Luiz Cezer Marrone Filho

Scrum Distribuído - Práticas Ágeis para equipes distribuídas

Monografia apresentada no curso de Pós-Graduação do Centro Universitário Católica de Santa Catarina como requisito parcial para obtenção do certificado do curso.

Joinville
2015

Luiz Cezer Marrone Filho

Scrum Distribuído - Práticas Ágeis para equipes distribuídas

Monografia apresentada no curso de Pós-Graduação do Centro Universitário Católica de Santa Catarina como requisito parcial para obtenção do certificado do curso.

Área de Concentração: Pós Graduação em Engenharia de Software

Orientador: Maurício Henning

Joinville
2015

Filho, Luiz Cezer Marrone
Scrum Distribuído - Práticas Ágeis para equipes distribuídas. Joinville, 2015.

Monografia - Centro Universitário Católica de Santa Catarina.

1. Desenvolvimento de Software 2. Equipes distribuídas 3. Scrum I.Centro Universitário Católica de Santa Catarina. Curso de Bacharaledo em Sistemas de Informação. Curso de Pós-Graduação em Engenharia de Software.

Sumário

Lista de Figuras

Resumo

O tema deste trabalho é expor a forma como equipes de desenvolvimento de software geograficamente distribuídas podem usufruir de métodos ágeis, como por exemplo o *Scrum*, abordando quais seus pontos fortes e desafios a serem superados.

Palavras-chave: Desenvolvimento de Software, Equipes distribuídas, Métodos Ágeis, Scrum

Abstract

The purpose of this coursework is to expose the way that remote development teams can use agile methods, like *Scrum*, with focus in advantages, challenges to overcome.

Keywords: Software Development, Distributed teams, Agile, Scrum

Capítulo 1

Introdução

Nos dias atuais, empresas de desenvolvimento de software precisam de cada vez mais agilidade na hora de desenvolver e entregar seus projetos. Além de agilidade é preciso garantir consistência e um ritmo sustentável de trabalho, sem deixar de lado o cliente, garantindo a ele entregas periódicas do projeto, tudo isso sem esquecer da garantia de qualidade do que foi e está sendo entregue e que os requisitos podem mudar a qualquer momento. Para isso, empresas fazem o uso de métodos ágeis de gestão, afim de entregar software de forma incremental, mais rapidamente e com uma acertividade maior (ÁGIL, 2001).

Um exemplo desses métodos ágeis de gestão é o *Scrum*. O *Scrum* é um *framework* de gerenciamento de projetos, que possibilita um desenvolvimento iterativo e incremental do projeto, visando uma maior comunicação entre todos os envolvidos no projeto (ÁGIL, 2013b).

Para que empresas possam evoluir e responder as demandas do mercado, precisam de cada vez mais profissionais melhores e mais capacitados, porém nem sempre é possível obter esses recursos humanos no local onde a empresa se encontra fisicamente, além disso o custo de manutenção de escritório, deslocamento dos funcionários e outros fatores podem atrapalhar na decisão de expandir a equipe com novos funcionários.

Na tentativa de reduzir esses impactos e ainda sim conseguir expadir a equipe, muitas empresas adotam a contratação de funcionários para trabalho remoto, o que faz com que a equipe se torne distribuida em partes ou em sua totalidade (ECôNOMIA, 2015) (AUDY; PRIKLADNICKI, 2007).

Porém mesmo que uma equipe seja parcialmente ou totalmente remota, ela ainda precisa ser organizada, comunicativa e focada na entrega, e o *Scrum* é uma das metodologias

que podem ajudar equipes que trabalham nesse modelo.

Este trabalho tem como objetivo apresentar o *Scrum* voltado a equipes remotas, descrevendo suas práticas, diferenças encontradas entre equipes presenciais e remotas, quais ferramentas e práticas ser adotas para minimizar esse problemas.

1.1 Objetivo Geral

Realizar um estudo sobre as principais práticas, desafios, problemas e ferramentas utilizadas por equipes remotas que utilizam a metodologia *Scrum*.

1.2 Objetivos Específicos

1. contextualizar por meio de referencial teórico sobre métodos ágeis e *Scrum*;
2. identificar quais melhores práticas se aplicam em equipes remotas;
3. identificar quais as dificuldades equipes remotas enfrentam com mais frequência;
4. propor melhores alternativas para facilitar o trabalho da equipe;
5. propor ferramentas de apoio;
6. coletar de dados com profissionais da área sobre práticas de equipes remotas.

1.3 Justificativa do trabalho

Com muitas empresas T.I. hoje aderindo a forma de contração de funcionários remotos, é preciso garantir que a equipe, seja ela em sua totalidade ou parcialidade remota, consiga trabalhar de modo ágil, sustentável, sempre focado na entrega e ainda sim mantendo a sua organização e colaboração ativa. Para isso as empresas podem adotar *Scrum* como metologia ágil afim de organizar melhor a equipe e focar nos seus resultados.

Capítulo 2

Referencial Teórico

Neste capítulo serão apresentados os principais conceitos que serão abordados durante o trabalho.

2.1 Metodologias Ágeis

Visando minimizar problemas do desenvolvimento de *software* sem algum planejamento é preciso adotar uma metodologia que ajude no processo de planejamento e na solução a ser desenvolvida.

Segundo (FOWLER, 2003) citado por (PASSUELO, 2005) "Metologias impõem um processo disciplinado no desenvolvimento de *software*, com objetivo de torná-lo mais previsível e mais eficiente".

Porém o problema de muitas metologias é a burocrácia. Esas metodologias são focadas em documentação e gerenciamento rígido do projeto, muitas vezes acabam sendo vistas como um grande atraso para o projeto, já que a grande maioria do tempo do projeto será gasto levantado requisitos e documentando todas as ações do sistema, antes mesmo de ele ser codificado. (FOWLER, 2003)

Então nos anos noventa, surge uma reação a essas metodologias mais rígidas, são as chamadas metodologias ágeis no processo de desenvolvimento de *software*.

Criado por Kent Beck, com uma equipe de desenvolvedores experientes, o Manifesto Ágil é o documento que reune todas as práticas e princípios dos métodos ágeis dentro do processo de desenvolvimento (BECK, 2001). O manifesto Ágil possui valores, que são eles:

- **Indivíduos e interações** mais que processos e ferramentas;
- ***Software* em funcionamento** mais que documentação detalhada;
- **Colaboração com cliente** mais que negócios e contratos;
- **Responder a mudança** mais que seguir um plano;

  Ou seja, mesmo havendo valor nos itens a direita, valorizamos mais os itens a esquerda. (ÁGIL, 2001)

Além dos valores, o Manifesto Ágil se baseia em alguns princípios:

- Satisfazer o cliente por meio de entregas adiantadas e contínuas do *software*;
- Aceitar as mudanças de requisito a qualquer momento dentro do projeto;
- Entrega de *software* em funcionamento com frequência semanal ou mensal;
- Pessoas que entendem do negócio e desenvolvedores devem estar sempre trabalhando juntas durante o processo de desenvolvimento;
- Trabalhar com pessoas motivadas no projeto, dando a eles um bom ambiente de trabalho e confiar que farão aquele que foi designado a eles;
- A melhor maneira de se transmitir informação é por meio de uma conversa pessoal;
- *Software* funcional é a medida primária de progresso;
- Processos ágeis promovem um ambiente sustentável;
- Contínua atenção a excelência técnica e bom design;
- Simplicidade;
- As melhores arquiteturas, requisitos e designs emergem de times auto-organizáveis;
- Em intervalos regulares, o time reflete em como ficar mais efetivo, então, se ajustam e aperfeiçoam seu comportamento de acordo.

Por meio desses princípios e valores pode-se notar que as metodologias ágeis deixam de lado a documentação pesada, processos de extração e elaboração de requisitos e passam a valorizar mais o *software* funcionando, liberando pequenas versões à medida que as mesmas ficam prontas, trazem o cliente para dentro do processo de desenvolvimento

fazendo o mesmo participar e acompanhar o crescimento do projeto, enfatiza interação entre pessoas sejam elas do time de desenvolvimento ou pessoas que entendem do negócio e abraçam mudanças no *software* a qualquer momento (FOWLER, 2003) (BECK, 2001).

2.2 Extreme Programming (XP)

A *Extreme Programming* mais conhecida com XP é uma metodologia de desenvolvimento que tem como finalidade entregar *software* de qualidade e com as necessidades que o cliente realmente precisa.

É uma metodologia voltada para pequenos e médios times de desenvolvimento que trabalham em projetos orientados a objetos e que seus requisitos podem sofrer constantes mudanças. (KUHN, 2008)

A satisfação do cliente é o que faz a metodologia ser bem sucedida. O XP busca o máximo desempenho da equipe mantendo um ritmo saudável de trabalho onde horas extra só são realizadas se forem agregar algo de valor ao produto final.

O cliente obterá seu produto em pequenos lançamentos, podendo utilizá-lo, aprender com o mesmo e avaliar se o que foi desenvolvido era o desejado. (ÁGIL, 2013a)

(ÁGIL, 2013a) menciona os valores seguidos pelo XP:

- ***Feedback*:** Quanto mais cedo um problema for descoberto, menos prejuízo e atraso ao *software* ele dará e maiores são chances de ele ser resolvido rapidamente;
- **Comunicação:** Da várias formas de se comunicar existentes, times que utilizam XP prezam por uma conversa presencial para melhorar a compreensão e entendimento entre os membros;
- **Simplicidade:** Embora muitas vezes no desenvolvimento é um hábito desenvolver pensando em futuras implementações, no XP o time é focado em desenvolver o que é necessário para agora, entregando o que realmente o cliente necessita para o momento, deixando o que vier depois, para depois;
- **Coragem:** Equipes de desenvolvimento precisam encarar mudanças constantes nos projetos e para isso é necessário coragem e confiança em suas práticas;
- **Respeito** É o valor mais básico e serve de base a todos, pois dará confiança e respeito a toda equipe durante o processo.

Aliado aos valores e princípios a metologia XP também possui uma série de práticas para descrever quais atitudes a equipe deve tomar no ambiente de trabalho e como se deve ocorrer o processo de desenvolvimento pela equipe (ÁGIL, 2013a).

- ***Design* simples e incremental:** O desenvolvimento da aplicação será de maneira iterativa e incremental, ou seja, a equipe se focará em desenvolver os *software* do modo mais simples possível, dando valor e codificando o que é realmente necessário para o cliente e permitindo o *software* crescer aos poucos;
- **Programação em pares:** É uma prática que visa nivelar o trabalho da equipe, sugerindo que dois programadores trabalhem juntos de forma colaborativa na mesma tarefa, enquanto um foca sua atenção na criação do código o outro tera o trabalho de revisar e manter o primeiro focado na solução mais simples;
- **Código de posse coletiva:** Em um projeto todos tem responsabilidade e liberdade de alterar todo e qualque código encontrado, o código não pertence a ninguém e sim a todo mundo. Isso ocorre quando equipes trocam de membro de forma constante e abordando o código dessa maneira é mais fácil transmitir o conhecimento do projeto, sem atrelar determinados códigos ou funcionalidades a determinados desenvolvedores;
- **Desenvolvimento guiado por testes:** É a prática que propoem a crição dos testes antes mesmo do código de produção ser implementado, tem como objetivo maior detectar e solucionar todas as possíveis falhas do desenvolvimento por meio dos testes. Dessa forma, quando um código é terminado não é preciso voltar até ele para testá-lo, pois ele já estará testado e pronto.

2.3 Scrum

Após a criação do Manifesto Ágil, surgiram várias metodologias que visavam tirar proveito dos valores e princípios, em 1990 surge o *Scrum* que segue a seguinte definição:

> "*Scrum* é um *framework* Ágil, simples e leve, utilizado para a gestão do desenvolvimento de produtos complexos imersos em ambientes comple- xos. *Scrum* é embasado no empirismo e utiliza uma abordagem iterativa e incremental para entregar valor com frequência e, assim, reduzir os riscos do projeto". (SABBAGH, 2013)

Embora o *Scrum* vise ser menos burocrático que outras metologias, ele trás consigo uma série de definições e papéis para melhor organizar sua equipe e seus processos. Porém pelo fato de ser definido como um *framework*, não há um processo rígido fixo e sim um processo que pode ser adaptado para cada projeto e cada equipe.

Em sua forma mais básica o *Scrum* é composto por um time, seus artefatos e seus eventos. O processo de desenvolvimento é baseado em *Sprints*, onde o time irá interagir e trabalhar no projeto. (ÁGIL, 2013b) (SABBAGH, 2013)

A visão geral do processo de *Sprint* pode ser vista na Figura 2.1 abaixo:

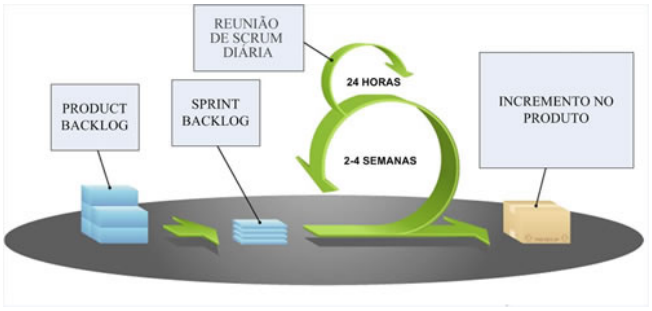

Figura 2.1: Visão geral de uma *Sprint* dentro do *Scrum*
(JAMIL, 2013)

2.3.1 O time do Scrum

Dentro do *Scrum* existem três papéis definidos: o Time de desenvolvimento, *Scrum Master* e o *Product Owner*. Embora existam essas três separações, toda equipe é igualmente responsável e responsabilizada pelo resultados obtidos, isso faz com que todos se comprometam com o projeto inteiramente.

Todos os membros juntos, formam o chamado Time do *Scrum*, todos visam trabalhar juntos de forma colaborativa, para que como equipe todos consigam alcançar um único objetivo coletivo.

2.3.1.1 Time de desenvolvimento

São os responsáveis pela entrega do produto, geralmente um time é composto de três a dez pessoas. Formam um time auto-organizado, multidisciplinar, motivado e com foco em qualidade. Dentro do time não há divisões entre programadores, *designers*, engenheiros ou analistas. Todos trabalham juntos de forma igualmente responsável, com objetivo de

entregar o que foi prometido para a *Sprint*. (SABBAGH, 2013) (ÁGIL, 2013b)

O time possui várias atividades e tarefas dentro da *Sprint*:

- Planejar seu trabalho junto ao *Product Owner* e posteriomente dividir as tarefas entre a equipe;
- Realizar as tarefas para que o objetivo da *Sprint* seja atingido;
- Interagir com *Product Owner* sempre que for necessário esclarecimento de alguma regra de negócio;
- Identificar e informar ao *Scrum Master* todo e qualquer impedimento que possa atrapalhar o trabalho da equipe;
- Entregar valor para o cliente de forma frequente, organizada e com qualidade.

2.3.1.2 Product Owner

É o responsável pela visão do produto, por tomar as decisões de negócio, definir e priorizar as atividades do *Product Backlog*, ele representa a necessidade do cliente em relação ao produto que está sendo desenvolvido.

Também colabora com o time de desenvolvimento a longo da *Sprint* no esclarecimento de dúvidas sobre as regras de negócio e fica responsável por aceitar ou não o trabalho entregue pela equipe no decorrer da *Sprint*, validando se o que foi entregue é realmente o que foi combinado e irá satisfazer a necessidade do cliente. (SABBAGH, 2013) (ÁGIL, 2013b)

2.3.1.3 Scrum Master

O *Scrum Master* é o responsável por garantir o funcionamento e execução do *Scrum* no decorrer de uma *Sprint*. Possui além de conhecimentos técnicos, habilidade de gerenciar pessoas, técnicas para facilitar o trabalho e está sempre tentando remover qualquer bloqueio que possa impedir o time de desenvolvimento. (SABBAGH, 2013) (ÁGIL, 2013b)

2.3.2 Artefatos do Scrum

Os artefatos do *Scrum* servem como ferramentas de opoio a todo o time, afim de que todos possam ter um acompanhamento geral dos trabalho durante a *Sprint* e uma visão geral do andamento do produto que está sendo feito.

2.3.2.1 Product Backlog

É lista de funcionalidades desejadas para o produto que está sendo desenvolvido. É responsabilidade do *Produto Owner* adicionar, remover, priorizar todas as tarefas dentro do *Product Backlog*. Necessariamente não contêm somente novas funcionalidades, podem também conter melhorias, correções de falhas, questões técnicas, ou seja, tudo que for necessário e possa vir a ser desenvolvido para entregar o produto.

Incialmente não precisa conter todas as tarefas, pode conter somente aquelas necessárias para iniciar o desenvolvimento do produto e a medida que o time e o cliente aprendem mais sobre o produto, novas tarefas serem adicionadas. (SABBAGH, 2013) (ÁGIL, 2013b)

2.3.2.2 Sprint Backlog

É lista de funcionalidades extraídas de dentro do *Product Backlog* que serão desenvolvidas durante uma *Sprint* pelo time de desenvolvimento. Os itens são extraídos de acordo com a priorização do *Product Owner* e a equipe tem a liberadade de escolher os itens que são inclusos na *Sprint* de acordo com a sua percepção do tempo que cada item levará para ficar pronto.

O desempenho do time pode ser medido, baseando-se na quantidade de itens que foram entregues entre um dia outro da *Sprint*. (SABBAGH, 2013) (ÁGIL, 2013b)

2.3.2.3 Quadro de Tarefas

Para tornar mais fácil e visual o acompanhamento das tarefas dentro da *Sprint*, é um comum equipes usarem um quadro de tarefas, que geralmente é quebrado de acordo com a situação de cada tarefa dentro da *Sprint*. (ÁGIL, 2013b)

Geralmente esse quadro possui as seguintes situações: Pendente, Andamento, Finalizado. Um exemplo desse quadro pode ser visto na Figura 2.2 abaixo:

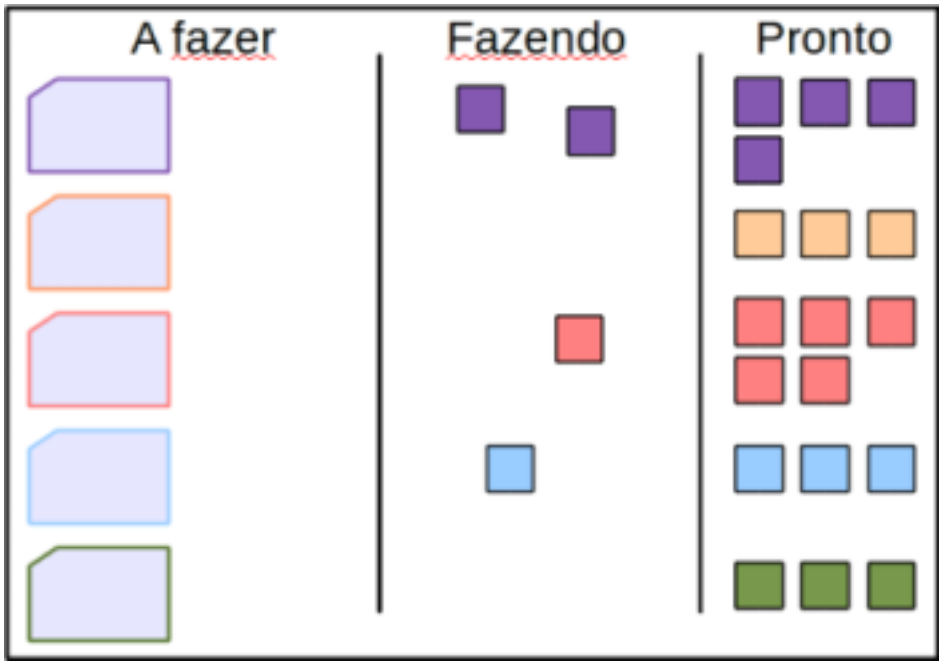

Figura 2.2: Quadro de tarefas de uma equipe ágil
(FERNANDES, 2011)

2.3.2.4 Gráfico de Burn Down

Aliado ao quadro de tarefas, é comum equipes ágeis utilizarem gráficos para melhorar o acompanhamento de tudo aquilo que foi feito e o que ainda está pendente dentro da *Sprint*. Com esse gráfico é possível medir o ritmo da equipe e detectar possíveis impedimentos. (ÁGIL, 2013b)

Um exemplo de gráfico, pode ser visto na Figura 2.3 abaixo:

2.3.3 Eventos do Scrum

Eventos do *Scrum*, são na verdade o próprio ciclo de desenvolvimento, denominado *Sprint*. Durante o período da *Sprint* a equipe faz um série de cerimônias com objetivo de atualizar todos os membros sobre o andamento do trabalho e também ajuda a detectar algum possível impedimento para o escopo da *Sprint*.

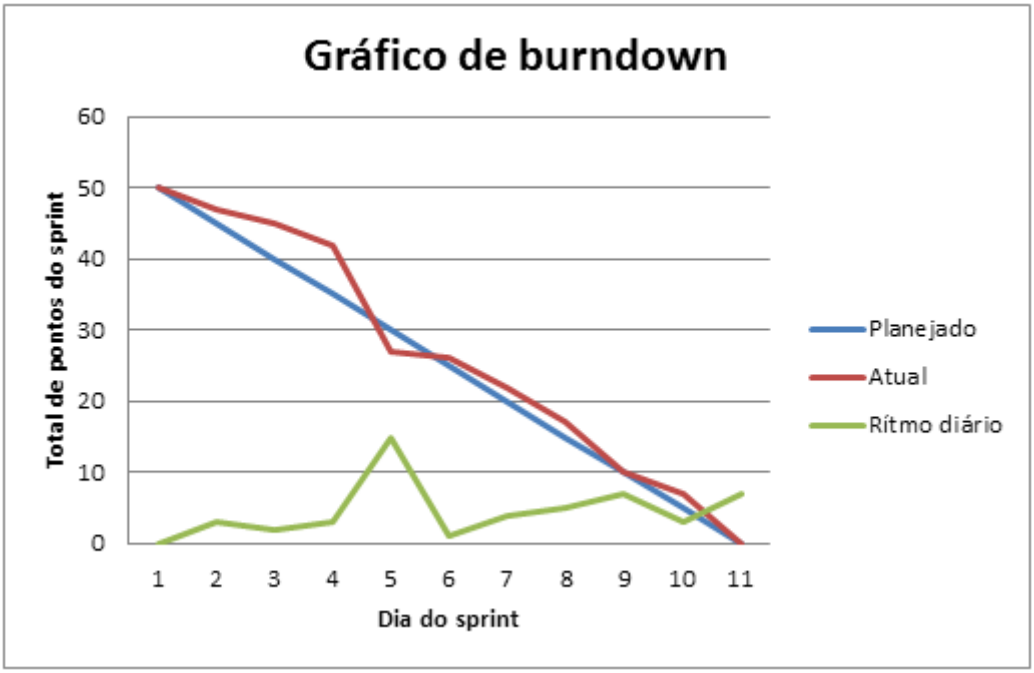

Figura 2.3: Gráfico de *Burn Down* do *Scrum*
(CUNHA, 2012)

2.3.3.1 Sprint

A *Sprint* é um mini-projeto, que tem duração entre uma e quatro semanas e é durante esse intervalo de tempo que o Time de desenvolvimento faz o incremento do produto. O objetivo maior da *Sprint* é que ao seu final uma nova parte utilizável do software seja entregue.

As *Sprints* sempre se iniciam com uma *Sprint Planning* que é uma reunião, com participação de todos os envolvidos no projeto e nessa reunião são definidas quais tarefas serão desenvolvidas pelo time de desenvolvimento para adicionar um novo incremento ao produto. Ocorrem em sequência, ou seja, não há intervalos entre as *Sprints*, cada vez que uma termina, uma nova *Sprint* se inicia. (SCHWABER, 2013) (SABBAGH, 2013)

Existem algumas regras que devem ser seguidas dentro de uma *Sprint*:

- Não são feitas mudanças que podem por em risco o objetivo da *Sprint*;
- As metas de qualidade nunca diminuem;
- O escopo pode ser negociado com o *Product Owner*;
- Somente o *Product Owner* tem autoridade para cancelar uma *Sprint*.

2.3.3.2 Sprint Planning

É a primeira cerimônia de uma nova *Sprint*. É uma reunião com participação de todos os envolvidos onde o *Product Owner* mostra quais as atividides de maior prioridade naquele momento e a equipe irá selecionar quais dessas atividades entram na *Sprint* para gerar um novo incremento no *software*.

Nessa reunião o time de desenvolvimento tem a liberdade de questionar o *Product Owner* sobre as atividades e também obter esclarecimento sobre dúvidas. Essas tarefas selecionadas darão origem ao *Sprint Backlog*. (SCHWABER, 2013) (ÁGIL, 2013b)

2.3.3.3 Dayli Scrum

É uma reunião curta e realizada todos os dias pela equipe de desenvolvimento, geralmente ocorre sempre no mesmo horário e tem uma duração de aproximadamente quinze minutos. Essa reunião visa melhorar a organização da equipe e tem como foco a transparência para que todos saibam o que todos estão fazendo e se alguém está passando por alguma dificuldade.

Durante os *Dayli Scrums* todos os membros da equipe se focam em responder três perguntas:

- O que eu fiz ontem ?;
- O que eu farei hoje ?;
- Algum impedimento para o que eu fiz ontem ou terei que fazer hoje ?

Se algum impedimento é encontrado, o *Scrum Master* irá agir com a finalidade de remover esse bloqueio para que o fluxo de trabalho da equipe continue. (ÁGIL, 2013b)

2.3.3.4 Sprint Review

A *Sprint Review* é uma reunião feita com toda a equipe ao final de uma *Sprint*. Tem como objetivo avaliar tudo que foi feito ao longo da *Sprint*, se o que foi feito está de acordo com o que foi combinado na reunião de *Sprint Planning* e se irá entrega um novo incremento ao *software*. Também serve para detectar se houveram problemas críticos que

pudessem atrapalhar o andamento do trabalho e como eles foram solucionados. (ÁGIL, 2013b)

2.3.3.5 Sprint Retrospective

É uma análise feita entre o a *Sprint Review* e a próxima *Sprint Planning*, serve para avaliar o que foi feito na última *Sprint*, quais pontos do processo foram falhos, quais ajudaram a equipe a ter uma melhor performance e quais pontos podem ser melhorados para otimização da equipe. (ÁGIL, 2013b)

2.4 Qualidade de Software

Segundo (PRESSMAN, 2011)

> Conformidade a requisitos funcionais e de desempenho explicitamente declarados, a padrões de desenvolvimento claramente documentados e a características implícitas que são esperadas de todo *software* profissionalmente desenvolvido.

A qualidade de *software* não pode estar ligada somente ao *software* fazer algo certo, como foi pedido pelo cliente e também não se restringe ao *software* final criado. Qualidade de *software* está ligada também a qualidade do código, reusabilidade, manuteabilidade, aos processos adotados durante o desenvolvimento como análise de requisitos e documentação.

Existem fatores para determinar a qualidade de *software* (PRESSMAN, 2011):

- **Corretitude:** O *software* satifaz as especificações determinadas e cumpre os objetivos desejados pelo cliente;
- **Confiabilidade:** O *software* executar determinada funcionalidade com a precisão esperada;
- **Eficiência:** Quantidade de recursos computacionais e de código exigida pelo *software* para executar uma funcionalidade;
- **Integridade:** Controlar acesso ao *software* ou a dados a pessoas não autorizadas;

- **Usabilidade:** Esforço para aprender, operar e preparar a entrada e interpretar uma saída do *software*;
- **Manuteabilidade:** Esforço para localizar e reparar erros no *software*;
- **Flexibilidade:** Esforço para modificar um programa operacional;
- **Testabilidade:** Esforço exigido para testar um *software* a fim de garantir que ele execute uma funcionalidade pretendida;
- **Portabilidade:** Esforço exigido para transferir o programa de um ambiente de sistema de *hardware*/*software* para outro;
- **Reusabilidade:** À medida que um *software* cresce possibilita algumas de suas partes serem reutilizadas em outros ou até no mesmo *software*;
- **Interoperabilidade:** Esforço exigido para se acoplar um sistema a outro.

Essas são os fatores propostos por McCall, Richards e Walters (1977) e citados por (PRESSMAN, 2011). Esses fatores podem ser agrupados e representados conforme a Figura 2.4.

Figura 2.4: A Pirâmide de Qualidade de Software
(CAMPOS, 2012)

2.5 Desenvolvimento Distribuído de Software

Hoje em dia a área de desenvolvimento de *Software* tem evoluído de forma cada vez mais rápida, não só a área de desenvolvimento, mas também as empresas precisam desenvolver *software* cada vez mais rápido e de maior qualidade. (AUDY; PRIKLADNICKI, 2007)

Paralelo a isso é possível notar um grande avanço em direção a globalização dos negócios e o *software* tem se tornado cada vez mais importante como componente de negócio. Com esse avanço, mercados que antes eram nacionais facilmente podem tornar-se multinacionais e para empresas que queiram acompanhar esse ritmo de crescimento fica cada vez mais trabalhoso, custoso e complexo desenvolver *software* no mesmo local físico ou empresa. Aliado ao avanço dos meios de comunicação e aumento dos custos para manter funcionários, empresas tem sido inventivadas a investir no Desenvolvimento Distribuído de *Software* formando equipe parcialmente um completamente distribuídas. (AUDY; PRIKLADNICKI, 2007) (ECôNOMIA, 2015)

Segundo (AUDY; PRIKLADNICKI, 2007), a distribuição do processo de desenvolvimento de *software* faz ampliar os problemas relacionados ao desenvolvimento tradicional e gera novos desafios como a distância, falta ou problemas de comunicação entre a equipe e diferenças culturais.

Segundo (ROCHA; MORAES; MEIRA, 2009) entre os problemas é possível citar:

- Distância socio-cultural;
- Motivação;
- Distância geográfica e temporal;
- Complexidade e tamanho do projeto;
- Comunicação;
- Ambientes e Ferramentas de Desenvolvimento;
- Metodologia;

Porém a abordagem de DDS também poderá trazer vantagens, que segundo (PRIKLADNICKI, 2003) são:

- Possibilidade de desenvolvimento contínuo, permitindo um aumento de produtividade e a redução dos prazos de entrega;
- Disponibilidade de recursos globais com baixos custos e a qualquer hora;
- Disponibilidade de recursos qualificados em áreas especializadas;
- Oportunidade de realizar o desenvolvimento do produto próximo ao cliente;
- Possibilidade de formar equipes virtuais.

Referências Bibliográficas

AUDY, J. L. N.; PRIKLADNICKI, R. *Desenvolvimento Distribuído de Software*. [S.l.]: Elsevier, 2007.

BECK, K. *Manifesto para Desenvolvimento Ágil de Software*. 2001. Disponível em: <http://www.manifestoagil.com.br/>. Acesso em: 17 jun. 2015.

CAMPOS, F. M. *Qualidade de Software e Garantia de Qualidade de Software são as mesmas coisas ?* 2012. Disponível em: <http://www.linhadecodigo.com.br/artigo/1712/qualidade-qualidade-de-software-e-garantia-da-qualidade-de-software-sao-as-mesmas-coisas.aspx>. Acesso em: 19 jun. 2015.

CUNHA, D. *Metodologia de Gerenciamento baseado em Scrum*. 2012. Disponível em: <http://www.olharcritico.com/engenhariadesoftware/2012/01/metodologia-de-gerenciamento-baseada-em-scrumparte-8/>. Acesso em: 18 jun. 2015.

ECôNOMIA, B. *Trabalho remoto ganha mais espaço nas empresas*. 2015. Disponível em: <http://brasileconomico.ig.com.br/vida-e-estilo/2015-01-30/trabalho-remoto-ganha-mais-espaco-nas-empresas.html>. Acesso em: 17 jun. 2015.

FERNANDES, C. *Pensando em métricas para times ágeis*. 2011. Disponível em: <http://blog.caelum.com.br/pensando-em-metricas-para-times-ageis/>. Acesso em: 18 jun. 2015.

FOWLER, M. *The new Methodology*. 2003. Disponível em: <http://martinfowler.com/articles/newMethodology.html>. Acesso em: 17 jun. 2015.

JAMIL, V. *O que é a metodologia Scrum em projetos?* 2013. Disponível em: <http://www.scriptcase.com.br/blog/metodologia-scrum-projetos/>. Acesso em: 18 jun. 2015.

KUHN, G. R. *Apresentando XP: Encante seus clientes com Extreme Programming*. 2008. Disponível em: <http://javafree.uol.com.br/artigo/871447/Apresentando-XP-Encante-seus-clientes-com-Extreme-Programming.html>. Acesso em: 25 jun. 2015.

PASSUELO, L. *A Nova Metodologia*. 2005. Disponível em: <http://www.p3software.com.br/home/artigos/6-a-nova-metodologia>. Acesso em: 17 jun. 2015.

PRESSMAN, R. S. *Engenharia de Software*. 7. ed. [S.l.]: MCGRAW HILL - ARTMED, 2011.

PRIKLADNICKI, R. *MuNDDoS: Um Modelo de Referência para Desenvolvimento Distribuído de Software*. Pontifícia Universidade Católica do Rio Grande do Sul, Brasil, 2003.

ROCHA, R. G. C.; MORAES, A. K. de O.; MEIRA, S. L. *Fatores que Afetam o Desenvolvimento Distribuído de Software*. Centro de Informatica Universidade Federal de Pernambuco (UFPE), Brasil, 2009.

SABBAGH, R. *Scrum - Gestão Agil para projetos de sucesso*. [S.l.]: Casa do Código, 2013.

SCHWABER, J. S. K. *Guia do Scrum*. [S.l.]: Scrum.org, 2013.

ÁGIL, D. *O que é Extreme Programming ?* 2013. Disponível em: <http://www.desenvolvimentoagil.com.br/xp/>. Acesso em: 25 jun. 2015.

ÁGIL, D. *O que é Scrum ?* 2013. Disponível em: <http://www.desenvolvimentoagil.com.br/scrum/>. Acesso em: 15 jun. 2015.

ÁGIL, M. *Manifesto Ágil*. 2001. Disponível em: <http://www.agilemanifesto.org/iso/ptbr/>. Acesso em: 15 jun. 2015.