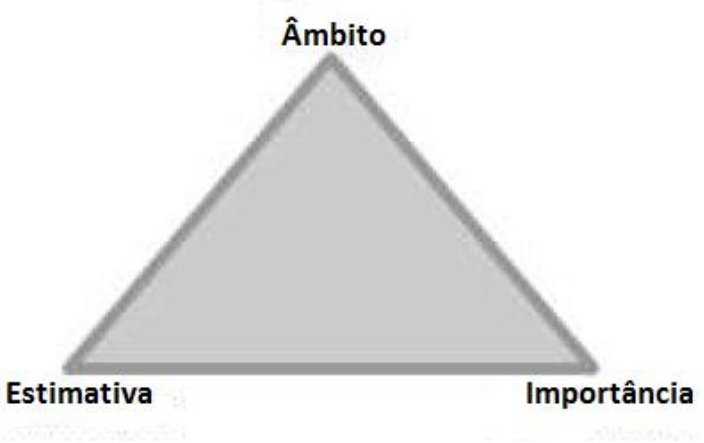


Figura 3- Variáveis de uma *Story* (Fonte: *Kniberg*,2007)

Algumas vezes o *product owner* não quer ou não pode participar no *sprint planning meeting* e justifica-se, dizendo que já listou os requisitos que deseja no *product backlog*. Esse é um problema grave, pois a equipa e o *product owner* devem selecionar as *user stories* juntos e refinar essas três variáveis em cada storie selecionada (Kniberg,2007).

As *User Stories* têm a mesma finalidade dos casos de uso *UML (Unified Modeling Language*). Normalmente, cada *user story* é um conjunto de frases curtas escritas pelo cliente para definir o que o sistema deve fazer para ele. Elas são utilizadas para criar estimativas de tempo nas reuniões de planeamento e, mais importante, analisar se o que foi desenvolvido corresponde às necessidades especificadas na *user story* (Neto, 2004).

2.2.3 Ciclo do *Scrum*

No *Scrum* os projetos são divididos em ciclos que são chamados de *sprints*. Estes dizem respeito ao espaço de tempo em que será executado um conjunto de atividades referente ao projeto. Tradicionalmente os ciclos de desenvolvimento duram cerca de trinta dias (Schwaber, 1995).

Todas as atividades ficam guardadas sob o controlo do *product owner* numa lista, que é chamada *product backlog*, em períodos de tempo essa lista é priorizada e dá origem a uma outra lista de atividades, que é chamada de *sprint backlog*, esta contém todas as atividades que serão desenvolvidas no *sprint*.

No decorrer do mesmo é realizada uma reunião diária para o alinhamento entre a equipa, onde todos ficam a saber o que está a ser realizado.

Desta forma é possível detetar os problemas e assim resolvê-los rapidamente. No fim da *sprint,* teremos o resultado do trabalho, seja uma funcionalidade ou um produto concluído, conforme o planeamento na *sprint backlog*. Na figura abaixo está ilustrado o ciclo do *Scrum*.

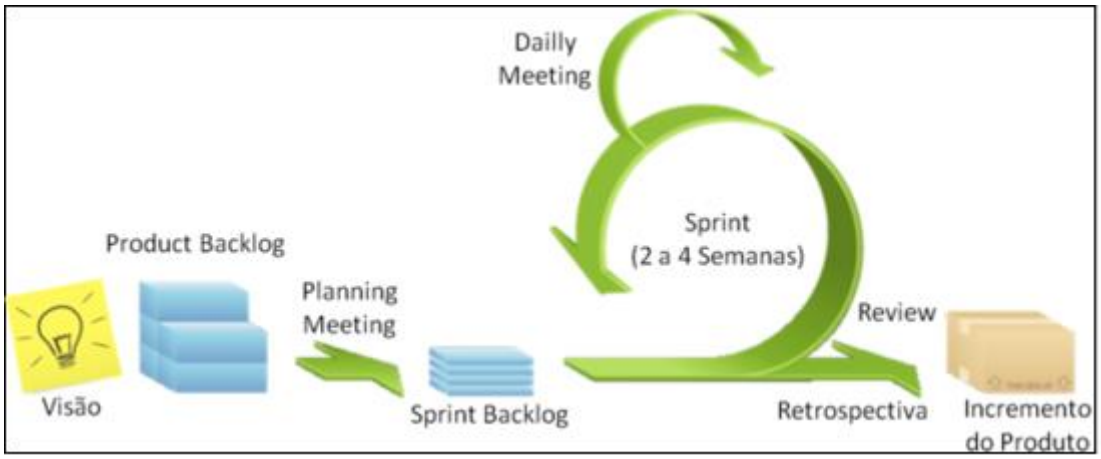


Figura 4 - Ciclo de vida do Scrum (Fonte: http://www.ideiaagil.com./imagens/Fluxo_Scrum.png)

2.3 *Scrum* e as suas Principais Práticas

As práticas do *Scrum* são fundamentais para o sucesso esperado por uma empresa que decide adotá-lo. A literatura menciona diversas práticas definidas pelos autores Larman, Highsmith e Schwaber, sendo elas as seguintes:

Sprint:. O coração do *Scrum* é a *sprint*. Os projetos que utilizam o *Scrum* são divididos em ciclos chamados de *sprint*, os quais correspondem a iterações.

Sutherland define *sprint* como "uma iteração, um ciclo repetitivo de trabalhos que produz um incremento de produto. O ciclo não é superior a um mês tendo geralmente a duração de mais de uma semana do ciclo. Eles permitem previsibilidade que garantem a inspeção e adaptação do progresso em direção à meta pelo menos a cada mês ocorrido." (Sutherland, 2012)

Pré- Game: O planeamento inicial é uma das primeiras práticas do *Scrum*. O *product owner* reúne-se com os interessados para definir o objetivo do projeto e a lista inicial das funcionalidades.

Depois desta definição inicial realizada, o *product owner* agenda uma reunião com os elementos da equipa para explicar as *user stories* que foram criadas e retirar as dúvidas que possam existir. Normalmente, os elementos aproveitam o momento em que estão juntos e estimam as *user stories* numa técnica chamada de *Planning Poker*[2]. (Cohn, 2005).

Sprint Planning: A *sprint planning* é realizado no início de cada *sprint*. Começa com uma reunião com todos os elementos, *product owner, scrum master* e a equipa de desenvolvimento e nesta reunião, o *product owner* define o objetivo da *sprint* e descreve as suas funcionalidades do *product backlog*.

A equipa de desenvolvimento tira as suas dúvidas com o *product owner* para entender melhor cada item e definir quais os itens e equipa que se irá comprometer a entregar na *sprint*, criando-se assim, o *select product backlog*.

Após a primeira reunião, a equipa reúne-se para quebrar as funcionalidades em tarefas e coloca-las na *taskboard*.

[2] Método usado pela equipa para estimar o tempo de desenvolvimento de cada *user stories*. Segundo Lori Schubring o "*planning poker* é uma boa maneira de chegar a um consenso sem gastar muito tempo em qualquer tópico. Ele permite que as pessoas expressam as suas opiniões, pensamentos e preocupações."

O sucesso da *sprint* será avaliado durante a *sprint review* e as melhores práticas serão adotadas durante a *sprint retrospective*. A cada *sprint planning*, o *product owner* pode acrescentar, alterar e remover os itens do *product backlog,* devendo comunicar sempre as alterações efetuadas á equipa.

Esta prática é importante porque permite a cada *sprint* estimar melhor os itens e verificar como ela deve ser melhor acompanhada para o progresso do projeto.

Daily Meeting: É uma das práticas mais importantes do *Scrum,* pois é nesta prática que se tem oportunidade diária de todos os elementos interagirem e sincronizar o que estão a fazer e comunicar ao *scrum master* as suas dificuldades para encontrarem uma possível solução. Esta prática deve ocorrer de preferência no mesmo local e no mesmo horário. Nela, os elementos da equipa responde a três questões:

- O que foi realizado desde a última *Daily Scrum*?
- O que será feito até à próxima *Daily Scrum?*
- Existe algum obstáculo para a realização da tarefa a decorrer?

Ela é normalmente realizada à frente da *taskboard*, pois esta ajuda a acompanhar as tarefas. Nesta reunião, cada elemento deve indicar o que fez desde a última reunião e o que pretende fazer ate à próxima reunião. Depois de todos os elementos conversarem é criada uma *sprint burndown* para verificar o progresso de cada *sprint*.

O *scrum master* verifica os impedimentos identificados e tenta resolve-los rapidamente.

Apesar destas reuniões serem de curta duração, Schwaber destaca que esta prática evita duplicação de trabalho, uma melhor compreensão sobre as interdependências entre os elementos da equipa e uma produção dinâmica. Esta é uma reunião chave para a inspeção e a adaptação.

Scrum of Scrum: Schwaber diz que muitos projetos requerem mais esforço do que uma única equipa *Scrum* pode fornecer. Por isso, nestas circunstâncias várias equipas podem ser implementadas, sendo necessário coordenar o trabalho em paralelo entre elas da melhor forma para mantê-las mais sincronizadas.

Uma forma de coordenar este trabalho em conjunto é realizar uma reunião semelhante a um *daily meeting* só que num contexto maior (Schwaber, 2004).

Esta prática consiste na realização de uma reunião com a participação de representantes de cada equipa. O representante pode ser diferente em cada reunião. Geralmente é escolhido aquele com maior capacidade técnica.

Nesta reunião é detalhado o que a equipa fez e irá fazer. Deste modo, este encontro serve para sincronizar o trabalho das equipas, dar uma visão de como está a correr o projeto e solucionar os problemas encontrados (Cohn, 2005).

Sprint Review: Toda a *sprint* possui um objetivo que normalmente é atingido com a execução de uma parte do produto (incremento), que foi acordado entre o *product owner* e a equipa. O incremento deve conter os itens que foram definidos no select *product backlog*. No final de cada *sprint* os elementos da equipa devem apresentar o resultado da *sprint* ao *product owner.*

Nesta reunião de revisão, a equipa de desenvolvimento demonstra o trabalho que está "pronto" e responde às questões sobre o produto

O resultado do *sprint review* é um *product backlog* revisto, que define o acompanhamento do progresso do projeto, desde o momento inicial ate à entrega final com o cliente.

Sprint Retrospective: A *sprint retrospective* consiste numa reunião, que normalmente, é realizada logo após o *sprint review* e só tem a participação do *scrum master* e da equipa de desenvolvimento. Esta prática tem como objetivo discutir o que deu certo e errado durante a *sprint* que os elementos da equipa avaliam.

Esta reunião permite também reforçar a visão do produto, possibilitando que os elementos interajam e analisem como podem ajudar os outros membros em prol do projeto, fazendo com que a equipa se torne cada vez mais auto-organizável.

A *sprint retrospective* cria um evento dedicado e focado na inspeção e adaptação, em que as melhorias podem ser adotadas a qualquer momento.

Grooming: Também apelidada de *backlog refinement meeting*, ou seja, é uma reunião em que não é obrigatório ocorrer em todas as *sprints*. Todos os elementos se reúnem, o *product owner, scrum master* e a equipa de desenvolvimento e escrevem os requisitos do produto em forma de *user story*. Normalmente, a duração desta reunião é de quinze minutos.

No entanto, esta reunião é importante, pois serve para haver um entendimento entre o *product owner* e a equipa de desenvolvimento sobre o que ainda falta realizar sobre o produto.

2.4 Modelos de *Scrum* Distribuídos

Muitas empresas estão preocupadas com a melhoria do processo de desenvolvimento de *software*, propondo a redução de custos, o aumento na qualidade do produto e a satisfação do cliente. Para materializar estes benefícios, várias dessas organizações têm procurado a implementação de uma nova metodologia de desenvolvimento, as metodologias ágeis.

Tal como o *Scrum,* estas metodologias começaram a apresentar bons resultados na indústria e neste sentido, novas técnicas do *Scrum* foram estudadas de forma a adaptá-las às suas práticas, artefactos e atores para que as equipas que utilizarem o *Scrum* possam estar geograficamente distribuídas, ou que várias equipas trabalhem sobre o mesmo produto.

Sutherland descreve que as equipas distribuídas podem ser organizadas em três modelos diferentes:

***Scrum* isolados:** As equipas são isoladas entre localidades. Em alguns casos, algumas equipas não utilizam o *Scrum*, o que torna difícil a gestão do projeto. Contudo, alguns integrantes podem trabalhar de forma distribuída e cada equipa possui um *product owner*. A figura 5 ilustra uma visão do modelo *Scrum* isolados, onde não existe ligações entre o *Scrum*.

Figura 5 - *Scrum* Isolados (Fonte: http://www.revistas.unifacs.br/index.php/rsc)

***Scrum of Scrum* distribuído:** As equipas de *Scrum* são isoladas em diferentes localidades e são integradas por meio do *Scrum of Scrum*. Neste tipo, o *product owner* é quem coordena o trabalho de múltiplas equipas.

A figura 6 mostra uma visão do modelo *Scrum of Scrum* distribuído. Neste modelo há eliminação de dependência entre as equipas distribuídas geograficamente existindo uma divisão de trabalho entre as equipas multifuncionais colocadas.

Figura 6 - *Scrum of Scrum* distribuído (Fonte: http://www.revistas.unifacs.br/index.php/rsc)

***Scrum* totalmente integrado:** As equipas *Scrum* são formadas por elementos com especialidades distintas e em diferentes localidades. Neste tipo de reuniões diárias ajudam a romper as barreiras culturais e as disparidades dos estilos de trabalho, o que faz com que este modelo crie uma confiança entre os elementos da equipa.

A figura 7 mostra uma visão do modelo *scrum* totalmente integrado, onde as reuniões do *scrum* acontecem com toda a equipa, mesmo quando a equipa está geograficamente distribuída.

Figura 7- *Scrum* totalmente integrado (Fonte: http://www.revistas.unifacs.br/index.php/rsc)

Entre os modelos, o mais recomendado pelo Scrum Alliance é o *Scrum of Scrum* distribuído. Neste modelo, cada equipa tem um *scrum master* que monitoriza diariamente o progresso dos seus membros.

Este modelo incentiva uma maior comunicação e colaboração entre as equipas distribuídas fazendo com que elas trabalhem como uma só em prol do projeto, o que desta forma ajuda numa resolução rápida de problemas entre elas (Sutherland et al., 2007).

2.5 Conclusão

As metodologias ágeis surgiram como uma reação ao desenvolvimento de *software* burocrático defendido pelas metodologias tradicionais. A sua adoção cresceu pelo facto de amenizarem os principais problemas que ocorrem ao longo do projeto, tais como: o atraso na entrega, orçamentos elevados e a insatisfação do cliente.

O conceito de desenvolvimento de *software* ágil é descrito como "o termo oposto do desenvolvimento de *software* tradicional, que se refere ao grupo de abordagens de desenvolvimento leves definidas como: facilitar a rápida resposta ao mercado (*time-to-market*) e a integração contínua de novos requisitos para aumentar a produtividade de desenvolvimento enquanto mantém a qualidade de *software* e flexibilidade (Deved e Milenković, 2011)".

Atualmente, as organizações, cada vez mais, necessitam de desenvolver *software* flexível para obter uma maior agilidade na entrega do produto. Para resolver esta situação, as organizações adotam uma filosofia ágil que faz com que a entrega do sistema de *software* ao cliente seja mais rápida, eficiência e eficaz. Esta filosofia é o *Scrum*.

Scrum é uma metodologia ágil que consegue reconhecer o que não está a funcionar e que sabe lidar com problemas enfrentados pela gestão de projetos, baseando-se num conjunto de boas práticas sobre os aspetos crucias para o sucesso de implementação de um projeto de *software*.

Simultaneamente, o *Scrum* foi experimentado e utilizado em projetos que envolve diferentes organizações para implementar o mesmo produto, obtendo assim bons resultados em projetos de dimensões limitadas (Dingsøyr et al., 2006).

Apesar dos testemunhos positivos e dos bons resultados alcançados, existem detalhes que o Scrum ainda não é capaz de lidar como: a abordagem metodológica *Scrum* não mencionar nenhum procedimento ou protocolo relacionado com o desenvolvimento do produto. O *Scrum* apenas indica como a equipa deve ser gerida a um alto nível focando-se apenas na relação entre os intervenientes no projeto.

Outro problema do *Scrum* é o tempo de adaptação que uma equipa leva até conseguir render o esperado. Este confere à equipa grande poder de definição dos objetivos e necessita de uma comunicação, conhecimento e comprometimento entre todos os elementos da equipa.

A dependência do *Scrum* na presença assídua do cliente pode tornar a utilização desta abordagem para os casos onde o cliente não está disponível ou não tem a capacidade de gerir,

testar o trabalho realizado. Neste caso, a organização deve encontrar um responsável para conduzir a equipa ao sucesso esperado.

É de ressaltar que a literatura já tem demonstrado o quanto o *Scrum* pode ser um grande aliado em projetos de desenvolvimento de *software* e uma das formas que a literatura apresenta para contrariar estas limitações é a adaptação do scrum em ambientes distribuídos.

Ainda não existe um referencial complexo como implementar o *Scrum* num projeto com uma equipa distribuída. Por isso, novos procedimentos vão ser estruturados através das práticas do *Scrum* e das características de desenvolvimento distribuído de *software* com a finalidade de que o projeto não termine em fracasso.

Neste capítulo, verificou-se que o *Scrum* possui vantagens e desvantagens e em alguns momentos do projeto uma desvantagem pode vir a causar serias falhas em projetos.

O desafio futuro do *Scrum* é encontrar meios de minimizar as suas desvantagens sem transforma-las em metodologias pesadas, como aumentar o número de pessoas que interagem com a equipa sem perder a confiabilidade e a eficiência na gestão da mudança. Portanto, uma solução para minimizar as suas desvantagens é a aplicação de um novo método de desenvolvimento, o desenvolvimento distribuído de *software*.

A junção destes dois métodos pode trazer grandes benefícios para o desenvolvimento do projeto, daí a necessidade de caracterizar o desenvolvimento distribuído de *software* e propor um conjunto de características para uma melhor compreensão deste tipo de desenvolvimento, no qual vai ser ilustrado no capítulo a seguir, capitulo 3 - Alterações ao *Scrum* para Ambientes DDS.

CAPITULO 3 – ALTERAÇÕES AO SCRUM PARA AMBIENTES DDS

3.1 Introdução

Scrum para ambientes distribuídos pode parecer algo meio paradoxo, uma vez que o *Scrum* é um método ágil, em que um dos seus benefícios é referente à forma como a equipa trabalha sendo ela de forma unida, lado a lado.

Atualmente, percebe-se que existe um crescimento do número de empresas na adoção das abordagens ágeis, contudo, elas também tem adotado outro tipo de desenvolvimento, o desenvolvimento de *software* distribuído, conseguindo muitos benefícios como: a redução de prazos de entrega, uma maior produtividade e mais eficácia e uma menor burocracia no desenvolvimento.

As metodologias ágeis, como o *Scrum* não abordam como aplicar as suas práticas num contexto distribuído. Desta forma, é necessário que elas tenham um apoio, que pode ser encontrado na literatura, através da utilização das características defendidas pelo desenvolvimento distribuído de *software*.

Estas características podem ser aplicadas, independentemente das metodologias aplicadas, pois o seu objetivo é ajudar no acompanhamento do projeto. Deste modo, estas são complementares às práticas do *Scrum* não existindo uma relação direta entre elas.

Com a globalização e a crescente procura por novos sistemas, as empresas deparam-se com a necessidade de estudar novas formas de conseguir uma produtividade elevada reduzindo o tempo e os custos. Uma opção investigada, para alcançar este objetivo foi o desenvolvimento distribuído de *software* (Karolak, 1999).

O desenvolvimento de *software* é tradicionalmente realizado por pessoas que fisicamente estão localmente próximas. Entretanto, em algumas situações as partes envolvidas estão separadas espacial e temporalmente devido a razões que variam desde a necessidade de redução de custos até a necessidade de disponibilizar uma mão-de-obra altamente qualificada. Nesses casos é necessário realizar uma outra forma de desenvolvimento de *software*: o desenvolvimento distribuído de *software*.

Por conseguinte, este capitulo tem como objetivo sistematizar a junção destes dois métodos de desenvolvimento, através da caracterização do desenvolvimento distribuído de *software* e adaptação das vertentes do *scrum* com o desenvolvimento distribuído.

3.2 Desenvolvimento Distribuído de *Software*

O desenvolvimento distribuído de *software* é conhecido principalmente pela sua colaboração e cooperação entre departamentos de organizações e por grupos de pessoas que trabalham no mesmo projeto ou com o mesmo objetivo, sendo que os participantes estão sediados na mesma localização física ou em cidades ou até países diferentes.

Para Carmel (1999) as principais características que diferenciam o desenvolvimento distribuído de *software* do desenvolvimento co-localizados são: a dispersão geográfica, a dispersão temporal e as diferenças culturais.

O autor destaca que trabalhar com o desenvolvimento distribuído de *software* é um dos maiores desafios que o ambiente atual dos negócios apresenta do ponto de vista do processo de desenvolvimento de *software*.

Diversos fatores contribuíram para acelerar o surgimento do desenvolvimento distribuído de *software*, entre eles podemos citar:

- A redução de custos e indisponibilidade de mão-de-obra;
- A necessidade de recursos globais para serem utilizados a qualquer hora;
- Maior acesso a recursos de telecomunicação;
- Vantagens de se aproximar dos mercados locais;
- A grande pressão para o desenvolvimento *time-to-market*[3], que utilizam as vantagens proporcionadas pelo diferente fuso horário, no desenvolvimento conhecido como *follow-the- sun*[4] (24 horas continuas tendo em conta que as equipas estão fisicamente distantes).

Neste sentido, o desenvolvimento distribuído de *software* acrescenta fatores como a dispersão geográfica, a dispersão temporal e a comunicação, que acentuam alguns dos desafios existentes e acrescentam novos desafios ao processo de desenvolvimento.

Vários fatores são utilizados para caracterizar o desenvolvimento distribuído de *software*, sendo alguns considerados fatores críticos de sucesso nos processos de desenvolvimento

[3] Tempo entre a análise de um produto e a sua disponibilização no mercado.

[4] As equipas são distribuídas ao redor do mundo de forma que há sempre pelo menos uma equipa disponível para realizar o trabalho. Desta forma uma desvantagem do DSS (a distribuição geográfica) torna-se numa vantagem competitiva que permite trabalhar durante 24 horas por dia.

distribuído de *software*. Desta forma, para o sucesso de um projeto num ambiente de desenvolvimento distribuído de *software*, identifica-se um conjunto de fatores (figura 8):

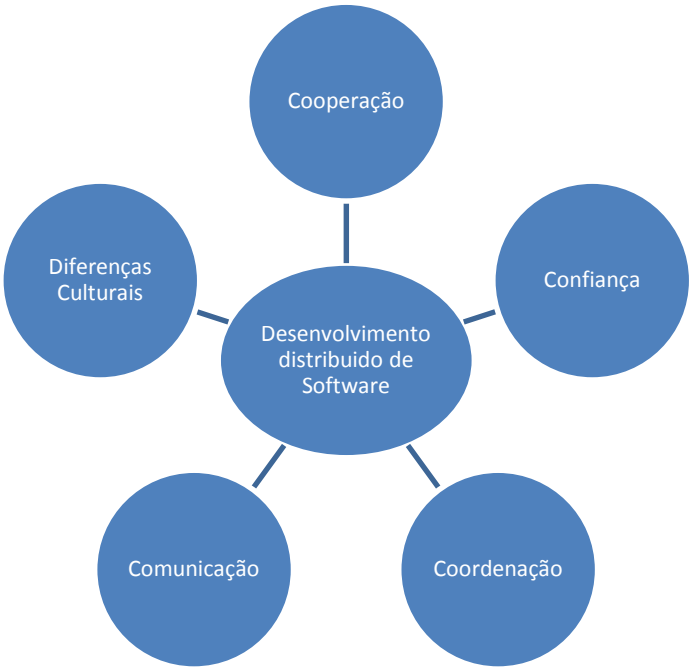


Figura 8- Fatores de desenvolvimento distribuído de *software* (Fonte: Elaborado por mim)

Diferenças Culturais: Quando o desenvolvimento de *software* é um cenário onde existe desenvolvimento distribuído de *software*, uma das primeiras coisas a verificar é o nível de diferenças culturais existentes entre as equipas fisicamente distantes. Estas diferenças devem ser identificadas e acomodadas entre os elementos do processo.

As diferenças culturais também podem afetar a eficácia da equipa na comunicação e na colaboração.

Há culturas que promovem assumir riscos, outras não. Além disso, enquanto umas desafiam todos, outras promovem a confiança (Woodward et al., 2010).

Comunicação: A comunicação num desenvolvimento distribuído de *software* é a chave para a execução de um projeto. A falta de comunicação faz com que as equipas fisicamente distantes não tenham conhecimento de informações básicas sobre o projeto, sobre a equipa do projeto, entre outros. Por isso, deve existir um fluxo de informações ágil e eficaz entre os membros da equipa.

Coordenação: A coordenação neste tipo de desenvolvimento é um fator que visa dispor as atividades de forma ordenada que são baseadas em processos e regras previamente definidas.

Confiança: A confiança em desenvolvimento distribuído de *software* é de uma importância vital para o bom fluxo de informações entre as equipas distribuídas. Ter confiança é ter segurança e firmeza no trabalho da equipa como um todo.

Por isso, é essencial procurar mecanismos que proporcionem um clima de confiança nas relações e ações entre as pessoas envolvidas.

Cooperação: A cooperação num desenvolvimento distribuído de *software* significa que, a colaboração da equipa tem objetivo comum, ou seja, ajudar e pensar como uma só equipa. Por isso, é fundamental a cooperação entre as equipas distantes fisicamente, pois não adianta existir uma boa coordenação e uma ótima comunicação se não existir cooperação entre os membros das equipas.

Além dos fatores, o desenvolvimento distribuído de *software* traz algumas vantagens competitivas para a solução global, tais como: redução do tempo de entrega através de equipas distribuídas que trabalham em paralelo, além da possibilidade de desenvolverem em diversos fusos horários e a redução de custos com recursos humanos, uma vez que contratar profissionais de outros países é muitas vezes mais barato do que os da mesma localidade (Hersbsleb and Moita, 2001).

A possibilidade de adquirir uma maior competitividade ao diminuir os custos, prazos e aumentar a qualidade dos produtos de *software* criados, entre outras motivações, faz com que o desenvolvimento distribuído de *software* seja uma opção cada vez mais recorrente às

organizações. No entanto, realizar um projeto deste tipo não é tão simples quanto um projeto em que o desenvolvimento é tradicional [5].

3.2.1 Características do Desenvolvimento Distribuído de *Software*

Este subcapítulo, tem como objetivo propor um conjunto de características que permitam entender melhor o desenvolvimento distribuído de *software*.

Para criar este conjunto de características foi realizado uma extensa pesquisa bibliográfica observando diversos relatos de experiências e discussões sobre este assunto. A partir destas fontes foram observados os principais problemas existentes neste tipo de desenvolvimento.

As caraterísticas que vão ser apresentadas são as seguintes (Siqueira e Muniz Silva, 2004):

- Agrupamento
- Distância Física
- Separação temporal
- Cultura regional
- Idioma
- Diferenças locais
- Culturas organizacionais
- Infra-estrutura
- Relação do negócio

[5] O desenvolvimento de *software* em que as partes envolvidas ou trabalham fisicamente próximas, ou estão razoavelmente acessíveis (cliente e utilizador), pode ser visto como um desenvolvimento tradicional.

Agrupamento: A principal ideia contida no desenvolvimento distribuído de *software* é que as pessoas não estão lado-a-lado mas não significa que as pessoas estejam separadas uma das outras.

Uma opção bastante comum é as pessoas estarem distribuídas em grupos e trabalhar lado-a-lado com algumas pessoas.

A forma como o grupo está distribuído demonstra ser uma questão multidimensional composta por três aspetos: a quantidade de pessoas, a quantidade de grupos e os papéis exercidos pelas pessoas.

Distância física: À medida que aumenta a separação física das pessoas, fica cada vez mais difícil realizar reuniões presenciais entre os membros do projeto, algo que pode ser importante durante alguns momentos do projeto.

Dependendo da distância, os meios de comunicação razoáveis são o telefone, videoconferência e conversa presencial (após uma viagem de avião).

Se existe pouca comunicação informal, a familiaridade entre os elementos do grupo e o sentimento do grupo também são afetados.

Por conseguinte, a distância física pode ocasionar alguns problemas. A comunicação entre pessoas pode sofrer com a desfasagem na transmissão de informação o que provoca inconvenientes durante uma videoconferência ou uma conversa por telefone.

Separação temporal: Os grupos de pessoas que efetuam o desenvolvimento distribuído de *software* podem estar dispersos pelo mundo, separados por grandes distâncias físicas.

Dependendo da distância entre os locais, a distância física acaba por trazer também uma distância temporal através das diferenças de fusos horários.

A existência de fusos horários diferentes diminui as interseções nos horários de trabalho entre pessoas de grupos diferentes.

Além da questão do fuso horário, a distância temporal pode também existir com uma diferença de horários de trabalho entre as pessoas do mesmo local de trabalho, como por exemplo a existência de diferentes turnos de trabalho.

Em suma, a separação temporal pode ser positivamente aproveitada para organizar o trabalho para um desenvolvimento durante 24 horas (Haywood, 2000).

Culturas Regionais: O desenvolvimento de *software* é uma atividade intensamente humana, contudo um dos problemas apontados em casos de desenvolvimento é o aspeto cultural.

Estes problemas ocorrem por causa das diferenças de comportamentos entre pessoas de diferentes culturas, como por exemplo: diferenças no planeamento do trabalho, no estilo da argumentação, entre outras (Olson e Olson,2003).A diferença de valores e práticas existentes entre pessoas provenientes de diferentes regiões.

Idioma: O idioma é uma peça chave para a comunicação humana. Sem ele, a passagem de informações torna-se precária em atividades coletivas como o desenvolvimento de *software*, a existência de um idioma é quase um pré-requisito.

Por mais que um idioma seja umas das manifestações da cultura, a sua importância durante um desenvolvimento de projeto faz com que ele sobressaia entre as demais diferenças culturais.

Diferenças locais: A localização de um dos grupos de um desenvolvimento distribuído de *software*, numa determinada região, afeta de diversas formas o desenvolvimento. O conjunto de problemas que melhor evidenciam as diferenças entre os locais são os aspetos jurídicos. Dois grupos localizados em diferentes países estarão sujeitos a diferentes leis. Essas diferenças de legislação podem afetar de diversas formas o desenvolvimento de *software*, como por exemplo dificultar a importação de um determinado *hardware* (Haywood, 2000).

Culturas organizacionais: No desenvolvimento de *software*, as diferenças de culturas organizacionais podem causar alguns problemas, como as diferenças de estratégias, objetivos, filosofias, crenças, perceções, pensamentos e sentimentos, que são criadas na organização onde as pessoas trabalham.

Infra-estrutura das organizações: Qualquer organização que desenvolve *software*, independentemente se realiza num desenvolvimento distribuído de *software* ou não, precisa de uma infra-estrutura adequada para permitir o trabalho das pessoas envolvidas.

Por infra-estrutura entende-se *hardware*, o *software*, as ferramentas, as técnicas e a operação e a manutenção do produto.

No desenvolvimento distribuído de *software,* a existência de uma infra-estrutura adequada por todas as partes envolvidas pode ser difícil de ser obtida. As organizações podem trabalhar com diferentes sistemas operacionais, ambientes de desenvolvimento, entre outras diferenças, que podem tornar a coexistência de inúmeros padrões um enorme desafio. Muitas vezes, o problema pode não ser exatamente um conflito na infra-estrutura, mas a falta de uma infra-estrutura adequada, um aspeto fundamental para o desenvolvimento distribuído de *software* é a existência de uma infra-estrutura adequada (Carmel, 1999).

Relação de negócio: Em projetos de desenvolvimento distribuído de *software* é bastante comum a existência de diversas organizações a trabalhar em conjunto, onde existe a divisão de atividades de desenvolvimento. Estas organizações podem estar ligadas através de uma contratação ou fazerem parte da mesma empresa.

Dependendo da natureza dessas relações, num projeto existirão algumas diferenças e problemas específicos no desenvolvimento de *software*. Segundo Carmel (1999) a relação de negócio é o que principalmente inclui na formação de uma equipa coesa. Além disso, a relação de negócio influi diretamente na passagem de conhecimento entre as partes (Kobitzsch, 2001).

Segundo Paasivaara os desafios causados pela distribuição incluem: problemas de comunicação, falta de proximidade física, falta de coesão da equipa, indisponibilidade dos membros da equipa, diferenças linguísticas e culturais (Paasivaara et al., 2009).

3.3.2 Práticas de Desenvolvimento Distribuído de *Software* para Acompanhar um Projeto *Scrum*

Na elaboração de um projeto, surgem vários problemas que devem ser solucionados para que o mesmo seja desenvolvido de forma mais eficiente e eficaz.

As práticas de desenvolvimento distribuído de *software* visam apoiar as características acima descritas, de forma a apoiar as equipas distribuídas, de modo a fornecer soluções para alguns desafios, como motivar o interesse e a dedução dos membros do projeto, na identificação de um bom líder capaz de organizar as equipas distribuídas.

As metodologias ágeis, tal como o Scrum não abordam como aplicar as suas práticas num contexto distribuído. Desta forma é necessário que elas tenham algum apoio para minimizar os

desafios enfrentados pelas equipas. Este apoio pode ser encontrado na literatura DDS que descreve algumas práticas que ajudam a minimizar os desafios enfrentados por estas.

Esta secção aconselha a adoção das seguintes práticas de DDS para apoiar os projeto que usam o *Scrum*:

Iniciar o projeto com as equipas juntas: esta prática sugere que todas as equipas devem iniciar o projeto no mesmo espaço físico, num local central o *product owner* e os clientes se encontram. Esta união serve para o *product owner* apresentar, a todos os elementos, a visão do projeto para que estas sejam elaboradas, realizar as primeiras *sprints* todos juntas e para que as equipas participem nas reuniões *Scrum,* nesse período (Phalnikar et al., 2009), (Vax and Michaud, 2008) e (Sutherland et al., 2007).

Contudo, devido aos orçamentos, nem sempre é possível que todos os elementos possam viajar. Nestes casos é necessário definir quais os elementos que vão viajar dando preferência aos elementos mais experientes como os representantes das equipas.

No entanto, é importante que o *product owner* realize uma reunião virtual com os elementos que não puderam viajar, para que estes tenham conhecimento pelo menos do objetivo do projeto.

Fornecer formação sobre os processos e as tecnologias que serão utilizadas no projeto: o objetivo desta prática é apresentar a forma de trabalho do projeto e realizar um alinhamento entre os elementos sobre os processos e as tecnologias que serão utilizadas. Assim, evita-se que cada equipa trabalhe de forma diferente facilitando a integração e os testes (Sureshchandra and Shrinivasavadhani, 2008).

Treinar a metodologia *Scrum* é um exemplo importante para que todas as práticas sejam bem entendidas e aplicadas de maneira correta. O *scrum master* é a pessoa certa para proporcionar este treino de *Scrum*.

É neste treino que é feito a referência de ferramentas, que são implementadas para a gestão ágil do projeto. O uso de uma ferramenta torna o treino mais dinâmico, além de melhorar a aprendizagem de outros elementos e permitir a realização de práticas em tempo real, tais como: criar *user stories*, realizar estimativas, acompanhar as tarefas utilizando a *taskboard* e acompanhar o projeto pelo *burndown charts.*

Esta prática torna-se mais efetiva quando todos os elementos estão presentes fisicamente, caso não seja possível, agenda-se um horário em que todos possam estar presentes, utilizando recursos de videoconferência.

Visitas frequentes: Começar o projeto com todas as equipas juntas é algo crucial e é importante que esse encontro não ocorra só inicialmente, principalmente quando se trata de projetos longos. Por isso, planear ao longo do projeto visitas regulares das equipas distribuídas para o local central e em alguns casos visitas de elementos do local central para o local das equipas distribuídas. (Paasivaara et al., 2008) e (Phalnikar et al., 2009).

Durantes estas reuniões as visitas devem participar nas reuniões que ocorram. Estas visitas devem ser planeadas para que sejam produtivas e que permitam alcançar o objetivo que é de realinhar o foco do projeto, resolver as possíveis dependências entre as equipas e reforçar os laços de confiança entre os elementos do projeto.

Meios de comunicação: Um ponto crucial num projeto distribuído é a comunicação. É importante estabelecer várias formas de comunicação como o *email*, videoconferência, chats.

Estes meios de comunicação ajudam na iteração de todos os elementos distribuídos, permitindo uma resolução de problemas do projeto mais rápida.

Envio de alguém mais experiente durante mais tempo para as equipas distribuídas: Em muitos projetos, as equipas distribuídas não possuem nenhum elemento com o conhecimento do domínio da aplicação que será desenvolvida.

Também existe o caso em que o orçamento do projeto não permite que todos os elementos comecem o projeto juntos, para difundir o conhecimento. Nestes dois casos, existe a necessidade de enviar um elemento experiente no domínio da equipa central para as equipas distribuídas por um maior período de tempo para que sejam difundidas as informações sobre o projeto (Paasivaara et al, 2008).

A vantagem desta prática é permitir que todos os elementos distribuídos tirem as suas dúvidas de forma a amenizar os problemas de comunicação. Também ajuda a reduzir a dificuldade em obter conhecimento sobre o projeto e evitar problemas no seu entendimento, como geralmente pode ocorrer durante a comunicação por meio de videoconferência entre a equipa distribuída e o cliente.

O conjunto de práticas apresentadas serve para apoiar as práticas do *Scrum* e a sua aplicação. Contudo, a sua aplicação vai depender dos recursos do projeto, e pode ser difícil aplicar em todos os projetos que surjam.

3.3 Adaptação do *Scrum* em ambientes DDS

Existe uma ideia que o *Scrum* pode ser utilizado em qualquer contexto desde que, a equipa trabalhe em prol de um objetivo comum.

Por isso, pode ser utilizado num ambiente em que o desenvolvimento é feito de forma distribuída.

Esta possibilidade, traz uma série de fatores importantes, como a disponibilidade de mão-de-obra, custos mais baixos, uma maior acessibilidade e evolução dos recursos de comunicação e revolução das ferramentas de desenvolvimento.

Como o *Scrum* é uma *framework* de práticas de gestão é necessário que também se adote um conjunto de práticas e técnicas que irão permitir que as entregas correspondam aos padrões de qualidade e adotá-las ao ambiente distribuído.

3.3.1 Papéis e Organização da Equipa:

No *Scrum* tradicional temos os seguintes papéis: o *product owner, o scrum master* e a equipa, contudo num contexto distribuído esses papéis precisam de se comportar um pouco diferente o que leva a criação de novos papéis no intuito de melhorar a integração das equipas tais como: representante do cliente e representante da equipa.

***Product Owner*:** a pessoa responsável por gerir o orçamento, o plano de projeto, criar a lista de requisitos e proporcionar à equipa todas as informações necessárias relacionadas com as suas expetativas sobre o *software*. Por ter um papel essencial no projeto, é necessário que exista uma *product owner* em cada equipa distribuída, pois desta forma, é mais fácil solucionar as dúvidas que cada equipa possa ter, de modo a não prejudicar a produtividade.

Scrum Master: é o responsável pelo processo *Scrum*. É ele que garante que este está a ser aplicado corretamente no projeto e que todos os elementos sigam as suas práticas e regras.

Este papel, no contexto distribuído é muito importante, pois como os elementos estão dispersos é possível colocarem algumas práticas importantes do *Scrum* de lado, por isso, cabe ao *scrum master* acompanhar o processo de desenvolvimento para que isso não ocorra (Luz and Teófilo, 2009).

Equipa de desenvolvimento: é responsável pelo desenvolvimento do projeto. As equipas têm de trabalhar coletivamente se auto-organizar e organizar as tarefas necessárias para alcançar o objetivo da *Sprint*.

Com as equipas distribuídas é necessário um maior esforço para que todos os elementos interajam e comuniquem (Luz and Teófilo, 2009).

Representante da equipa de desenvolvimento: com a dificuldade de gerir os obstáculos e as dependências entre as equipas distribuídas é importante a criação deste papel para representar a equipa.

Este papel deve ser ocupado pelo elemento mais experiente da equipa, que deve ser responsável por interagir com os outros representantes sempre que for necessário. O representante da equipa é aquele que a vai representar a equipa nas reuniões do *Scrum* em que não seja obrigatório a presença de todos elementos (Sureshchandra and Shrinivasavadhani, 2008).

Ele é responsável por interagir com os outros representantes da equipa e com o representante do cliente sempre que seja necessário liderar.

Representante do cliente: ajuda a transferir conhecimento do domínio do negócio do cliente para os desenvolvedores e assim melhorar a comunicação.

Este papel é útil quando o *product owner* não está disponível para responder a perguntas e assim ele pode tomar as decisões em seu nome, ou seja, serve como interface. Contudo, quando surgem questões que ele não sabe responder, o mesmo é responsável por procurar a informação com o *product owner* de modo a não atrapalhar a produtividade da equipa. (Layman et al., 2006); (Paasivaara et al., 2009).

As empresas que, normalmente, trabalham com equipas no mesmo espaço físico encontram dificuldades, quando começam a trabalhar de forma distribuída. Por isso, um ponto

importante é a organização das equipas que pode ser feita de forma errada e podem trazer alguns problemas tais como: mudanças constantes nos requisitos como consequência de distorções durante a comunicação e também a falta de motivação das equipas distribuídas que não se sentem parte do projeto.

De forma a amenizar estes obstáculos enfrentados pela distribuição das equipas, o primeiro passo, para estruturar o projeto é a colocação dos membros em cada equipa. Ou seja, é mais interessante construir equipas que possuam todas as especialidades e fazer com que todos os elementos interajam (Shrinivasavadhani and Panicker, 2008).

3.3.2 Práticas do *Scrum* para Apoiar Projetos Distribuídos.

Esta seção é uma das principais contribuições deste trabalho, pois será discutido como as práticas do *Scrum* devem ser adaptadas para serem aplicadas num ambiente distribuído. Sendo elas as seguintes:

Pré-game: é a prática realizada no início do projeto, em que o *product owner* se reúne com os interessados para definir o objetivo do projeto e a lista inicial das funcionalidades. Para isso, ele realiza uma reunião com os interessados do projeto para criar e priorizar as *user stories* que correspondam ao negócio. No fim desta etapa de planeamento os elementos aproveitam esse momento juntos para estimar as *user stories* numa prática chamada de *planning poker* (Cohn, 2005).

Contudo, a maioria das reuniões entre as equipas distribuídas apresentam problemas por se realizarem de forma distribuída como: diferença de fusos horários, diferenças culturais e linguísticas que podem causar silêncio em alguns participantes (Paasivaara et al., 2009).

De modo a lidar com tais problemas enfrentados pelas equipas distribuídas, torna-se necessário realizar alterações nesta prática.

Na primeira reunião, que ocorre entre todos os elementos do projeto, não existe muitos problemas uma vez que, a maioria dos elementos encontram-se na mesma localidade, o que torna mais fácil estarem todos presentes fisicamente. No caso de alguns participantes se encontrarem noutra localidade, podem participar por videoconferência ou viajam para participar na reunião presencialmente.

Quando as reuniões distribuídas são marcadas deve-se ter em conta uma característica, o fuso horário dos elementos do projeto. Outro ponto importante são as ferramentas que permitem que os elementos interajam e se comuniquem oralmente e de forma escrita para que possam realizar as atividades colaborativas como: criar e descrever as *user stories,* definir o grau de importância de cada item e priorizar as *user stories*.

Depois da reunião as equipas recebem a lista com itens criados e priorizados que vão ser analisadas e posteriormente, na próxima reunião, com o *product owner*, essas dúvidas serão esclarecidas.

Quando um encontro chega ao fim, os elementos reúnem se para analisar as *user stories* através do *planning poker*, sendo necessário a participação de todos os elementos, incluindo os elementos das equipas distribuídas.

***Sprint Planning**:* Esta prática é realizada no início de cada sprint, nela o *product owner* define o objetivo da mesma e negoceia com os elementos, *selected product backlog.*

O *Sprint Planning* é dividido em duas partes. Na primeira, o *product owner* reúne-se com todos os elemento da equipa ou pessoalmente com os representantes. Também poderá ser por videoconferência onde descreve as prioridades das *user stories* e esclarece as dúvidas sobre cada uma delas.

Na segunda parte, cada equipa vai realizar a sua própria reunião. Para evitar dependências, as *user stories* serão divididas pelas equipas e cada uma delas fica responsável por dividi-las em tarefas.

Para que seja possível realizar estas duas reuniões de forma distribuída e adequada é necessário um ambiente virtual que forneça os recursos que permitam a comunicação oral e escrita e que permita aos elementos quebrarem as *user stories* em tarefas distribuídas e em tempo real.

Daily Meeting: Esta prática, consiste na realização de um acompanhamento diário do projeto que possibilita aos elementos da equipa sincronizarem as suas tarefas. Contudo, quando esta prática é aplicada num contexto distribuído, surgem alguns desafios tais como: dificuldade em agendar a reunião que ocorre normalmente por email, os elementos da equipa não aparecerem diariamente á reunião no horário combinado.

A adaptação desta prática consiste num conjunto de recomendações que podem ser aplicadas pelos membros das equipas distribuídas de modo a tratar os principais pontos do *daily meeting*.

Esta abordagem sugere agendar o horário da reunião, compartilhar informações sobre o projeto, controlar a duração desta, organizar a ordem dos membros antes da reunião e salvar as informações da mesma.

Outra recomendação da abordagem é a partilha de informações importantes do projeto em tempo real para os elementos da equipa, como por exemplo: *taskboard* e a *sprint burndown*.

Também é necessário fornecer recursos que permitam aos elementos comunicar, tais como áudio, vídeo e *chat.* O áudio e o vídeo possibilitam uma maior iteração entre os elementos e o *chat* ajuda a superar as questões do idioma e em casos de problemas tecnológicos No entanto, para auxiliar, também é necessário um ambiente virtual com diversos recursos integrados.

Assim, temos um conjunto de recomendações definidas pela abordagem proposta para auxiliar a realização do *daily meeting* distribuído.

Scrum of Scrum: O objetivo desta prática é dar suporte a situações em que a equipa é muito grande e necessita de ser divida em várias que precisam interagir constantemente em prol do progresso do projeto. O foco desta pratica é similar ao *daily metting* só que em uma escala maior. Enquanto, o *daily meeting* é mais viável para equipas pequenas por ser uma reunião diária curta, o *scrum of scrum* visa realizar uma reunião mais especializada com o objetivo de manter as equipas atualizadas em relação aos acontecimentos desde a última reunião.

O encontro ocorre diariamente mas só uma vez por semana devido a distância geográfica das equipas. Neste encontro temos a participação do representante de cada equipa e é debatido o que foi discutido no *daily meeting* das mesmas, para que as atividades sejam sincronizadas.

Para agilizar este encontro e torná-lo mais produtivo, a abordagem proposta sugere que cada representante crie antes da reunião um quadro com o resumo de todos os fatos importantes que aconteceram na sua equipa durante a semana.

Sprint review: esta prática consiste numa reunião que acontece no fim de cada *sprint* os elementos da equipa mostram as *user stories* concluídas ao *product owner* (Schwaber, 2004).

Tal como no *sprint planning,* esta reunião ajuda a melhorar a comunicação e a coordenação entre os elementos aumentando a coesão da equipa e promovendo uma maior

iteração entre os membros e o *product owner*. Esta prática aumenta a viabilidade do progresso do projeto e melhora o acompanhamento do trabalho das equipas distribuídas.

Quando esta reunião ocorre de forma distribuída algumas mudanças precisam de ser feitas para reduzir os problemas enfrentados pelas equipas.

A *sprint review* pode ser realizada de duas formas. No primeiro caso em que cada equipa encontra-se numa única localidade. A opção ideal é a participação de todos os membros, permitindo que cada equipa apresente, ao *product owner,* tudo o que foi realizado na *sprint*. É importante que os representantes e o *scrum master* permaneçam o tempo todo.

Contudo, se cada equipa se encontrar em localidades diferentes devido á dificuldade em realizar uma reunião distribuída com todos os elementos presentes, a solução seria a participação dos representantes das equipas.

Pois, nem todas as equipas apresentam um orçamento que permite a deslocação dos elementos, o que leva á existência de uma terceira opção, que sugere ter somente a equipa central, em que apresenta o que foi criado no *sprint* com os representantes de cada equipa apenas como ouvintes.

Nas duas últimas opções é necessário que, após as reuniões sejam partilhados os comentários elaborados pelo *product owner* a todos os restantes elementos da equipa.

Contudo, é de referir que na ultima opção é necessário marcar um encontro entre os representantes de todas as quipás para que as equipas possam mostrar o que desenvolveram.

Sprint retrospective: Esta pratica sugere que deve haver um *sprint retrospective* por equipa, para armazenar as informações levantadas em cada reunião. Estas devem ser partilhadas por todos os elementos, permitindo que analisem as melhorias sugeridas por outras equipas.

No caso das equipas espalhadas por diversas localidades é necessário a utilização de recursos tecnológicos como: videoconferência e um quadro virtual que se possa atualizar em tempo real (similar a *taskboard*) permitindo que os membros levantem os pontos positivos e negativos da *sprint.*

Este trabalho, também, menciona a importância da realização de uma reunião entre o *scrum master* das equipas, de forma parecida ao *scrum of scrum* para que sejam discutidos os comentários do *sprint retrospective* de cada equipa e com isso se conseguir obter um alinhamento entre todas as equipas.

Esta reunião pode ocorrer no dia posterior ao *sprint retrospective* das equipas e é crucial que os *scrum masters* verifiquem os pontos levantados pelas outras equipas antes dessa reunião de modo a optimizar a duração da mesma.

3.3.3 Soluções Adotadas para o Desenvolvimento Distribuído com o *Scrum*

A adoção do desenvolvimento de *software* distribuído tem crescido e desempenhado um papel crucial para as empresas que querem competir no mercado global. Contudo, as características de um dos princípios fundamentais de qualquer metodologia ágil de *software* é a importância da comunicação entre as várias pessoas envolvidas no desenvolvimento de *software*. Além disso, métodos ágeis colocam um grande prémio na melhoria da comunicação através da comunicação face-a-face.

A tabela 1, ilustra alguns dos problemas que as equipas podem encontrar na adoção do *Scrum* em ambientes de *software* distribuídos.

A primeira coluna apresenta os problemas enfrentados ao longo do desenvolvimento distribuído. É de salientar que muitos desses problemas não existiriam numa equipa de *Scrum* tradicional como descreve a segunda coluna.

A terceira descreve as ações adotadas para a equipa contornar esses problemas.

Tabela 1 – Soluções Adotadas para o Desenvolvimento Distribuído com o *Scrum*

Problemas	*Scrum*	Soluções
Equipa distribuída geograficamente	Aspeto estritamente relacionado com a presença física da equipa nas atividades de monitoramento e feedback. Em geral, reuniões rápidas e diárias com toda a equipa de forma a identificar e corrigir possíveis impedimentos e/ou erros no processo de desenvolvimento de *software*.	O desafio de proporcionar a efetiva iteração entre os elementos da equipa geograficamente distribuída foi obtido através do uso de ferramentas de comunicação como videoconferência. Estes recursos, aliados ao comprometimento, aspeto importante no *Scrum*, possibilitam uma boa iteração entre os membros da equipa.
Reuniões diárias	O *Scrum* prevê reuniões diárias de duração de quinze minutos com todos os elementos da equipa.	Esta característica torna-se complicada quando é aplicado numa equipa geograficamente distribuída. Contudo, este problema pode ser solucionado com o uso de ferramentas de comunicação como videoconferência, groupware entre outras
Visibilidade das atividades	Todos os elementos da equipa podem ver o quadro de atividades, onde podem visualizar e manipular o quadro de atividades de forma rápida e prática	O uso de uma planilha compartilhada com todas as atividades planeadas utilizadas por cada elemento para atualizar o estado das suas atividades mantendo assim a transparência em toda a equipa.
Atribuições do Scrum Master	Tem como função auxiliar a equipa e o *product owner.*	Com a equipa geograficamente distribuída, as atribuições e as responsabilidades do *scrum master* aumentam. Pois, além de estar atento à gestão dos impedimentos reportados pela equipa, também precisa estar atento à iteração entre os elementos das mesmas.

3.4 Mapeamento das Práticas do *Scrum* vs Práticas do DDS

Ao longo deste capitulo foram caracterizadas as praticas do desenvolvimento distribuído de *software*, assim como, as praticas do Scrum adaptadas a este tipo de desenvolvimento

Para uma melhor compreensão foi elaborado um mapeamento. Este mapeamento, tabela 2, tem como objetivo demonstrar como as práticas do DDS influenciam as práticas do Scrum adaptadas. Do mesmo modo, tenta demonstrar os benefícios que este mapeamento pode trazer para as equipas distribuídas, para conseguir gerir da melhor forma um projeto distribuído adaptado ao *Scrum*.

Tabela 2 - Práticas do *Scrum* vs. Práticas do DDS

Práticas *Scrum*/ Praticas DSD	**Iniciar o projeto com todos os elementos juntos**	**Fornecer formação sobre os processos e as tecnologias que serão utilizadas no projeto**	**Visitas frequentes**	**Meios de Comunicação**	**Envio de alguém mais experiente durante mais tempo para as equipas distribuídas**
Pré-Game	Iniciar o projeto com todos os elementos, de forma presencial na reunião	A realização da formação deve ocorrer no início do projeto	————	Proposta de um ambiente propício para a reunião	A pessoa enviada deve participar na reunião da equipa visitada, na qual pode servir de representante do *Product Owner*
Spring Planning	Igual ao *pré-game*	Igual ao *pré-game*	Referente ao representante da equipa na qual deve participar na reunião	Igual ao *pré-game*	Igual ao *pré-game*
Daily Meeting	Igual ao pré-game	Igual ao pré-game	Igual ao Spring Planning	Implementação de ferramentas para a realização da reunião com os recursos necessários para todos os elementos interagirem de melhor forma	Igual ao pré-game
Spring Review	Igual ao pré-game	Igual ao pré-game	Igual ao Spring Planning	Igual ao pré-game	Igual ao pré-game
Spring Retrospective	Igual ao pré-game	Igual ao pré-game	Igual ao Spring Planning	Igual ao pré-game	Igual ao pré-game
Scrum of Scrum	Esta reunião só é realizada quando necessária	Igual ao pré-game	Igual ao Spring Planning	Possível utilização de um ambiente implementado para o daily meeting	Igual ao pré-game

3.5 Conclusão

Este capítulo apresenta os pontos que podem ser aplicados num projeto em que a equipa de desenvolvimento de *software* adaptando-se ao método ágil *Scrum*.

Primeiro foi definido o desenvolvimento distribuído de *software* assim como o levantamento das suas características e práticas.

Após analisar os principais desafios proporcionados pelo *Scrum*, observou-se que existem alguns pontos críticos comuns às duas abordagens como a necessidade de comunicação, um ambiente e suporte para facilitar o trabalho colaborativo e a utilização de meios para promover a motivação e a coesão da equipa.

A distribuição do processo do desenvolvimento de *software* amplia os problemas relacionados ao desenvolvimento, o que cria desafios ao desenvolver a distância física, dispersão temporal e diferenças culturais.

Relativamente às principais práticas do *Scrum*, as adaptações definidas visam melhorar a realização das reuniões de modo a minimizar a distância geográfica entre os elementos distribuídos e a comunicação.

Através da análise das práticas do *Scrum* e do mapeamento que foi elaborado com elas pode-se concluir que as que visam solucionar os problemas de comunicação são o *sprint planning* e *scrum of scrum*.

Relativamente à minimização da distância geográfica a prática que apoia este objetivo é o *daily meeting*.

Tendo em conta os papéis, pode-se concluir que eles foram adaptados de forma a minimizar os problemas encontrados, como a redução de custos em despesas de viagens através da seleção de um representante da equipa e com o objetivo de intensificar a interação, colaboração e comunicação entre as equipas e com isso melhorar o acompanhamento do projeto.

Nos dias de hoje, as organizações têm procurado a implementação dos processos produtivos para melhorar a qualidade e reduzir custos. Além disso, os mercados procuram por uma maior agilidade e flexibilidade no desenvolvimento de *software*, por isso a adaptação do *Scrum* no ambiente distribuído é uma das soluções

CAPITULO 4 - ESTUDO DE CASOS

4.1 Introdução

Neste capítulo, vão ser apresentados dois estudos de caso, em que ambos são desenvolvidos pelo Centro de Computação Gráfica, no domínio de investigação aplicada.

Este centro é uma associação sem fins lucrativos pertencente ao sistema científico português. A sua missão é a investigação científica e tecnológica aplicada na área da computação gráfica e sistemas de informação.

Ela posiciona-se desde a sua génese como o interface entre as fontes do saber (Universidades) e a economia (Empresas) através de projetos abertos, vocacionados para prestação de serviços e transferência de tecnologia nas áreas análogas à computação gráfica, tecnologias de informação, comunicação, eletrónica e suas aplicações.

Os estudos de casos analisados, neste capítulo têm a seguinte designação: Murallas Digital e CITT.

O projeto designado por Murallas Digital é co-financiado pela União Europeia através do fundo europeu de desenvolvimento regional num programa de cooperação transfronteiriça Espanha-Portugal, na qual, o Centro de Computação Gráfica assume neste projeto o papel de parceiro tecnológico para desenvolvimento de soluções técnicas a desempenhar e implementar. Alem das atividades de desenvolvimento para a "rota das muralhas" também é responsável pelo suporte de multimédia móveis para a visita no terreno e aplicações interativas para a disponibilização da informação relativa ao Murallas.

O Centro de Computação Gráfica, neste projeto apoia em particular os municípios portugueses com orientações técnicas e controlo nas suas tarefas de recolha e produção de conteúdo.

O projeto CITT, *Cross Independent Testing Tool*, tem como objetivo é a criação de uma ferramenta de testes de automatização num *I&DT* em co-promoção.

Relativamente ao referido, o Centro de Computação Gráfica tem um papel importante no levantamento de requisitos e no desenvolvimento da ferramenta.

Todavia, este capitulo tem dois objetivo: analisar estes projetos e efetuar uma análise comparativa entre ambos através das suas caraterísticas e método e a simulação do projeto Murallas com aplicação do método *Scrum* em ambientes de *software* distribuído.

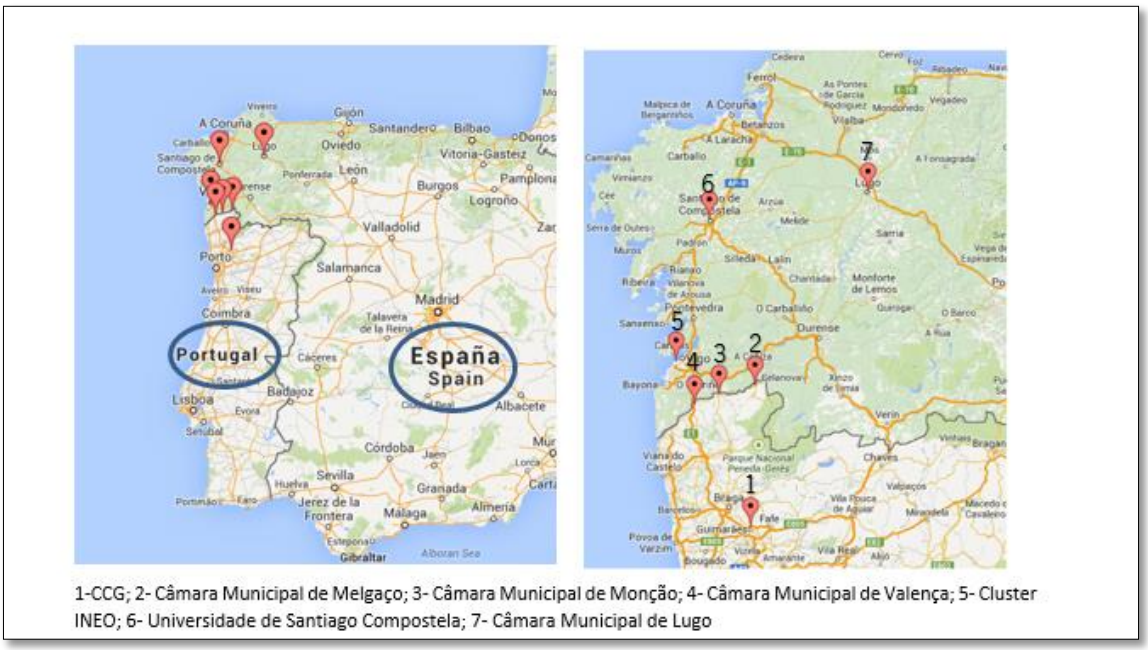
4.2 Estudo de Caso: Murallas Digital

No Murallas Digital, o principal objetivo é a criação de uma rede de cidades amuralhadas da Galiza e norte de Portugal, que apostam na gestão conjunta para a valorização do património histórico-arqueológico através das novas tecnologias

Este projeto caracteriza-se pelo seu caracter inovador, uma vez que porá em marcha um novo conceito de comunicação, onde colocara o valor patrimonial histórico-arqueológico de uma forma mais dinâmica e de assimilação mais rápida, através do uso das novas tecnologias da informação e a comunicação.

Murallas Digital é composto por sete parceiros (Câmara Municipal de Lugo, Universidade de Santiago Compostela, Câmara Municipal de Melgaço, Câmara Municipal de Monção, Câmara Municipal de Valença, a Cluster INEO e o Centro de Computação Gráfica) estando eles alocados entre Portugal e Espanha, como ilustra a figura 9.

É de salientar que o parceiro principal é a Camara Municipal de Lugo e os restantes parceiros são responsáveis pela garantia e a viabilidade das atuações e a capacidade de gestão.



1-CCG; 2- Câmara Municipal de Melgaço; 3- Câmara Municipal de Monção; 4- Câmara Municipal de Valença; 5- Cluster INEO; 6- Universidade de Santiago Compostela; 7- Câmara Municipal de Lugo

Figura 9 - Parceiros do Murallas Digital (Fonte: Elaborado por mim)

Após uma análise geral do projeto verificou-se que a localização é relevante uma vez que se trata de dois países partindo do princípio que todos os elementos da equipa trabalham de forma remota, independentemente do local ou do fuso horário a que pertencem tratando-se assim de um ambiente distribuído.

Depois de uma revisão de literatura elaborada para o desenvolvimento distribuído de *software* e uma análise geral do projeto Murallas, as informações adquiridas, sobre este projeto, foram obtidas através da aplicação de um questionário e da realização de entrevistas, elaborados com objetivo de entender o projeto realizado na qual detetaram alguns problemas gerais.

Alguns desses problemas foram detetados entre os parceiros, como a circunstância de não trabalharem de forma conjunta, a inexistência de um planeamento, o facto de o idioma da aplicação ser português/castelhano e existirem erros de tradução devido à falta de comunicação. Relativamente ao conhecimento que cada parceiro continha, uma vez que a maioria deles, tratam-se de Câmaras Municipais, ou seja, não existem engenheiros informáticos mas sim técnicos que têm uma forma de pensar diferente.

O foco principal desta análise é a equipa de desenvolvimento do Centro de Computação Gráfica no desenvolvimento de soluções técnicas a desempenhar e a implementar neste projeto.

Tendo em conta esta equipa e analisando todas as informações obtidas nas entrevistas através da elaboração de um questionário, verificou-se, que mesmo dentro da associação, se trata de um desenvolvimento distribuído, sendo ele composto por duas equipas designadas por *Engineering Process Maturity e Quality* (EPMQ) e *Urban and Mobile Computing*.(UMC)

A equipa EPMQ é constituída por uma equipa multidisciplinar que tem como objetivo o desenvolvimento de aplicações para a rota das muralhas e a elaboração de aplicações interativas para a disponibilização da informação.

Relativamente à equipa UMC esta é responsável por proporcionar às pessoas experiências novas e mais completas, através de novas aplicações computacionais, a monitorização de espaços públicos em tempo real mais a respetiva recolha de dados, a fusão de dados, a construção cooperativa de mapas e a visualização dinâmica das cidades.

4.2.1 Características do Desenvolvimento de Distribuído *Software* no Murallas

Diversas situações de desenvolvimento de *software* são possíveis, cada um com diferentes problemas a serem geridos e soluções a serem estudadas e aplicadas. De forma a evidenciar os detalhes fundamentais do projeto em estudo, a tabela 3 apresenta as características do desenvolvimento distribuído de *software* na equipa de desenvolvimento do Centro de Computação Gráfica.

Tabela 3- Características do DDS

Características do Desenvolvimento Distribuído de *Software*		
Agrupamento	EPMC: – Gestor de projeto - Analista - *Design* - Programador	UMC: - Gestor de projeto - *Design* - Programador
Distância física	Centralizada.	
Separação temporal	Inexistente	
Cultura Regional	Sem diferenças	
Idioma	Igual	
Diferença dos locais	Praticamente Nenhuma	
Cultura organizacional	Sem diferenças relevantes	
Infra-estrutura	Ferramentas e técnicas pouco diferentes	
Relação de Negocio	Parceria entre as organizações	

Uma grande vantagem deste projeto, em relação ao desenvolvimento distribuído de *software* foi a inexistência de grandes diferenças relativas as equipas envolvidas, o idioma era o mesmo, a cultura regional é parcialmente igual e não havia diferenças relevantes entre os locais.

Após análise destas características foram focadas duas características: a distância física e o agrupamento. A distância física porque este critério define quão distante estão as pessoas envolvidas e as suas respetivas áreas de negócio e o agrupamento por ser uma questão multidimensional composta por três aspetos: quantidade de pessoas, quantidade de grupos e os papéis exercidos pelas pessoas.

O número de pessoas do CCG que participam no projeto Murallas são oito, nas quais, estão divididas em dois grupos: o EPMQ e o UMC.

A equipa EPMQ é composta por uma analista, um programador e um *design*, enquanto, a equipa UMC é constituída por um programador e um *design*.

Relativamente ao gestor de projeto este é o mesmo para ambas equipas.

Verificou-se que a equipa de desenvolvimento encontram-se com a mesma localização física, pois toda a equipa esta instalada no mesmo edifício, o CCG.

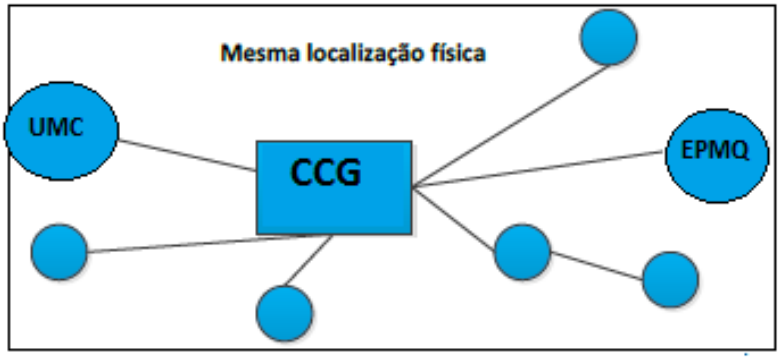


Figura 10- DDS Centralizado (Fonte: Elaborado por mim)

Ou seja, trata-se de uma equipa de desenvolvimento centralizada (co-localizada), pois esta localizada no mesmo espaço físico como ilustra a figura 10

A figura 11, demonstra as equipas de desenvolvimento localizadas no mesmo espaço físico e como esta dividida em duas sub-equipas, a equipa do EPMQ e o UMC.

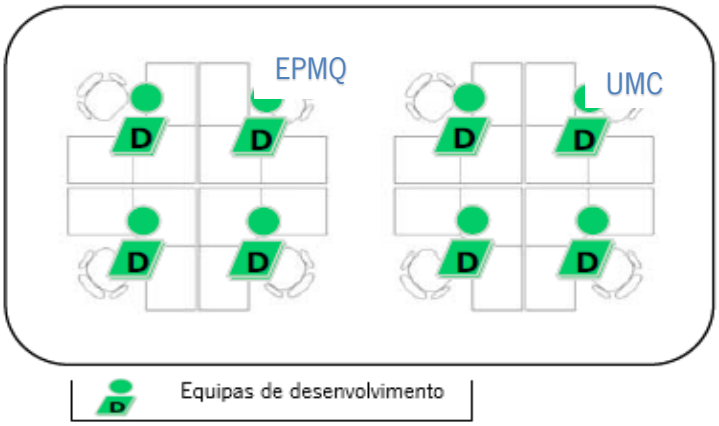


Figura 11- Equipas subdivididas (Fonte: Elaborado por mim)

Cabe salientar que a distribuição da equipa de desenvolvimento não leva em conta a localização dos outros parceiros do projeto.

Contudo, estas demonstraram a existência de alguns problemas no desenvolvimento.

No caso do agrupamento, um problema encontrado foi a dificuldade do chefe de equipa monitorizar as duas equipas, por exemplo, a comunicação entre as mesmas. Apesar de existirem reuniões quinzenais a comunicação entre as equipas era muito pouca. Muitas vezes, a forma de comunicação era por *emails* o que causava um entendimento equivocado do que se quis dizer por ser uma forma ambígua de comunicação.

Em relação à padronização, como o padrão de código e métodos aplicados, cada grupo decide a forma de trabalho que acha mais interessante. Outro problema da infra-estrutura foi a diferença de sistemas operacionais e versões de ferramentas, uma vez que uma equipa é para desenvolvimento mobile, enquanto a outra era a responsável pelo *BackOffice* e pela integração do *webservice*.

Relativamente a caracterização do projeto, este transcorreu sem problemas, uma vez que as características, como o idioma e a cultura organizacional, pouco influenciaram o projeto, o que impediu não só uma análise completa das características, como também a observação dos problemas nas atividades dos papéis de gestão em foco.

4.2.2 Murallas Digital e o *Scrum*

O desenvolvimento distribuído de *software* ainda deixa muitas dúvidas quanto à sua real eficácia, como, por exemplo: as equipas distribuídas têm a mesma eficiência que equipas centralizadas? A comunicação distribuída é tão eficiente quanto à comunicação síncrona? Os processos de *software* atuais são capazes de lidar com as características do desenvolvimento distribuído enquanto garantem a qualidade do produto? Neste contexto é possível visualizar os métodos ágeis como solução para auxiliar no desenvolvimento distribuído de *software*, aumentando o ritmo no desenvolvimento, organização das equipas de desenvolvimento e solucionar os problemas encontrados nas práticas, através da sua adoção. Como exemplo, temos um método baseado em princípios ágeis, denominado *Scrum*, o qual foi criado para acrescentar foco, comunicação, clareza e transparência para o desenvolvimento.

Com base nisso, esta secção apresenta a simulação de uma implementação do *Scrum* num ambiente de *software* distribuído, para auxiliar na gestão das equipas distribuídas, nas

ferramentas e técnicas apropriadas para gestão, comunicação e organização das equipas de desenvolvimento.

Equipa e papéis: A equipa *Scrum* possui três papéis principais: o *scrum master*, *product owner* e a equipa de desenvolvimento, contudo, no contexto distribuído, estes papéis são adaptados, assim como, a adoção dos seguintes papéis como: o representante do cliente e da equipa de desenvolvimento.

O *scrum master* é o responsável por garantir que o processo será compreendido e acompanhado. Pode pertencer a várias equipas.

Como o projeto Murallas é um projeto co-financiado pela União Europeia, por vezes o contacto constante preponente do projeto pode ser complicado e demorado, o que é necessário encontrar um elemento responsável pelo interface de comunicar com o preponente. Sendo ele o responsável pela equipa.

O *product owner* é o elemento que participa ativamente no projeto esclarecendo as dúvidas e validando o produto em cada iteração. O coordenador das equipas pode ser considerado o *product owner*.

Relativamente à equipa de desenvolvimento, esta é responsável por desenvolver as tarefas definidas. Uma vez que as equipas estão de um certo modo distribuídas, a solução aplicada é selecionar um representante que vai representar a equipa em reuniões, em que não seja necessário a presença de todos os elementos.

Contudo, a equipa de desenvolvimento mantém os papéis de um processo tradicional como: um analista, um programador, um *design*, um testador entre outros.

Relativamente ao representante da equipa ele pode exercer qualquer função, ou pode mesmo variar de reunião para reunião de forma a todos os elementos conseguirem ser o representante pelo menos uma vez, ou, pode ser o elemento com mais capacidades.

Práticas: O Murallas apresenta um cenário em que as equipas trabalham no mesmo espaço, mas de forma distribuída. As adaptações necessárias nas práticas de *Scrum* para este cenário é a adoção de um modelo defendido pelo *Sureshchandra*. Este consiste em quatro estados: avaliação, iniciação, transição e pronto.

No estado de avaliação, é decidido se o projeto deve ser realizado de forma distribuída. Quando a decisão é tomada, a iniciação e a transição são os estados intermédios em que o projeto

é adotado aos problemas da distribuição e atinge o estado pronto quando considera que o projeto já se adaptou a esse novo contexto.

Neste estado, a organização determina o grau de agilidade e o grau de distribuição para verificar se o projeto é mesmo viável ser executado de forma distribuída.

No estado de iniciação as práticas do *scrum* aqui aplicadas são o *sprint planning*, ou seja, uma reunião é elaborada com a presença de cada representante. Aqui, é feito o detalhamento das *user stories* e das estimativas. Neste estado também temos a *sprint review* em que é elaborada a apresentação do produto para a equipa. O *product owner* pede, antes da reunião, que as equipas distribuídas mostrem o que realizaram. Esta demonstração também pode ser feita através do uso de videoconferência evitando assim gastos em deslocações.

Ao fim da reunião as melhorias identificadas pelo *product owner* são repassadas para os representantes de cada equipa distribuída.

Na transição, os representantes das equipas voltam para o seu espaço de trabalho. Neste estado existe um contexto de mudança, em que os elementos todos do projeto podem participar como ouvintes.

No último estado, o pronto, todos os elementos de cada equipa participam na reunião por utilização de uma ferramenta de comunicação e o elemento representante da equipa distribuída apresenta o que a sua equipa fez.

Por fim, a última prática destacada neste modelo é a *sprint retrospective.* Cada equipa distribuída realiza uma reunião localmente e depois transmitia às outras equipas o que foi discutido nessa reunião.

Na transição, as reuniões locais continuam, enquanto, no estado pronto as reuniões são planeadas com antecedência, o planeamento da reunião é feito um dia antes e com a participação de todos os elementos.

Outra sugestão é realizar uma reunião *scrum of scrum* após cada uma das reuniões *Scrum*. Deste modo, cada equipa distribuída teria o seu próprio *sprint planning, sprint review* após cada uma dessas reuniões teríamos o *scrum of scrum* para debater o que foi discutido nas reuniões de cada equipa com a presença do *scrum master*.

Relativamente ao ciclo de vida do *Scrum* adaptado ao ambiente distribuído este pode ser o seguinte: a forma de trabalho que a equipa pode utilizar é que no início de cada *sprint* as *user stories* são definidas juntamente com o *product owner* e realizadas as suas priorizações obtendo assim, o *selected product backlog.*

Com ele definido, as reuniões são realizadas para a discussão das funcionalidades e definição das atividades necessárias para a implementação. Esta informação deve ficar disponível numa planilha compartilhada por via web, com a informação do responsável, com o estado e o tempo estimado para que seja possível analisar e atualizar o andamento das atividades.

Quando um sprint termina, uma reunião de *sprint review* e *retrospective* deve ser realizada para que sejam identificados os erros e os acertos, assim como os pontos que precisam de ser melhorados para a próxima sprint.

Contudo, uma vez por semana deve ser feita uma reunião *Scrum of Scrum* em que o *scrum master* e o representante de cada equipa se encontram com o *product owner* e com os clientes. Neste caso é com os parceiros para resolver certos impedimentos que surjam ao longo do desenvolvimento.

A técnica do *Scrum* que se pode utilizar é o *Planning Poker* que ocorre durante a *sprint planning.* Nesta fase é o momento em que toda a equipa deve entrar em consenso relativamente as estimativas das *user stories*.

O objetivo do *planning poker* aqui é simular sessões de estimativas distribuídas, de modo a que todos os elementos do projeto possam discutir e argumentar como se estivessem reunidos presencialmente. Por isso, é necessário criar um ambiente virtual com recursos multimédia, por exemplo: o uso do *skype*.

Também é possível realizar um *daily scrum* sem obedecer o *face-to-face*, com o uso de ferramentas de comunicação.

Relativamente ao *daily scrum meeting* as reuniões diárias devem ter uma duração de 15 minutos e precisam de ser objetivas. Por vezes, as reuniões podem não ser muito objetivas devido ao uso de comunicação assíncrona. Por exemplo, o uso do email não permitiria o prolongamento da reunião, assim a comunicação diária é prejudicada o que torna mais vantajoso utilizar meios de comunicação síncrona.

Relativamente à equipa de desenvolvimento, uma das implementações a efetuar para melhorar a qualidade do *software* é aplicação da programação em pares distribuída. Esta prática combinada com o *Scrum* aumenta consideravelmente a produtividade da equipa.

Esta prática ajuda a manter o foco no desenvolvimento e auxilia no entendimento do problema a ser resolvido. A programação em par distribuída ajuda à promoção do trabalho e da comunicação dentro da equipa distribuída.

Recursos Tecnológicos: Pelo facto das equipas não estarem no mesmo espaço físico, um recurso que é mencionado em vários trabalhos são as ferramentas de comunicação, o chat, a videoconferência (por exemplo: o *skype*) e o *email*. Estes recursos são fundamentais para tentar reduzir a falta de comunicação Face-to-face permitindo assim diversas reuniões do *Scrum.*

As ferramentas de comunicação podem ser agrupadas de acordo com dois critérios: tempo e localização. O critério tempo é utilizado para indicar se a equipa está a trabalhar ao mesmo tempo (síncrono) ou em horários diferentes (assíncrono). O critério de localização indica se a equipa esta a trabalhar no mesmo local de modo disperso.

A tabela 4 apresenta uma relação das ferramentas agrupadas de acordo com o tempo e a localização.

Tabela 4 – Ferramentas de comunicação

		Tempo	
		Síncrono	**Assíncrono**
Localização	**Mesmo local**	Reuniões Apresentações	Computadores Compartilhados
	Disperso	Telefone Chat Videoconferência Quadro eletrónico	*Email* Correio de voz Groupware

Relativamente ao acompanhamento das informações do projeto como: distribuição das User Stories, distribuição de tarefas, criação de *burndown charts* são utilizadas ferramentas específicas para a gestão ágil de projetos como: *VersionOne, Jira, ScrumWorks* e TFS.

4.3 Estudo de Caso: CITT

O projeto CITT surge no âmbito de três organizações, a empresa 1, a empresa 2 e o Centro de Computação Gráfica adiante, designadas por consórcio

"*Cross Independent Testing Tool*", pretendem desenvolver um projeto de I&DT em co-promoção, cujo objetivo é disponibilizar uma ferramenta que possibilite uma automatização de testes em *software* desenvolvido em diferentes tecnologias e para diferentes plataformas.

O objetivo é criar uma ferramenta independente de motores de execução que permita evoluir o estado da arte de produção de testes automáticos, conseguindo-se dessa forma um aumento de eficiência nestas tarefas e uma redução de tempo e do desgaste dos recursos.

A solução terá a capacidade de acompanhar o ciclo de desenvolvimento de um *software*, necessitando apenas de ser configurada e parametrizada para que, testes definidos inicialmente para um software, continuem a ser válidos ao longo do seu desenvolvimento, bastando apenas pequenos ajustes que acomodem a evolução do *software*. Desta forma, esta será a única solução no mercado capaz de conseguir uma reutilização dos testes, independentemente da linguagem de programação e do motor de execução de testes.

Assim, tendo como ponto de partida as mais recentes técnicas utilizadas, pretende-se desenvolver uma metodologia inovadora e claramente diferenciadora do que existe, capaz de evoluir as soluções disponíveis no mercado atualmente, que se caracterizam pela sua morosidade, complexidade e rigidez, através de uma ferramenta capaz de reaproveitar o trabalho já realizado.

A complementaridade de competências e de interesses comuns no aproveitamento de resultados de atividades de I&DT, culminou com a criação deste Consórcio.

O projeto "*Cross Independent Testing Tool*" nasce das necessidades sentidas pelos clientes dos promotores, e pelos próprios, em desenvolverem uma plataforma que permita realizar testes de forma mais rápida, eficiente e num menor período de tempo.

O CITT é um projeto em que utiliza dois tipos de métodos de desenvolvimento de *software*, um processo ágil, em que é utilizado o *Scrum* e as suas respetivas vertentes e o desenvolvimento distribuído de *software*.

Na análise geral deste projeto, a recolha de informação foi obtida através de entrevistas, com apoio de um questionário para uma melhor perceção da implementação destes dois processos. Desta análise obteve-se a seguinte tabela:

Tabela 5- Características do DDS no CITT

Características do Desenvolvimento Distribuído de *Software*	
Agrupamento	Equipa Scrum: - Scrum Master -Product Owner - Equipa de desenvolvimento: . Equipa do CCG: *Gestor de projetos *Analista *Gestor de testes *Testador . Programadores
Distância física	Distribuída
Separação temporal	Inexistente
Cultura Regional	Sem diferenças
Idioma	Igual
Diferença dos locais	Relativamente alguma
Cultura organizacional	Sem diferenças relevantes
Infra-estrutura	Ferramentas e técnicas iguais
Relação de Negocio	Parceria entre organizações

A tabela 5, demonstra que neste projeto, no agrupamento, mais propriamente na seleção da equipa, teve-se em conta o objetivo do projeto e os métodos nele aplicado. Ou seja, a aplicação do método *Scrum* para ambientes de *software* distribuído

A equipa *Scrum* do CITT é composta por um *scrum master*, um *product owner* pela equipa de desenvolvimento.

A equipa de desenvolvimento é composta pela equipa do CCG e pelos programadores.

A equipa do CCG é constituída por: um analista, um gestor de projeto, um gestor de testes e um testador. Os programadores são três elementos, em que um elemento é da empresa 1 e os outros dois são subcontratações desta, ao CCG e a empresa 2.

Relativamente ao desenvolvimento distribuído de *software* e tendo em contas as suas características, verificou-se que em termos de distância física, esta era restringida pelo mesmo país mas em cidades diferentes, como ilustra a figura 12.

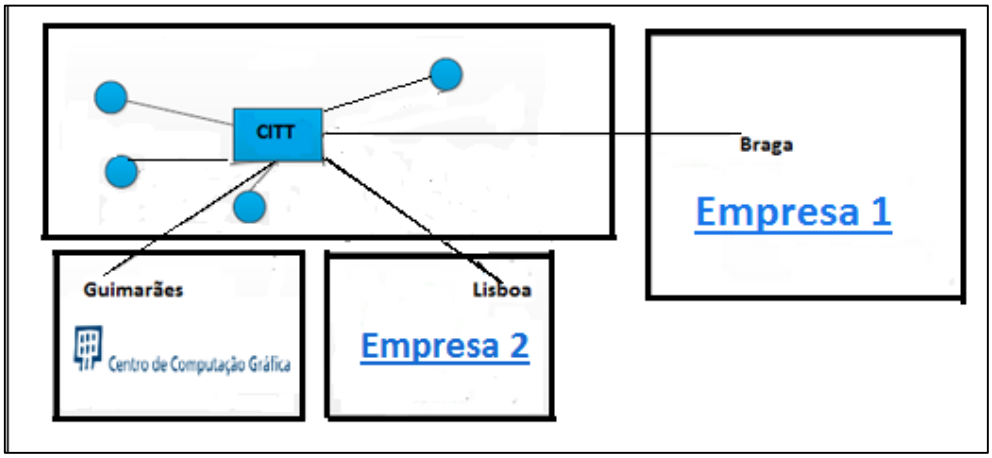


Figura 12- DDS do CITT (Fonte: Elaborado por mim)

Relativamente ao idioma, não existia qualquer problema uma vez que todos falam a mesma língua.

Os meios de comunicação utilizados pela equipa era emails e por *skype*. Em que por algumas ocasiões existiam deslocações para reuniões em que todos os elementos tinham que estar presentes. Outos métodos de comunicação utilizados devem-se ao facto de existir outra metodologia de desenvolvimento aplicada ao *Scrum* que apela à utilização da ferramenta TFS.

Em relação ao negócio trata-se de uma parceria entre organizações, designada por consórcio, em que o objetivo é o mesmo para todos, ou seja, desenvolver o produto de forma eficaz e eficiente.

Tendo em conta os papéis, práticas e valores defendidos pelo *Scrum*, na implementação neste projeto no CCG verificou-se:

Equipa e Papéis: Durante o processo é importante o conhecimento dos responsáveis por cada atividade. O CITT é um projeto diferente pois não tem os papéis definidos como o RUP[6]. Contudo, segue a estrutura de uma equipa *Scrum*, composta: por uma equipa de desenvolvimento, um *product owner* e um *scrum master*.

[6] É uma metodologia para desenvolvimento de *software* com uma estrutura formal e bem definida. No desenvolvimento usa a abordagem, orientação a objetos e é projetado e documentado utilizando a notação UML (*Unified Modeling Language*)

O *scrum master* é representado por um dos elementos da empresa 1. A empresa 2, principalmente, o CCG e a empresa 1 são *product owners*.

A equipa de desenvolvimento esta é composta por um programador de todas as organizações. O papel de analista pertence a um elemento do CCG que têm como função, o levantamento dos requisitos de aplicação de testes que esta a ser desenvolvida, ou seja, na criação das user stories

Neste projeto, o CCG tem outros elementos fundamentais como: o papel de gestor de testes e o testador.

No CITT, todas as organizações implementaram o método *Scrum*, o que levou a equipa do CCG a selecionar um responsável pela equipa que neste caso é o gestor de projetos, que tinha a função participar nas *sprints reviews* quinzenais.

Práticas: Uma das características principais do projeto CITT é a implementação das equipas distribuídas no desenvolvimento de uma aplicação num ambiente ágil.

Relativamente as *sprints*, neste projeto todas elas têm uma duração de quinze dias O *scrum master* e o *product owner* demonstram total disponibilidade para apoiar a equipa de desenvolvimento.

Nas práticas *sprint review* e *sprint retrospective* os problemas existentes são reportados e a solução é encontrada.

De forma, a que todos os elementos da equipa estejam dentro do assunto, é eleito um representante da mesma. Ele está presente nas reuniões e no fim comunica aos outros elementos da equipa.

Uma das práticas do *Scrum* com mais preponderância neste projeto é o *grooming*., que consiste numa reunião importante, pois serve de plataforma entre o *product owner* e a equipa de desenvolvimento sobre o trabalho que ainda falta realizar sobre o produto e para o esclarecimento de dúvidas que surjam ao longo da realização das *user story*. Trata-se de uma reunião diária que tem como duração quinze minutos.

Recursos Tecnológicos: Uma vez que as equipas não se encontram no mesmo espaço físico, as ferramentas de comunicação são fundamentais para a elaboração do projeto.

Neste projeto, uma das ferramentas de comunicação mais utilizada é o *skype*. Como o programador, elemento do CCG, subcontratado neste projeto pela empresa 1 para o desenvolvimento de aplicação, utiliza esta ferramenta para o *daily meetting*.

As ferramentas utilizadas para a gestão ágil do projeto é o programa TFS, o que oferece um sistema de compilação e ferramentas e métricas de planeamento ágil para a gestão de projetos de desenvolvimento de *software*. É através desta ferramenta que são criadas as *user stories.*

Contudo, uma vez que tal método nunca tinha sido aplicada no desenvolvimento de projetos no CCG, os elementos desta organização tiveram que ter formação para um melhor entendimento do método e uma explicação de como funciona a ferramenta, uma vez que estes são responsáveis pelo levantamento de requisitos e pela criação das *user stories*.

4.4 Comparando os Casos

O Murallas Digital e o CITT são dois projetos desenvolvidos pelo Centro de Computação Gráfica e que apresentam características e problemas diferentes, mas ao mesmo tempo algumas semelhanças.

Após analisar estes dois projetos verificou-se que as características gerais de ambos, são as seguintes:

Tabela 6- Características gerais dos projetos

CITT	Murallas
Abordagem ágil	Abordagem tradicional
Portugal – Espanha	Norte e Sul de Portugal
Equipa Scrum e equipa de desenvolvimento de *software* distribuído	Equipa de desenvolvimento de *software* distribuído
Organizações: - Empresa 1 - Empresa 2 - Centro de Computação Gráfica	Parceiros: -Câmara Municipal de Lugo, -Universidade de Santiago Compostela, -Câmara Municipal de Melgaço, -Câmara Municipal de Monção, -Câmara Municipal de Valença, - Cluster INEO; - Centro de Computação Gráfica.
Ferramentas: -Skype; -Correio eletronico -TFS	Ferramentas: -Skype - Correio eletrónico
Reuniões: Diárias	Reuniões: Quinzenais

Após análise da tabela 6 é notável a utilização de abordagens diferentes em ambos os projetos, apesar de aplicarem o mesmo tipo de desenvolvimento, o desenvolvimento distribuído de *software*.

No Murallas Digital, a abordagem utilizada é a tradicional, enquanto, no CITT é uma abordagem ágil.

A primeira constatação desta análise é sobre as diferenças, entre as abordagens.

A principal diferença, neste projetos, esta nas técnicas implementadas. Isto é, na forma como é elaborado o planeamento e o controlo.

No Murallas Digital, o planeamento é realizado uma única vez e com um grande nível de detalhe no início do projeto. Já no CITT, o grau de detalhe no planeamento é menor, pois o objetivo principal são as entregas constantes ao cliente.

Por isso, à duração das atividades no Murallas Digital é um processo mais longo, com um planeamento mais detalhado, em que o tempo de desenvolvimento de cada release é de um mês. No CITT, o processo é mais curto como foco em entregas e resultados mais rápidos, em que a duração das *sprints* é de quinze dias.

A segunda diferença é na definição do âmbito do projeto, no Murallas Digital o âmbito do projeto tem como objetivo demonstrar a solução que se pretende chegar como: a definição de regras e orientando-se para as atividades de execução do projeto. No CITT, o objetivo não é mostrar o resultado final do projeto mas direcionar a equipa para um conjunto de soluções possíveis.

Outra diferença, entre estes projetos, esta na forma como as atividades são definidas. No projeto ágil é realizada uma lista de atividades, enquanto no tradicional as atividades são organizadas de forma hierárquica (WBS) e são sequenciadas como um todo.

Estas diferenças são as mais significativas entre estes projetos, contudo, existe um fator semelhante, que é ambos utilizarem o desenvolvimento distribuído de *software*.

Tendo em conta as caraterísticas do desenvolvimento de *software* definido neste trabalho, chegou-se à seguinte conclusão:

Para manter a equipa, é necessário fornecer aos seus elementos apoio e confiança necessária para a realização do trabalho. Em ambos, a comunicação entre a equipa serve de motivação e tem como objetivo fornecer um ambiente integrado

Por conseguinte, a comunicação é um dos maiores valores aplicados nestes projetos, pois é através desta característica que a informação circula dentro da equipa de forma mais eficiente.

Tendo em conta a literatura, a forma mais eficiente da informação circular dentro da equipa é através de uma conversa face-to-face. No entanto, ambos os projetos são desenvolvidos de forma distribuída o que torna esta característica uma desvantagem pois a ela estão associados

deslocações o que faz com que a organização tenha despesas elevadas e uma das soluções para a redução desses custos é a utilização de ferramentas de comunicação.

A utilização deste tipo de ferramentas, é uma forma de todos os elementos acompanharem as reuniões no seu local de trabalho, sem necessitarem de deslocarem-se de sítio.

O CITT promove reuniões diárias, designadas por *grooming*, o que promovem à equipa um desenvolvimento mais sustentável onde os impedimentos e os sucessos são compartilhados. Já o Muralhas promove reuniões quinzenais entre as equipas de desenvolvimento.

Tendo em conta, as outras características, que são comparadas neste capítulo, verifica-se que existe várias para a identificação da abordagem utilizada em cada projeto. A tabela 7, caracteriza resumidamente cada uma delas.

Tabela 7 - Características para a identificação da abordagem utilizada

Característica	Abordagem ágil	Abordagem tradicional
A forma de elaborar o plano de projetos	Há dois planos de projeto: a um plano geral que considera o tempo total de duração do projeto e um plano de curto prazo (iteração) que contém apenas as entregas e atividades referentes a uma fração de tempo do projeto. (Cohn, 2005)	Há um único plano de projeto, que abrange o tempo total do projeto (*Project Management Institute*, 2008)
Descrição do âmbito do projeto	Descrição do resultado final de forma mais abrangente, desafiadora, ambígua e metafórica. O objetivo não é mostrar o resultado final do projeto mas orientar a equipa para um conjunto possível de soluções. (Cohn, 2005)	Descrição exata do resultado final com a definição de regras e com orientações para a execução das atividades (*Project Management Institute*, 2008)
Estratégia utilizada para controlo do tempo de projeto	Implementação de dispositivos visuais As reuniões são curtas e frequentes. (Cohn, 2005)	Implementação de relatórios com indicadores de desempenho, documentos escritos, auditorias. As reuniões da equipa não são frequentes. (Project Management Institute,2008; Wysocki & McGary, 2007)
Definição das atividades	Realização de uma lista de atividades	As atividades são organizadas de forma hierárquica (WBS)

Relativamente as características do desenvolvimento distribuído, averiguou-se, através da tabela 8, que em nenhum projetos, o idioma é um problema, pois ambos utilizaram o português.

Tabela 8- Características do DDS no Murallas e CITT

Características do DDS	Murallas Digital		CITT
Agrupamento	EPMC: -Gestor de projeto - Analista - *Design* - Programador	UMC: - Gestor de projeto - *Design* - Programador	Equipa Scrum: - Scrum Master - Product Owner - Equipa de desenvolvimento: . Equipa do CCG: *Gestor de projetos *Analista *Gestor de testes *Testador . Programadores
Distancia física	Centralizado		Distribuído
Separação temporal	Inexistente		Inexistente
Cultura Regional	Sem diferenças		Sem diferenças
Idioma	Português		Português
Diferença dos locais	Praticamente nenhuma		Relativamente alguma
Cultura organizacional	Sem diferenças relevantes		Sem diferenças relevantes
Infra-estrutura	Ferramentas e técnicas pouco diferentes		Ferramentas e técnicas iguais
Relação de Negocio	Parceria entre organizações		Parceria entre organizações

Analogamente às diferenças físicas, apesar de em ambos os casos serem desenvolvidos no mesmo país, Portugal, o processo de distribuição é elaborado de forma diferente. No projeto Murallas a equipa de desenvolvimento era distribuída de forma centralizada, ou seja, a distribuição é feita no mesmo espaço físico, co-localizada. Ou seja, a comunicação neste tipo de desenvolvimento é mais facilitada, pois a comunicação na maioria das vezes é face-to-face com reuniões mensais.

No CITT a distribuição da equipa é feita de forma distribuída, porque, apesar de o produto ser desenvolvido no mesmo país, as equipas estão localizadas em cidades diferentes (Braga, Guimarães e Lisboa), o que origina a falta de contacto pessoal e a obrigatoriedade de haver um

relacionamento distribuído. Desta forma, as ferramentas de comunicação são utilizadas com uma maior frequência.

Comparativamente, ao fuso horário, ele é igual, uma vez que ambos os projetos são desenvolvidos no mesmo país. A diferença cultural, também não demonstrou qualquer existência relevante.

Na infra-estrutura, as ferramentas utilizadas no projeto Murallas divergem um pouco, uma vez que cada equipa usa o seu método de trabalho, enquanto, no CITT as ferramentas utilizadas são as mesmas para todos os elementos, pois trata-se de um projeto ágil, em que o *Scrum* é aplicado. As ferramentas de comunicação e gestão de desenvolvimento de um projeto ágil são definidas.

A característica agrupamento é parcialmente diferente, pelas seguintes razões: no Murallas Digital, os papéis dos elementos das equipas são tradicionais. A equipa EPMC é constituída por um analista, *design* e um programador, enquanto a equipa UMC é composta por um design e um programador.

O gestor de projeto, este é responsável pelas duas equipas, ou seja, é responsável pela gestão de toda a área de desenvolvimento de *software*.

O analista é responsável por todo o conjunto de requisitos (funcionais e não funcionais) de forma a produzir a documentação necessária. O *design* é responsável pela conceção de uma parte do sistema, dentro das limitações impostas pelos requisitos, arquitetura e processo de desenvolvimento de *software* e o programador neste projeto é responsável por observar as boas práticas em vigor na organização no que concerne ao desenvolvimento de *software,* entre as quais pode estar a seguinte: atualizar frequentemente no sistema de controlo de versões de código fonte dos componentes que se encontram envolvidos.

No CITT, a equipa é ágil, mais especificamente, é uma equipa *Scrum*. No *Scrum*, os papéis das equipas diferenciam-se das outras metodologias por seguir uma estrutura pouco hierarquizada.

A equipa neste projeto é constituída por: um scrum master que é representado por um elemento da empresa 1. Ele é responsável por desempenhar um papel de liderança, melhorar a vida e a produtividade da equipa de desenvolvimento e desta forma promover a criatividade e o conhecimento. O product owner que é representado por um elemento de cada organização. Eles são responsáveis por definir as funcionalidades e por aceitar ou rejeitar os resultados dos trabalhos.

Por fim, a equipa de desenvolvimento, neste projeto, é constituída pela equipa do CCG e por três programadores. Um dos programadores pertence á empresa 1 e os outros dois elementos foram subcontratados por esta, do CCG e da empresa 2.

A equipa do CCG tem como objetivo neste projeto o levantamento dos requisitos, na criação das user stories e pela elaboração dos testes.

Tendo em conta o que foi dito anteriormente, o CITT, relativamente à distribuição dos papéis é um pouco diferente. Pois, apesar de ser uma equipa Scrum, o processo utilizado internamente pelo CCG é o tradicional. Os papéis dos elementos da equipa do CCG são: analista, um gestor de projeto, um gestor de teste e um testador

É de salientar, que estes papéis tradicionais não mapeiam os papéis definidos pelo Scrum, por isso, chegou-se ao seguinte ao consenso: as equipas Scrum são compostas pelo scrum master, product owner e pela equipa de desenvolvimento., sendo esta uma equipa multifuncional, como uma divisão interna de tarefas. São compostas basicamente por *designers*, testadores e desenvolvedores e podem durante o desenvolvimento de software exercer atividades secundárias, relativamente a outros papéis.

A tabela 9 apresenta um conjunto de habilidades que caracterização a equipa de desenvolvimento *Scrum*, sendo uma delas a autogestão, característica forte e necessária aos elementos da equipa *Scrum*.

Tabela 9 – Caracterização dos papéis funcionais que compõem uma equipa de desenvolvimento *Scrum*

Papel Funcional	Responsabilidade	Habilidades necessárias
Analista	Identificar os requisitos do sistema	▪ Facilidade de expressão e comunicação; ▪ Facilidade de adaptação à mudança; ▪ Iniciativa na solução do problema; ▪ Tolerância a pressão; ▪ Organização; ▪ Trabalho em equipa; ▪ Autogestão; ▪ Objetividade.
Programador	Desenvolver as componentes	▪ Atenção a detalhes e boa memória; ▪ Capacidade de concentração; ▪ Paciência; ▪ Pro-atividade; ▪ Trabalho em equipa; ▪ Autogestão; ▪ Raciocínio logico desenvolvido; ▪ Disciplina.
Testador	Testar as componentes e o sistema	▪ Atenção a detalhes e boa memória; ▪ Capacidade de concentração; ▪ Disciplina; ▪ Paciência; ▪ Organização; ▪ Perseverança; ▪ Trabalho em equipa; ▪ Autogestão.
Design	Planear, conceder e desenhar o sistema	▪ Iniciativa na solução do problema; ▪ Facilidade de adaptação à mudança; ▪ Organização; ▪ Pro-atividade e objetividade; ▪ Capacidade de concentração; ▪ Atenção ao detalhe, ▪ Trabalho em equipa; ▪ Autogestão.

Todas estas características acima analisadas também apresentam alguns problemas. No Murallas Digital, no agrupamento um problema referenciado, foi a dificuldade que o gestor de projeto teve na monitorização da equipa, pois muitas vezes não sabia exatamente como estava o progresso de uma determinada tarefa.

Outro problema encontrado neste projeto, é referente à característica infra-estrutura em que os padrões de código e os métodos aplicados são diferentes. A inexistência de uma infra-estrutura de comunicação adequada impedia a participação virtual nas reuniões. Ou seja, quando um elemento da equipa faltava à reunião, a sua opinião era dada através de mensagens via email, ou esperar a próxima reunião para expor as ideias.

No projeto CITT, um dos problemas encontrados foi a falta de contacto pessoal apesar de existirem reuniões diárias, quinzenais e mensais.

Relativamente aos problemas referidos anteriormente no Murallas, eles não foram encontrados no CITT, uma vez que a utiliza uma abordagem ágil, o *Scrum* e nele são aplicadas as suas práticas e regras e fazem com que esses problemas da utilização do desenvolvimento distribuído de *software* sejam resolvidos.

É de salientar que o *Scrum* aplica nos seus projetos os três pilares. Ele aumenta a agilidade, a colaboração dentro da equipa e a transparência em relação ao que deve ser feito, por isso, um projeto desenvolvido com o *Scrum* para ambientes de *software* distribuído é elaborado de forma mais rápida e com sucesso.

4.5 Conclusão

Este capitulo teve como objetivo analisar, comparar e destacar as caraterísticas e os problemas encontrados e fazer o levantamento de eventuais soluções, de modo que elas possam ser aplicadas em equipas distribuídas.

A partir das modificações aplicadas ao *Scrum*, foi possível fazer um paralelo entre as suas práticas quando utilizadas por equipas no mesmo espaço físico e os problemas dessas práticas quando aplicadas de forma distribuída. Relativamente às dificuldades encontradas pode-se destacar a gestão da equipa e a comunicação entre os membros distribuídos, que podem ser ultrapassadas com as ferramentas e práticas do *Scrum*.

Os problemas e as soluções encontradas são apresentadas neste trabalho, através do estudo de casos em que ambos utilizam o desenvolvimento distribuído de *software* . Contudo só

um deles é apoiado pelo *Scrum*, ou seja, no projeto CITT é aplicado o *Scrum* num ambiente para *software* distribuído.

Através desta análise foi realizada uma simulação da aplicação do *Scrum* no projeto Murallas através da adaptação das práticas, papéis e recursos tecnológicos.

A adoção do *Scrum* num ambiente distribuído trouxe benefícios percetíveis para a elaboração do projeto como:

Maior comunicação e integração entre o cliente e a equipa de desenvolvimento no compartilhamento de processos, modelos, boas práticas e experiências.

Um aumento da produtividade da equipa devido à distribuição do trabalho e da diminuição do trabalho proporcionado pelas entregas contínuas e frequentes, permitem alinhar o entendimento dos requisitos, o refinamento da estratégia para os restantes requisitos, a obtenção de um feedback contínuo do cliente e o aumento da qualidade do produto que é construído através de pequenos incrementos que são testados pela equipa e avaliados pelo cliente.

O aumento da satisfação da equipa, devido à utilização de um processo mais flexível, é o facto de ser concedido uma maior autonomia aos integrantes do projeto, além de incentivar a colaboração entre eles. Também, fazem com que o cliente sinta uma maior satisfação devido a sua iteração mais frequente no projeto, assim, existe um maior feedback e entregas mais constantes.

Analogamente á análise comparativa dos projetos, verificou-se que o projeto CITT é mais complexo que o projeto Murallas, devido á falta de contacto pessoal e a obrigatoriedade de existir um relacionamento distribuído, aumentando assim a necessidade na utilização de ferramentas de comunicação.

Já, o Murallas Digital, por se tratar de um desenvolvimento centralizado, a comunicação é mais facilitada, porque na maior parte das vezes, o tipo de comunicação é face-to-face.

Relativamente à constituição da equipa verificou-se que no método *Scrum* a equipa é designada por equipa *Scrum* em que os papéis são: *Scrum Master*, *Product Owner* e a equipa de desenvolvimento sendo o ultimo composto por indivíduos com capacidades e características complementares de maneira a constituir uma equipa multidisciplinar capaz de atingir, de forma ágil, os objetivos do projeto.

Contudo, através da comparação destes caso e da análise destas características é possível identificar a eficácia e a eficiência da utilização do Scrum para um ambiente de *software* distribuído.

CAPITULO 5 - CONCLUSÃO

5.1 Síntese Final

Boa parte das metodologias ágeis surgiram da necessidade de fazer coisas distintas, para que os resultados diferentes fossem obtidos.

Com a globalização e o aumento das disputas entre as empresas de desenvolvimento de *software*, em que estas procuram sempre novas formas de aumentar a produtividade, reduzir o tempo de entrega e melhorar a qualidade do produto e reduzir os custos. O principal objetivo dessas mudanças é manter a competitividade e satisfazer o cliente.

Por isso, as empresas começaram a adotar novas formas de desenvolvimento, sendo elas ágeis, o *Scrum* , e o desenvolvimento distribuído de *software*.

O desafio futuro das metodologias ágeis é encontrar maneira de eliminar os seus pontos fracos sem torná-las em metodologias pesadas. Dai o surgimento da junção de dois métodos, o *Scrum* e o desenvolvimento distribuído de *software*.

As vantagens da utilização deste tipo de desenvolvimento são: a disponibilidade de mão-de-obra e custos mais baixos, uma maior acessibilidade e evolução dos recursos de telecomunicação.

Pode-se verificar que neste trabalho de dissertação os métodos ágeis e o desenvolvimento distribuído de *software* são duas realidades bem presentes no mundo altamente competitivo e globalizado atualmente demonstrado pelas empresas.

Relativamente aos objetivos deste trabalho, foi elaborado uma sinopse ao *Scrum* oferecendo uma panorâmica do funcionamento de um ciclo iterativo, explicando as responsabilidades de cada papel que pertence à equipa *Scrum*, detalhando as principais práticas que o mesmo pode utilizar para tornar a *sprint* mais produtiva e necessária para completar o ciclo.

Ao longo deste trabalho foi dada uma especial atenção à adaptação do *Scrum* em ambientes de *software* distribuído.

Por isso, também foi analisada de forma cuidada o desenvolvimento distribuído de *software* e as suas características. Através destas análises foram detetados problemas que foram colmatados a partir da junção destes dois métodos.

Nesta junção, as características do *Scrum* foram adaptadas ao desenvolvimento distribuído assim como os papéis de forma a solucionar alguns dos problemas encontrados em ambos os métodos de forma a melhorar o desenvolvimento e a equipa.

Paralelamente, foi dada uma especial atenção ao fatores de qualidade aplicados ao *Scrum*. Este fatores ao longo do trabalho foram adaptados ao desenvolvimento distribuído de *software*. A segurança, a integridade e a confiabilidade são uns desses fatores adaptados e que através da sua aplicabilidade resultados positivos são alcançados.

A segurança significa o fator de apoio ao uso do Scrum em ambientes distribuídos em possuir habilidades de evitar falhas que possam provocar alguns danos. Este fator é importante neste tipo de projeto, pois ao longo do seu desenvolvimento existe uma grande quantidade de troca de informações e muitas vezes de países diferentes.

Ela está relacionado com problemas de comunicação, fusos horários e horários de trabalho diferentes, diferenças culturais, diferenças linguísticas, ferramentas, administração da equipa, administração das reuniões à distância, gestão do processo *Scrum* e uma visão externa da organização à equipa. (Guerra et al., 2009; Albuquerque, 2001; Rocha, 1983; Carvalho, 1997; Grisolia, 1999; Valle et al., 1997; Belchior, 1997; Lima, 1999; Rocha et al., 1993; Fitzpatrick, 2007; Fitzpatrick, 2000).

No caso da integridade este fator apoia o uso do *Scrum* em ambientes distribuídos em preservar os seus dados e o processamento de situações anormais.

A confiabilidade apoia na manutenção do seu nível de desempenho ao longo do tempo. O resulto deste fator é não permitir grandes quantidades de falhas.

Por fim, foi comparado dois projetos do laboratório EPMQ, onde foi sistematizado as características de ambos.

Paralelamente, foi sugerida a adaptação do *Scrum* a um desses projetos, onde varias equipas trabalharam em simultâneo e onde a comunicação é fundamental.

A conclusão final a que se pode chegar com a adaptação do *Scrum* em ambientes de *software* distribuído é que, é importante perceber que há uma compensação a ser feita quando a equipa é distribuída. Uma equipa DDS que utiliza uma metodologia ágil, compensa as barreiras de comunicação acrescentado processos eficazes na troca de informações.

Qualidade e adaptação às mudanças são importantes diferenciais que podem ser atingidos utilizando-se estas metodologias pois elas procuram constantemente estabelecer paradigmas para o desenvolvimento de *software*.

Embora, este trabalho centre-se em equipas ágeis distribuídas, muitas das suas práticas aqui discutidas também são utilizadas com o sucesso em equipas co-localizados.

A utilização do *Scrum* em ambientes de *software* distribuído é certamente viável em projetos diversificados e podem trazer grandes benefícios em termos de qualidade do produto, produtividade, comunicação e satisfação da equipa e do cliente.

É importante frisar que o Scrum não promete resolver os problemas de uma organização. Pois ele apenas evidencia os problemas que sempre existiram e estabelece regras que incentivam a uma solução colaborativa dos mesmos.

Ele recomenda, que o utilizador deva ter um papel ativo no decorrer do projeto, permitindo assim, o cliente acompanhar o progresso do projeto e certificar-se que ele não fica abaixo das suas expetativas e assim aumenta a satisfação do cliente com o projeto.

Embora não seja a solução para todos os problemas, o scrum em ambientes distribuídos, mostra uma maneira de trabalhar bastante organizada e iterativa, podendo inclusive contribuir para um ambiente de trabalho mais amigável.

As entregas periódicas de pequenos incrementos, neste tipo de abordagem possibilitam um melhor controlo nos riscos do projeto através da mitigação periódica ao longo do projeto. Contudo, o constante envio do feedback pelo cliente faz do desenvolvimento algo bastante intenso para quem nele participa, podendo ser mentalmente cansativo e tornar difícil a equipa de desenvolvimento encontrar o equilíbrio.

Em suma, com base no que foi dito anteriormente, pode-se dizer que o enfase do *Scrum* em ambientes de *software* distribuído esta em criar o produto certo.

5.2 Trabalho Futuro

No que diz respeito ao trabalho futuro sugere-se:

Validar num projeto real: Para verificar qual o sucesso do *scrum* em ambientes de *software* distribuído. A avaliação ficaria mais completa se o projeto envolve-se equipas com fusos horários diferentes e elementos de outros países.

Implementar um ambiente virtual: Criação de ambientes que possam apoiar a implementação desta abordagem.

Adaptação de outras práticas ágeis: Neste trabalho ficou definido que se iria implementar o *Scrum* em ambientes distribuídos. Contudo, no futuro era interessante investigar outras práticas que possam ser adotadas e adaptadas num contexto distribuído

Criação de um método para classificar as características do desenvolvimento distribuído de *software* de forma a permitir uma classificação mais precisa do projeto.

REFERÊNCIAS BIBLIOGRÁFICAS

Agile Manifesto. (2004). Manifesto for Agile Software Development. Retrieved March 12, 2014, from http://agilemanifesto.org/

Agile Project Management Software, Agile Tools, Scrum Tools, Agile Software, Scrum Software | VersionOne. (n.d.). Retrieved March 12, 2014, from http://www.versionone.com/

Aken, J. E. van. (2004). Management Research Based on the Paradigm of the Design Sciences: The Quest for Field-Tested and Grounded Technological Rules. Journal of Management Studies, 41(2), 219–246. doi:10.1111/j.1467-6486.2004.00430.x

Akita, F. (2009). Voçe não entende nada de Scrum. Retrieved January 10, 2014, from http://www.akitaonrails.com/2009/12/10/off-topic-voce-nao-entende-nada-de-scrum#.UyBsdz9_s7E

Albuquerque, A. A.(2001). Qualidade de Websites de Comércio Eletrônico, Dissertação de Mestrado, UNIFOR, MIA, Fortaleza.

Alliance, A. (2004). Agile Alliance :: Home. Retrieved November 22, 2013, from http://www.agilealliance.org/

Ambler, S. (2005). The Agile Unified Process (AUP) Home Page. Retrieved March 12, 2014, from http://www.ambysoft.com/unifiedprocess/agileUP.html

Bayazit, N. (2006). Investigating Design: A Review of Forty Years of Design Research. Massachusetts Institute of Technology, 20, 16–29. Retrieved from http://www.mitpressjournals.org/doi/abs/10.1162/074793604772933739

Beck, K. (1999). Extreme Programming Explained: Embrace Change [Paperback] (p. 224). Addison-Wesley Professional; US ed edition. Retrieved from http://www.amazon.com/Extreme-Programming-Explained-Embrace-Change/dp/0201616416

Beck, K., & Andres, C. (2004). Extreme Programming Explained: Embrace Change, 2nd Edition (p. 224). Addison-Wesley; 2nd edition. Retrieved from http://www.amazon.com/Extreme-Programming-Explained-Embrace-Edition/dp/0321278658

BECK, K., & et al, M. for A. S. (2001). Agile Manifesto. Snowbird. Retrieved November 23, 2013, from http://agilemanifesto.org

Belchior, A. D. (1997). Controle da qualidade de sof-tware financeiro, Tese de Mestrado, UFRJ-COPPE, Rio de Janeiro.

Bento, A. (2012). Como fazer uma revisão da literatura: Considerações teóricas e práticas. Revista JA, 42–44.

Brod, C. (2008). Dados do “jeito Toyota.” Retrieved January 13, 2014, from http://www.brod.com.br/node/424

Brod Tecnologia | www.brod.com.br. (n.d.). Retrieved March 12, 2014, from http://www.brod.com.br/node/424

Carmel, E. (1999). Global Software Teams: Collaborating Across Borders and Time Zones. Prentice Hall.

Carvalho, I. C. L.(1997). Bibliotecas universitárias federais: o cenário da informatização, São Luís.

Cockburn, A. (2000). Selecting a project’s methodology. IEEE Software, 17(4), 64–71.

Cockburn, A. (2004). Crystal clear a human-powered methodology for small teams. Addison-Wesley Professional. Retrieved from http://dl.acm.org/citation.cfm?id=1406822

Cockburn, A. (2006). Agile Software Development: The Cooperative Game (p. 504). Addison-Wesley Professional; 2 edition. Retrieved from http://www.amazon.com/Agile-Software-Development-Cooperative-Edition/dp/0321482751

Cohn, M. (2005). Agile Estimating and Planning. Retrieved from http://www.google.pt/books?hl=pt-PT&lr=&id=BuFWHffRJssC&oi=fnd&pg=PT2&dq=COHN+M.+Agile+Estimating+and+Planning.&ots=WnhhwfV-Jk&sig=lqyoELWHToK2P2tS2Gp78p_BRzY&redir_esc=y#v=onepage&q=COHN M. Agile Estimating and Planning.&f=false

Cohn, Mike. (2009) Succeeding with Agile: Software Development Using Scrum. The Addison-Wesley Signature Series.

Conboy, K. (2009). Agility from first principles: reconstructing the concept of agility in information systems development. Information Systems Research, 20(3), 329–354. doi:10.1287/isre.1090.0236

DSDMC, (1997). DSDM Consortium. Delivering Aginle Business Solution on Time. Disponível em http://www.dsdm.org/. acesso em 14 de dezembro de 2013.

Fitzpatrick, R. (2000). Additional Quality Factors for the World Wide Web, Dublin, Ireland.

Fitzpatrick, R .(2007). A Theory and Practice of Website Engagibility, This Theses, Ph.D, DIT, Dublin,Ireland.

Grisolia, S. V. (1999). Uso de assistentes pessoais no prontuário médico eletrônico, Tese de Mestrado, COPPE/UFRJ, Rio de Janeiro.

Guerra, A. C.; Colombo, R. M. T. (2009). Tecnologia da Informação: Qualidade de Produto de Software, PBQP Software, Campinas.

Highsmith, J., & Cockburn, A. (2001). Agile software development: the business of innovation. Computer, 34(9), 120–127. doi:10.1109/2.947100

Haywood. (2000). M. Working in Virtual Teams: A Tale of Two Projects and Many Cities. IT Professional, v.2, n.2, p.58-60,. Disponível em: <http://www.computer.org/>. Acesso em: 17 de setembro. 2014.

Herbsleb, J.; Moitra. (2001). D. Global Software Development. IEEE Software, v.18, n.2, p.16-20.

Karolak,D.W.(1999). *Global Software Development: Managing Virtual Teams and Environments*. IEEE Computer Society Press, Los Alamitos, CA, USA.

Kniberg, Henrik. Scrum e XP direto das Trincheiras: Como fazemos Scrum. [s.L.]: C4media, 2007. Disponível em: <http://www.infoq.com/br/minibooks/scrum-xpfrom-the-trenches>. Acesso em: 14 maio 2014.

Kniberg, Henrik; SKARIN, Mattias. Kanban e Scrum: Obtendo o melhor de ambos.[s. L.]: C4media, 2009. Disponível em: <http://www.infoq.com/br/minibooks/kanbanscrum-minibook>. Acesso em: 1maio 2014.

Kobitzsch, W.; Rombach, D.; Feldmann, R. L.(2000). Outsourcing in India. IEEE Software, v.18, n.2, p.78-86.

Lacerda, D. P., Dresch, A., Proença, A., & Antunes Júnior, J. A. V. (2013). Design Science Research: método de pesquisa para a engenharia de produção. Gestão & Produção, 20(4), 741–761. doi:10.1590/S0104-530X2013005000014

Larman, C. (2004). Agile and Iterative Development: A Manager's Guide. Retrieved from http://books.google.pt/books/about/Agile_and_Iterative_Development.html?id=76rnV5Exs50C&pgis=1

Leffingwel, D., Smits, H., & Schwaber, K. (2013). Cartilha do CIO para Adoção do Método Scrum de Obtenção de Agilidade em Softwares. Rallly Software Development.

Luz,Marlon, G.D. and Teófilo,M. (2009). Challenges on adopting scrum for distributed teams in home office environments. *World Academy of science, Engineering, 2005. Canadian Conference on*, pages 920-926.

Manson, N. J. (2006). Is operations research really research? Orion, 22(2), 155–180.

Mnkandla, Ernest; Dwolatzky, Barry. (2006).Defining Agile Software Quality Assurance. Proceedings of the International Conference on Software Engineering Advances, pag. 36-36.

Nieveen, N. (1999). Prototyping to Reseach Product Quality. Design Aproaches and Tools in Education and Training, 125–135.

Olson, J. S.; Olson, G. M.(2003). Culture Surprises in Remote Software DevelopmentTeams. Queue Focus: Distributed Development, v.1, n.9, p.52-59, Disponível em: <http://www.acm.org>. Acesso em: 14 de setembro2014.

Paasivaara,M., Durasicwicz,S., and Lassenius,C. (2009). Using scrum in distributed agile development: A multiple case study. In I*CGSE´09: Proceedings of the 2009 fourth IEEE International Conference on Global Software Engineerings*, pages 195-204, washington, DC, USA. IEEE Computer Society

Peffers, K., & et al. (2008). A Design Science Research Methodology for Information Systems Research. Journal of Management Information Systems, 24(3), 45–47.

Peter, S. (n.d.). "Mandar adoptar Scrum é um padrão que falha." Retrieved December 11, 2013, from http://www.computerworld.com.pt/2013/11/29/mandar-adoptar-scrum-e-um-padrao-que-falha/

Phalnikar,R., Deshpande, V., and Joshi,S. (2009). Applying agile principles for distributed software development. *Advanced Computer Control, International Conference on*, 535-539.

PMI. (2005). Glossário Oficial do PMBOK. Retrieved from http://www.pmtech.com.br/downloads/GlossarioPMI.pdf

Prahalad, C. ., & Hamel, G. (1994). Strategy as a field of study: Why search for a new paradigm? Strategic Management Journal, 15, 5–16.

Prahalad, C. K., & Hamel, G. (2007). Strategy as a field of study: Why search for a new paradigm? Strategic Management Journal, 15(S2), 5–16. doi:10.1002/smj.4250151002

Pressman. (2011). Software Engineering: a Practitioner´s Approach. The McGraw-Hill Companies.

Pressman, R. S. (2010). Software engineering: a practitioner's approach. Retrieved from http://books.google.pt/books/about/Software_engineering.html?id=y4k_AQAAIAAJ&pgis=1

Project Management Institute - PMI. (2008). Guia PMBOK: um guia do conjunto de conhecimentos do gerenciamento de projetos (4. ed.). Pennsylvania: Project Management Institute.

Rocha, A. R.; Campos, G. H. B. (1993). Avaliação da Qualidade de Software Educacional, COPPE/UFRJ, Rio de Janeiro.

Roger, P. (2009). Software Engineering: A Practitioner's Approach [Hardcover] (p. 928). McGraw-Hill Science/Engineering/Math; 7 edition. Retrieved from http://www.amazon.com/Software-Engineering-A-Practitioners-Approach/dp/0073375977

Schubring, L. (n.d.). Play.Estimate.Plan. Retrieved December 10, 2013, from http://www.planningpoker.com/

Schwaber, K. (2004). Agile Project Management with Scrum (p. 188). Microsoft Press. Retrieved from http://www.amazon.com/Agile-Project-Management-Microsoft-Professional/dp/073561993X

Schwaber, K., & Sutherland, J. (2011). Guia do Scrum. Retrieved January 02, 2014, from https://www.scrum.org/Portals/0/Documents/Scrum Guides/Scrum Guide -

Siqueira, F. L.; Muniz Silva, P. S.(2004) As Características do Desenvolvimento Distribuído de Software. In: Simpósio Brasileiro de Sistemas de Informação, Porto Alegre, 2004. Anais. p171-178.

Simon, H. A. (1996). The Sciences of the Artificial. MIT Press. Retrieved from http://www.google.pt/books?hl=pt-PT&lr=&id=k5Sr0nFw7psC&oi=fnd&pg=PR9&dq=SIMON,+H.+A.+The+Sciences+of+the+Artificial.+3rd+ed.+Cambridge:+MIT+Press,+1996+.&ots=-u4JoDGNFv&sig=FGpqCggSt_vaUaQkEMyu2CVEEgk&redir_esc=y#v=onepage&q=SIMON%2C H. A. The Sciences of the Artificial. 3rd ed. Cambridge%3A MIT Press%2C 1996 .&f=false

Sommerville, I. (2007). Engenharia de Software. Pearson Addison-Wesley, 8 edition.

Sureshchandra,K. and Shrinivasavadhani,J. (2008). Adopting agile in distributed development. *In Global Software Engineering, 2008. ICGSE 2008*. IEEE International Conference on, pages 217-221.

Sutherland, J. (2007). The SCRUM papers: nuts, bolts and origins of an agile process. PatientKeeper, Inc. Retrieved from file:///C:/Users/crisl_000/Downloads/scrumpapers.pdf

Sutherland, J., Viktrov, A., Blount, J., & Puntikov, N. (2007). Distributed Scrum: Agile project management with outsourced development teams. IEEE Computer Society Professional Practices Committee, 274–284.

SWEBOK. (2004). Guide to the Software Engineering Body of Knowledge. IEEE Computer Society Professional Practices Committee. Retrieved from http://www.computer.org/portal/web/swebok/html/ch1

Takeda, H., & et al. (1990). Modeling Design Process. AI Magazine, 11(4), 37–48.

Ulman, D. (2002). The Mechanical Design Process [Hardcover] (p. 432). McGraw-Hill Science/Engineering/Math; 3 edition. Retrieved from http://www.amazon.com/Mechanical-Design-Process-Ullman-David/dp/0072373385

Ulrich, K. T., & Eppinger, S. D. (2000). Product design and development. McGraw-Hill. Retrieved from http://books.google.pt/books/about/Product_design_and_development.html?id=4QNPAAAAMAAJ&pgis=1

Vaishnavi, V., & Kuechler, W. (2009). Design Research inInformation Systems.

Vax,M. and Michaud,S. (2008). Distributed agile: Growing a practise together. In AGILE. pages 310-314.IEEE.

Versionne. (n.d.). 7th ANNUAL STATE of AGILE VERSIONONE® Agile Made Easier DEVELOPMENT SURVEY. Retrieved February 04, 2014, from http://www.versionone.com/

Woo, F., Mikusauskas, R., Bartlett, D., & Law, R. (2006). A Framework for the Effective Adoption of Software Development Methodologies.

Anexo A- Questionário de Análise do Projeto CITT

Com a criação deste questionário, definiu-se os seguintes objetivos:

1- Entender como o projeto é realizado
   a) O que o projeto pretendia
   b) Quais são as práticas do Scrum no projeto
   c) Quais são as principais dificuldades e benefícios do Scrum no projeto

2- Analisar as atividades de gestão do projeto
   a) Como é realizada a gestão de projetos
   b) Como o Scrum afetou a realização das atividades
   c) O que podia ter sido feito

Tendo em conta os objetivos, a seguir serão apresentadas as questões formuladas:

Visão geral

1- O projeto

a. O que esta a ser construído?
b. Qual a duração do projeto?
c. Qual é a experiencia das pessoas envolvidas com o desenvolvimento de software?
d. As pessoas têm conhecimento de algum problema?
e. As pessoas têm conhecimento do que é abordagem ágil?

2- Scrum

2.1. Como avalia o processo de implementação do Scrum no projeto?

2.2. O que motivou a adoção do Scrum?

2.3. Quais as práticas dos Scrum utilizadas?

2.4. Utiliza o Scrum com outra metodologia?

a. Qual?

b. Porquê?

2.5. Papéis da equipa

2.5.1. Quantos papéis da equipa existe?

a. Quais são? Qual é o número de elementos para cada um?

b. Que responsabilidades têm?

c. Como os cargos são atribuídos?

2.6. Relativamente à equipa, ela é auto-organizavel?

2.7. Como avalias a comunicação entre os membros das equipas após a implementação do Scrum na equipa?

2.8. Relativamente ao ciclo de vida do Scrum como ele esta a ser aplicado neste Projeto?

a. Qual é a duração de cada Sprint?

b. As reuniões de retrospetiva são elaboradas apos cada Sprint?

i. Qual é a sua opinião relativamente a esta atividade (Necessário ou não)? Porquê?

2.9. Ao utilizar o Scrum verifica alguma melhoria na qualidade do produto?

2.10. Quais as principais dificuldades encontradas na adoção do Scrum?

2.11. Quais os principais benefícios encontrados na adoção do Scrum?

3- Gestão de requisitos no Scrum

Responsável por extrair, organizar, documentar os requisitos do sistema e manter um acordo entre o cliente e a equipa do projeto.

3.1. São estabelecidas as necessidades para a realização da gestão de requisitos?

a. Quem estabelece?
b. Como é estabelecido (reunião, decisão unilateral, entre outros)?
c. Quais são as dificuldades para definir o que é preciso para o processo?

3.2. Como foram definidas as ferramentas e as tecnologias utilizadas?

3.3. São analisadas as possibilidades para a realização dessa atividade conforme definido

3.3.a. Como o Scrum influencia esta análise?

3.4. Como é feito o planeamento para a realização da extração, organização, documentação dos requisitos do sistema?

3.4.a. Como foi elaborado?

3.4.b. Como as pessoas participaram?

3.4.c. Quais os problemas encontrados?

3.4.d. Como devia ter sido feito?

3.5. Quais são as dificuldades encontradas para executar o plano?

3.6. Como são obtidos, definidos, documentados e divulgados os requisitos do sistema?

3.6.a. Que práticas do Scrum influencia?

3.7. Como são obtidos, definidos, documentados e divulgados os requisitos do software?

3.7.a. Que práticas do Scrum influencia?

3.8. Os requisitos são rastreados?

3.8.a. Quem faz isso?

3.8.b. Existe algum tipo de documentação?

3.8.c. Que problemas ocorrem?

3.9. O plano definido é monitorizado?

3.9.a. Quem fez? Como é feito?

3.9.b. Que problemas ocorrem?

3.10. Houveram problemas na extração, organização, documentação dos requisitos do sistema?

3.10.a. Que problemas ocorrem?

3.10.b. Como são solucionados?

3.11. Os resultados de extração, organização, documentação dos requisitos do sistema são avaliados?

3.11.a. Como é feita essa avaliação?

3.11.b. Como o Scrum dificultou a avaliação?

3.12. Na conclusão do projeto serão analisados os resultados e registos da atividade de extração, organização, documentação dos requisitos do sistema?

3.12.a. Verifica-se algum problema?

3.12.b. Como é gerido?

Engenharia de Processos

Responsável por moldar um processo que seja adequado as características do projeto

4.1. Como é decido quais as pessoas que devem trabalhar?

a. Como se chega a esse acordo?
b. Como o trabalho é dividido?

4.2. Como as pessoas se juntam no trabalho? Como é feita a integração do sistema?

4.3. Como é implementado o sistema?

4.4. Quais as ferramentas, softwares e métodos que usaram?

4.4.aComo se chega a esse acordo?

4.5. Como os problemas são geridos?

4.5.1. Existe alguma proposta de mudança?

4.5.1.a. Porque que é feita?

4.5.1.b. Como é feita?

4.5.2. Estas mudanças são documentadas?

4.6. É analisado se o processo é adequado?

4.6.a. Quem faz essa análise?

4.6.b. O que é dito pelos desenvolvedores?

4.6.c. O que pode ser feito?

Anexo B - Questionário de Análise do Projeto Murallhas Digital

Com a criação deste questionário, teve-se os seguintes objetivos:

1- Entender o projeto realizado
   a) O que o projeto pretendia
   b) Quais eram as características do DSD no projeto
   c) Quais foram os principais problemas encontrados causados pelo DDS

2- Analisar as atividades de gestão do projeto
   a) Como foi realizada a gestão de projetos
   b) Como o DDS afetou a realização das atividades
   c) O que deveria ter sido feito

Considerando os objetivos, a seguir serão apresentadas as questões formuladas:

Visão geral

1- O projeto
   a. O que foi construído?
   b. Qual a duração do projeto?
   c. Qual era a experiencia das pessoas envolvidas com o desenvolvimento de software?
   d. As pessoas tinham conhecimento de algum problema?

2- Características do DSD

2.1. Agrupamento

Num desenvolvimento de *software* distribuído as pessoas podem estar fisicamente separadas de diversas formas, formando grupos de pessoas que trabalham fisicamente próximos e outros grupos trabalham à distância.

   a) Qual é a quantidade de grupos?
   b) Qual é a quantidade de pessoas em cada grupo?
   c) Como os papéis estão distribuídos?

2.2. Distancia Física

Pessoas em grupos diferentes encontram dificuldade em encontrarem-se fisicamente.

   a. Qual é a distancia entre os grupos?
   b. Há recursos para viagens?
   c. Quantas viagens foram realizadas?

i. O grupo todo fisicamente chegou-se a reunir?
ii. As pessoas já se conheciam?

2.3. Separação Temporal

A existência de horários distintos entre os grupos

a. Havia um horário em comum de trabalho?
    i. Como é que as pessoas se comunicam para tirar dúvidas urgentes?
    ii. Houve algum tipo de reunião com os membros da equipa?

2.4. Culturas regionais

A diferença de valores e práticas existentes entre pessoas provenientes de diferentes regiões

a. Há alguma diferença de cultura regional entre as pessoas?
    i. Ela influenciou de alguma forma o desenvolvimento de software?

2.5. Idioma

A dificuldade de expressão e entendimento na língua padrão adotada para o projeto o que envolve a diferença da língua mãe entre as pessoas envolvidas no projeto.

a. Foi definida uma língua padrão para o projeto?
b. Houve problemas devido as diferenças de entendimento e expressão na língua padrão do projeto?

2.6. Diferença dos locais

Dificuldades ocorridas pela localização dos grupos

a. Pelo fato de um determinado grupo estar em determinado local trouxe problemas para o desenvolvimento?
    i. Qual foi o problema?
    ii. Como foi solucionado?

2.7. Culturas organizacionais

As diferenças de estratégias, objetivos, filosofias, crenças, pensamentos e sentimento que são criadas nas organizações que as pessoas trabalham

a. As pessoas já tinham trabalhado juntas?
b. Houve conflitos por objetivos, formas de trabalho e ideias diferentes?

2.8. Infra-estrutura

A diferença de hardware, software, ferramentas básicas entre outras

a. Há muita diferença relativamente ao hardware, software, ferramentas e técnicas usadas?
    i. Como foram solucionadas essas diferenças?
    ii. Foi feito algum investimento?
b. O que usaram para a comunicação à distância entre os grupos?

2.9. Relação de negócio

A relação existente entre as organizações e as pessoas envolvidas no projeto

a. Como as pessoas estão ligadas ao projeto?

3- Problemas

3.1. Quais foram os principais problemas ocorridos?

a. Gestão
b. Técnico
c. Grupo
d. Cultural
e. Comunicação
f. Legislação
g. Organizacional
h. Tecnológicos
i. Infra-estrutura
j. Logístico
k. Gestão de conhecimento
l. Mais algum?

4- Gestão de requisitos

Responsável por extrair, organizar, documentar os requisitos do sistema e manter um acordo entre o cliente e a equipa do projeto.

4.1. Foram estabelecidas as necessidades para a realização da gestão de requisitos?
a. Quem estabeleceu?
b. Como foi estabelecido (reunião, decisão unilateral, entre outros)?
c. Quais foram as dificuldades para definir o que é preciso para o processo?

4.2. Como foram definidas as ferramentas e as tecnologias utilizadas?
4.3. Foram analisadas as possibilidades para a realização dessa atividade conforme definido
    a. Como o DSD influenciou nessa análise?

4.4. Como foi feito o planeamento para a realização da extração, organização, documentação dos requisitos do sistema?

a. Como foi elaborado?
b. Como as pessoas participaram?
c. Quais os problemas encontrados?
d. Como devia ter sido feito?

4.5. Quais foram as dificuldades encontradas para executar o plano?
4.6. Como foram obtidos, definidos, documentados e divulgados os requisitos do sistema?
a. Que características do DSS influenciaram?
4.7. Como foram obtidos, definidos, documentados e divulgados os requisitos do software?
a. Que características do DSD influenciaram?

4.8. Os requisitos são rastreados?
a. Quem fez isso?
b. Existia algum tipo de documentação?
c. Que problemas ocorreram?

4.9. O plano definido foi monitorizado?
a. Quem fez? Como foi feito?
b. Que problemas ocorreram?

4.10. Houveram problemas na extração, organização, documentação dos requisitos do sistema?
a. Que problemas ocorreram?
b. Como são solucionados?

4.11. Os resultados de extração, organização, documentação dos requisitos do sistema são avaliados?
a. Como é feita essa avaliação?
b. Como o DSD dificultou a avaliação?

4.12. Na conclusão do projeto serão analisados os resultados e registros da atividade de extração, organização, documentação dos requisitos do sistema?
a. Verifica-se algum problema?
b. Como é gerido?

5. Engenharia de Processos

Responsável por moldar um processo que seja adequado as características do projeto

5.1. Como é decido quais as pessoas que devem trabalhar?
a. Como se chega a esse acordo?
b. Como o trabalho é dividido?
5.2. Como as pessoas se juntam no trabalho? Como é feita a integração do sistema?
5.3. Como é implementado o sistema?
5.4. Quais as ferramentas, softwares e métodos que usaram?

a. Como se chegou a esse acordo?

5.5. Como os problemas são geridos?

a. Existiu alguma proposta de mudança?

5.5.a.1. Porque que foi feita?

5.5.a.2. Como foi feita?

b. Estas mudanças são documentadas?

5.6. É analisado se o processo é adequado?

a. Quem faz essa análise?

b. O que é dito pelos desenvolvedores?

c. O que poderia ter sido feito?