Pontifícia Universidade Católica
do Rio de Janeiro

Evandro Oliveira das Flores

Uma Análise de Práticas na Aplicação de SCRUM em Projetos de Grande Porte

Dissertação de Mestrado

Dissertação apresentada como requisito parcial para a obtenção do grau de Mestre pelo Programa de Pós-graduação em Informática do Departamento de Informática do Centro Técnico Científico da PUC-Rio.

Orientador: Prof. Arndt von Staa

Rio de Janeiro
Março de 2012

Pontifícia Universidade Católica
do Rio de Janeiro

Evandro Oliveira das Flores

Uma Análise de Práticas na Aplicação de SCRUM em Projetos de Grande Porte

Dissertação apresentada como requisito parcial para a obtenção do grau de Mestre pelo Programa de Pós-graduação em Informática do Departamento de Informática do Centro Técnico Científico da PUC-Rio. Aprovada pela Comissão Examinadora abaixo assinada.

Prof. Arndt von Staa
Orientador e Presidente
Departamento de informática - PUC-Rio

Prof.ª Simone Diniz Junqueira Barbosa
Departamento de informática - PUC-Rio

Prof. Gustavo Robichez de Carvalho
Departamento de informática - PUC-Rio

Prof. José Eugenio Leal
Coordenador Setorial do Centro
Técnico Científico – PUC-Rio

Rio de Janeiro, 5 de Março de 2012

Evandro Oliveira das Flores

Graduou-se em Tecnologia em Processamento de Dados pela Fundação Educacional Serra dos Órgãos (FESO) em Maio de 2001. Tem experiência na área de Ciência da Computação, com ênfase em Desenvolvimento de Software. Tem trabalhado em análise de sistemas desde 1997.

Ficha Catalográfica

Flores, Evandro Oliveira das

Uma análise de práticas na aplicação de SCRUM em projetos de grande porte / Evandro Oliveira das Flores; orientador: Arndt von Staa. – 2012.

79 f. : (il. color.) ; 30 cm

Dissertação (mestrado)–Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática, 2012.

Inclui bibliografia

1. Informática – Teses. 2. Gestão de projetos de desenvolvimento de software. 3. Processos ágeis. 4. Scrum. 5. Kanban. I. Staa, von Arndt. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. III. Título.

CDD 004

A minha esposa pelo apoio incondicional durante todas as fases do mestrado.

Agradecimentos

Agradeço primeiramente à Deus, que me conduziu durante essa caminhada, colocando as pessoas certas no meu caminho.

À minha família, pela ajuda e apoio ao longo deste trabalho e de toda minha vida.

À minha esposa Vanesa por acreditar nos meus ideais e por me apoiar durante todo o mestrado.

À Globo.com, por ter financiado o mestrado e por dispor do tempo para que pudesse ser concluído.

Ao meus colegas da Globo.com, pelas experiências que me motivaram a escrever essa dissertação e pelos estudos paralelos a ela.

Ao meu orientador Arndt von Staa, pela orientação nas pesquisas, comentários construtivos e pelo apoio ao longo de meu trabalho.

Resumo

Flores, Evandro Oliveira das; Staa, Arndt Von (Orientador). **Uma Análise de Práticas na Aplicação de SCRUM em Projetos de Grande Porte.** Rio de Janeiro, 2012. 79p. Dissertação de Mestrado - Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Na literatura disponível hoje, encontram-se exemplos da utilização do Scrum em projetos e times pequenos, deixando um questionamento sobre a possibilidade de utilização desta metodologia em projetos e/ou times grandes. Esta dissertação tem por objetivo examinar casos práticos de empresas conhecidas onde foi utilizado Scrum em projetos grandes, enfatizando as dificuldades encontradas ao longo de todo processo e as soluções adotadas, destacando as práticas que levaram os projetos a obter sucesso.

Palavras-chave

Gestão de Projetos de Desenvolvimento de Software; Processos Ágeis; Scrum; Kanban.

Abstract

Flores, Evandro Oliveira das; Staa, Arndt von (Advisor). **An Analysis of Practices in Applying SCRUM on Large Projects.** Rio de Janeiro, 2012. 79p. MSc Dissertation - Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

In today's writings, there are many examples of using Scrum in small teams and projects, leaving a knowledge gap about the possibility of applying Scrum in big teams and large projects. This work aims at presenting case studies in known companies where Scrum has been applied in large projects, emphasizing the roadblocks found throughout the process, and the solutions adopted, highlighting the practices that lead the projects to success.

Keywords

Project Management; Agile Processes; Scrum; Kanban.

Sumário

Lista de imagens

Lista de tabelas

1 Introdução

Os processos tradicionais de desenvolvimento de software são baseados em paradigmas e no modelo mental herdados da revolução industrial. A forma analítica de encarar os problemas, dividindo-os em partes menores e depois atacando cada uma delas como problemas isolados, a forma de encarar o trabalhador como um recurso, substituível, todo esse modelo mental foi formado a partir do trabalho do Taylor [Taylor, 1911].

Em grandes projetos, um dos maiores desafios é manter-se no curso do planejamento, em termos de escopo, prazo e qualidade. De acordo com o modelo mental Taylorista, a forma mais simples de controlar atrasos ou acelerar o desenvolvimento de um projeto é adicionar mais recursos ao mesmo. O PMBOK reflete isso muito bem ao recomendar a técnica de *crashing*, ou seja, adicionar recursos ao projeto (humanos ou materiais) com o objetivo de acelerar seu progresso, mesmo que isso signifique a perda de eficiência. E como diz Brooks no livro *The Mythical Man-Month* isso é uma solução ineficaz, piorando a situação do projeto, *"adding manpower to a late software project makes it later"* [Brooks, 1975].

Considerando a forte associação do modelo mental Taylorista à revolução industrial, podemos observar que os reflexos nos processos, técnicas e métodos utilizados para gerenciar projetos hoje têm a sua base nas premissas de meados do século XX. Esses métodos funcionaram para produção de bens materiais, construções, indústrias, e foram adaptados para projetos de software. Entretanto, projetos de software, diferente dos modelos citados anteriormente, são mais dependentes de conhecimento empírico e comunicação, consequentemente, adicionar mais pessoas a esse meio só faz dificultar a eficácia da comunicação, por adicionar mais canais.

Em um processo ágil, utilizando XP ou Scrum, um time é formado seguindo uma sugestão de quantidade pequena de pessoas (7 ± 2). Mas, o que acontece quando existe a necessidade de aumentar a velocidade das entregas? O que acontece caso se tenha um projeto grande com um prazo de entrega curto? É possível utilizar Scrum em projetos de grande porte?

Analisando as possibilidades de generalizar as soluções em futuras aplicações, diante destes questionamentos, esta dissertação se propõe a examinar casos práticos em que foi utilizado Scrum em grandes projetos, enfatizando as dificuldades encontradas ao longo de todo o processo e as soluções adotadas, destacando as observações realizadas no ajuste e acompanhamento de projetos desenvolvidos em uma grande empresa sediada no Rio.

Uma das características do Scrum [Leffingwell, 2007], é o trabalho em pequenos times: *"Small, cross-functional teams work closely together in an open environment to produce incremental releases of a product in 30-day increments, or sprints"*.

Se Scrum prega a aplicação de equipes reduzidas, o que acontece com um grande projeto? Uma forma possível de "escalar" usando Scrum é criar múltiplos times. *"Many projects require more effort than a single Scrum Team can provide. In these circumstances, multiple Teams can be employed. Working in parallel..."* [Schwaber, K. 2004].

"Scrum is the first process with well-documented linear scalability. When you double team size in a well-implemented Scrum, you can double software output, even when the teams are distributed and outsourced". [Vaihansky, P. / Sutherland, J. / Victorov, A. 2006]. Essa afirmativa conflita com o livro *The Mythical Man-Month "adding manpower to a late software project makes it later"* [Brooks, 1975] que diz que adicionar recursos a um projeto em andamento fora do seu cronograma, fará com que esse projeto se atrase ainda mais, por questões de perdas devidas à comunicação entre os participantes.

Considerando um grupo de pessoas precisa manter uma comunicação diária entre si, ao aumentar o número de pessoas aumentam as combinações dessas

comunicações, fazendo com que, em situações patológicas, o esforço gasto em comunicação possa tornar-se maior do que o esforço gasto em produção. Uma maneira de diminuir o ruído nessas comunicações é o uso de algumas práticas no ambiente ágil, que não só facilitam a comunicação como também a detecção de problemas de forma mais rápida, que em consequência podem ser atacados mais rapidamente, seja na sua solução ou no replanejamento do plano original do projeto. Algumas dessas práticas que serão abordadas nessa dissertação são: integração contínua, entregas pequenas e frequentes, definição prévia do ciclo de desenvolvimento de componentes [Leffingwell, 2007].

1.1. Organização da dissertação

Essa dissertação está organizada da seguinte forma:

No capítulo 2 são apresentados alguns métodos ágeis, mais detalhadamente o Scrum.

Algumas boas práticas utilizadas em um ambiente ágil no que diz respeito à utilização dessas metodologias em um projeto grande são exemplificadas no capitulo 3.

O capitulo 4 irá abordar duas formas possíveis de organizar pessoas em um projeto grande, em um único time grande ou compondo diversos times pequenos. Outro ponto abordado são os casos de empresas que aplicaram Scrum em projetos de médio e grande porte e as lições que aprendidas a partir de suas experiências.

Por fim, no capítulo 5 são apresentadas as conclusões e propostas de trabalhos futuros.

2 Métodos Ágeis

2.1. Manifesto Ágil

Em fevereiro de 2001, dezessete representantes de diversas práticas e metodologias de desenvolvimento se reuniram em uma estação de esqui, em Utah nos EUA para discutir métodos mais leves de desenvolvimento do que o tradicional desenvolvimento orientado a documentos.

Autodenominados de "*The Agile Alliance*" criaram o *Manifesto for Agile Software Development* ou simplesmente Manifesto Ágil para definir a abordagem hoje conhecida como desenvolvimento ágil.

"Estamos descobrindo maneiras melhores de desenvolver software fazendo-o nós mesmos e ajudando outros a fazê-lo. Através desse trabalho, passamos a valorizar:

Indivíduos e interação entre eles mais que processos e ferramentas;
Software em funcionamento mais que documentação abrangente;
Colaboração com o cliente mais que negociação de contratos;
Responder às mudanças mais que seguir um plano.

Ou seja, mesmo havendo valor nos itens à direita, valorizamos mais os itens à esquerda." [Highsmith, 2001]

2.2. Princípios

Além do manifesto ágil, foram documentados doze princípios com a intenção de dar suporte a esse manifesto.

- Nossa maior prioridade é satisfazer o cliente através da entrega contínua e adiantada de software com valor agregado.

- Mudanças nos requisitos são bem-vindas, mesmo tardiamente no desenvolvimento. Processos ágeis tiram vantagem das mudanças visando vantagem competitiva para o cliente.

- Entregar frequentemente software funcionando, em poucas semanas a poucos meses com preferência à menor escala de tempo.

- Pessoas de negócio e desenvolvedores devem trabalhar diariamente em conjunto por todo o projeto.

- Construa projetos em torno de indivíduos motivados. Dê a eles o ambiente e o suporte necessário e confie neles para fazer o trabalho.

- O método mais eficiente e eficaz de transmitir informações para e entre uma equipe de desenvolvimento é através de conversa face a face.

- Software funcionando é a medida primária de progresso.

- Os processos ágeis promovem desenvolvimento sustentável. Os patrocinadores, desenvolvedores e usuários devem ser capazes de manter um ritmo constante indefinidamente.

- Contínua atenção à excelência técnica e bom design aumenta a agilidade.

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

- As melhores arquiteturas, requisitos e designs emergem de equipes auto-organizáveis.

- Em intervalos regulares, a equipe reflete sobre como se tornar mais eficaz e então refina e ajusta seu comportamento de acordo.

[Highsmith. 2001]

2.3. Processos definidos versus processos empíricos

"É típico adotar a abordagem de modelagem definida (teórica) quando os mecanismos subjacentes pelos quais um processo opera são razoavelmente bem entendidos. Quando o processo é muito complexo para ser definido, a abordagem empírica é a escolha apropriada." [Ogunnaike / Ray, 1992]

O modelo de controle de processos definidos requer que cada parte do trabalho seja completamente entendida. Dado um conjunto de inputs bem definidos, os mesmos outputs são gerados todas às vezes. Um processo definido pode ser iniciado e executado até sua conclusão, com os mesmos resultados a cada execução. As principais características de processos definidos que podemos destacar são:

- Processo definido com mecanismos subjacentes claramente entendidos
- Sucessão de atividades claramente definidas e lineares
- Capacidade de estimar tempos de execução de cada atividade

O modelo de controle de processos empíricos provê e exercita o controle através de inspeção frequente, com visibilidade, e adaptação para processos que não são perfeitamente definidos e geram saídas imprevisíveis e não-repetíveis. Por muitos anos, metodologias de desenvolvimento de software foram baseadas no modelo de controle de processo definido. Mas desenvolvimento de software não é um processo que gera a mesma saída dada uma certa entrada. O método ágil de desenvolvimento de software Scrum é baseado no modelo de controle de processos empírico [Schwaber / Beedle, 2001].

2.4. Scrum

2.4.1. Histórico

O Scrum foi criado oficialmente na década de 90, por Ken Schwaber e Jeff Shuterland, com a ajuda de Mike Beedle, John Scumniotales e Jeff McKenna. A sua criação foi na realidade feita de maneira completamente independente, em duas frentes: de um lado, Ken Schwaber, e do outro Jeff Shuterland. Em 1995-1996, Ken e Jeff se reuniram e documentaram o processo do Scrum como se conhece hoje em dia [Sutherland, 2004]. Em 2001, Ken Schwaber e Jeff Shuterland fizeram parte do grupo de profissionais que assinou o *Agile Manifesto*.

Desde então, Scrum vêm sendo usado nos mais diferentes projetos, pelas mais diferentes organizações. Grandes empresas multinacionais, pequenas empresas startup's, desenvolvedores para agências do governo, e até a indústria de games e animação vêm se beneficiando do Scrum [Sutherland, 2004].

2.4.2. O que é?

Scrum é um processo iterativo incremental para desenvolver qualquer produto ou gerenciar qualquer trabalho. No fim de cada *Sprint* (iteração), é produzido um incremento de funcionalidades potencialmente entregáveis. As principais características do Scrum são [Schwaber, 2004]:

- É um processo ágil (*agile*) para gerenciar e controlar trabalho.
- É um "embrulho" para as práticas existentes de engenharia.
- É uma aproximação coletiva (equipes) para desenvolver produtos e sistemas iterativamente e incrementalmente, onde requisitos mudam rapidamente.
- É um processo que controla o caos de interesses e necessidades conflitantes.
- É uma maneira de melhorar a comunicação e maximizar cooperação.
- É uma forma de detectar e remover barreiras que entrem no meio do desenvolvimento e entregas de produtos.
- É uma forma de maximizar produtividade.

• Scrum é escalável de um único projeto a toda uma organização, com vários projetos inter-relacionados.

• Scrum é uma maneira de fazer com que todos se sintam bem em relação ao seu trabalho, suas contribuições, e que eles fizeram o melhor possível, o melhor que eles poderiam fazer.

• Scrum requer trabalho duro.

• Scrum requer comprometimento

2.4.3. Fluxo

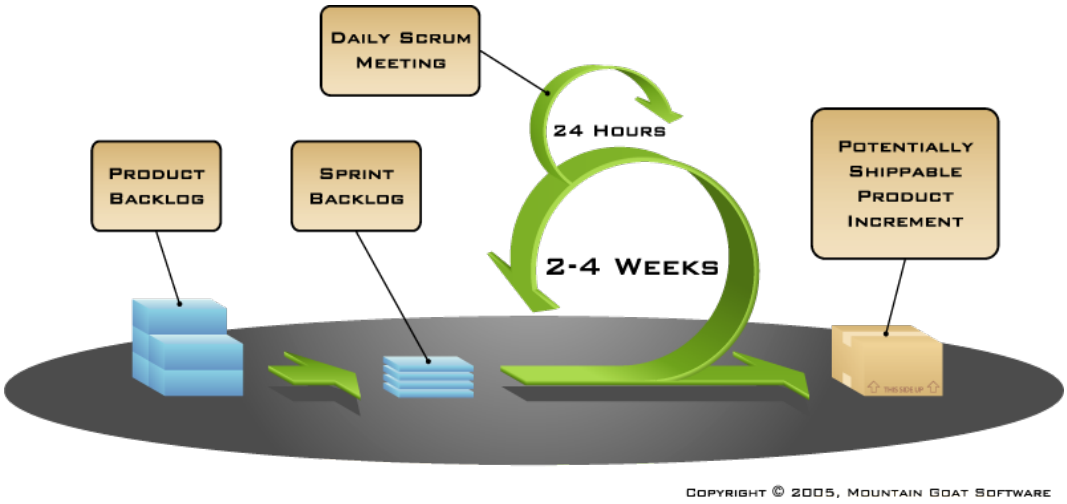

Figura 1 – Fluxo do Scrum

[Mountain Goat, 2005]

O Scrum define papéis, artefatos e cerimônias, que provêm o *Scrum Framework.* O Scrum trabalha em um processo iterativo e incremental, onde as iterações são chamadas de *Sprint*. Os *Sprints* têm duração pré-definida e fixa. Normalmente um *Sprint* dura entre 2 a 4 semanas. Durante o *Sprint*, o produto é planejado, codificado e testado. No final de cada *Sprint* há uma entrega.

2.4.4. Duração dos Sprints

É importante que todos os *Sprints* possuam a mesma duração, pois a utilização de um período constante leva a um melhor ritmo da equipe. Procure

manter a duração o mais constante possível, qualquer que seja o tamanho do *Sprint* que você escolher. Essa constância dá estabilidade ao time.

Ao planejar a duração do *Sprint* do seu projeto, procure responder à seguinte pergunta:

Por quanto tempo é possível manter uma mudança "fora" do *Sprint*?

Dependendo da estabilidade dos seus requisitos e características do seu negócio, você encontrará a duração ideal. Manter a estabilidade dentro de um *Sprint* é a regra.

2.4.5. Papéis e responsabilidades

2.4.5.1. Product Owner

O *Product Owner*, ou simplesmente PO, representa o cliente, patrocinadores e/ou financiadores do projeto. Muitas vezes, pode ser um representante do cliente dentro da empresa, ou um analista de negócio. É quem vai definir, gerenciar e priorizar os requisitos, gerenciar o ROI (*Return over investment*). O *Product Owner* trabalha como parte do time que realiza a entrega.

É responsabilidade do PO:

- Definir e manter uma visão compartilhada do projeto
- Gerenciar o ROI
- Apresentar requisitos iniciais e incrementais ao time
- Priorizar cada requisito em relação ao valor de negócio (*Business Value*)
- Gerenciar e priorizar o *Product Backlog*
- Gerenciar novos requisitos e sua priorização
- Fazer o planejamento de entregas (*Release Planning*)
- Agir como mediador quando houver mais de um cliente
- Garantir que especialistas no domínio do negócio estarão disponíveis para o time quando necessário
- Aceitar ou rejeitar o resultado dos trabalhos

Uma das responsabilidades do *Product Owner*, "Definir e manter uma visão compartilhada do projeto", merece ser mais bem explicada. A visão do produto é,

uma visão geral, ou "*big picture*", do produto, que todos têm conhecimento e entendimento desta visão. Esta visão vai guiar as atividades de desenvolvimento de produto, e provê direção ao time em relação ao que estamos tentando alcançar. Esta visão permite nos abrir para o fato que podemos atingir o possível.

Exemplo:

"Implementar um repositório eletrônico central de documentos para solicitações de hipoteca, provendo ao banco (cliente) a habilidade para lidar com as quantidades crescentes de solicitações de hipoteca sem contratar novos funcionários, através de ganho de produtividade. A solução vai nos permitir reduzir o tempo de espera dos clientes por respostas, mantendo nossa vantagem competitiva como um fornecedor/provedor orientado à clientes de serviços financeiros, enquanto ao mesmo tempo reduziremos atritos entre os funcionários, provendo economia significante em treinamento e reposição de funcionários existentes."

2.4.5.2. Scrum Master

O *Scrum Master*, ou simplesmente SM, representa a gestão do projeto, mas não no sentido de gestão tradicional. Na realidade, o SM funciona mais como um mediador e líder *coach* (em sentindo de *coaching*, como um técnico de time de futebol). O *Scrum Master* trabalha com e para o time.

É responsabilidade do SM:

- Permitir que o time se auto-organize para realizar o trabalho
- Garantir que as vias de comunicação estejam livres e acessíveis
- Garantir e ajudar que o time siga o processo do Scrum
- Cuidar e proteger a equipe de interferências externas, de modo a garantir que a produtividade do time não seja a afetada
- Remover impedimentos (obstáculos) que o time encontrar
- Agir como facilitador nas cerimônias como o *Daily Meeting*
- Garantir a colaboração entre os papéis

2.4.5.3. Equipe

A equipe é composta por indivíduos que estão comprometidos em realizar o trabalho proposto. Os times são *cross-functional*. Mas o que isso quer dizer? Nos métodos tradicionais, nós temos papéis distintos, por exemplo, arquitetos, testadores, etc. Em um time Scrum, temos pessoas com habilidades de arquitetura que podem ter uma habilidade secundária para ajudar nos testes. Ou seja, colaboração é a palavra de ordem.

Cada membro do time é responsável por:

- Definir a meta do *Sprint* (*Sprint Goal*)
- Comprometer-se com o trabalho, e fazê-lo com o máximo de qualidade
- Trabalhar tendo em mente a visão do produto e o objetivo do *Sprint*
- Colaborar com demais membros do time e ajudar o time a se auto-organizar
- Comparecer às reuniões do *Daily Meeting*
- Levantar impedimentos
- Estimar requisitos
- Manter seu próprio progresso (com ajuda do *Scrum Master*)

As equipes de Scrum têm o tamanho ideal de 7 +/– 2 pessoas. 7 é o número ótimo. Times pequenos são mais produtivos que times grandes, por causa da quantidade de vias de comunicação existente entre os membros do grupo. Se o seu time for maior que o recomendado, então ele deve se dividir em dois e o trabalho de ambos deve ser sincronizado. Outro ponto importante a lembrar é manter a estabilidade do time durante o *Sprint*, ou seja, evitar mudanças na equipe no meio de um *Sprint*.

2.4.6. Cerimônias

As cerimônias do Scrum são reuniões, formais ou não, que acontecem em momentos específicos do *Sprint*. A seguir, falaremos sobre as seguintes cerimônias do Scrum:

- *Sprint Planning*
- *Daily Meeting*
- *Sprint Review*
- *Sprint Retrospective*

2.4.6.1. Sprint Planning

A reunião de *Sprint Planning* é uma reunião de curta duração (geralmente 4 horas), e tem como objetivo planejar o trabalho da equipe durante o *Sprint*. O dia em que ocorre a reunião de *Sprint Planning* é considerado o primeiro dia do *Sprint*. Esta reunião é dividida normalmente em 2 partes: *Sprint Planning* I e II, de igual duração.

Nesta reunião, participam somente as pessoas comprometidas com a entrega. Pessoas interessadas ou especialistas no negócio podem ser convocadas à reunião pontualmente, entretanto devem se retirar após responder os questionamentos ou prestar os esclarecimentos devidos.

2.4.6.1.1. Sprint Planning I

- Participantes:
    - Equipe (*Team*)
    - *Scrum Master*
    - *Product Owner*
- Entradas:
    - *Product Backlog* (Estimado e Priorizado)
    - Dados históricos de capacidade da equipe (se houver)
    - Condições do negócio
- Saídas
    - *Selected Product Backlog*
    - Objetivo do próximo *Sprint*
    - Equipe comprometida e com entendimento dos requisitos

O objetivo do *Sprint* é decidido pelo *Product Owner* em conjunto com o time. Todos discutem os itens de *Backlog* priorizados e estimados e definem o objetivo, que deve ser uma frase simples e objetiva. Por exemplo:

"Permitir que pessoas se cadastrem em nossa newsletter corporativa e se descadastrem caso eles não desejem mais recebê-la."

Esta reunião também tem como saída o comprometimento da equipe em realizar o trabalho proposto durante o *Sprint*. A equipe, e somente a equipe, pode decidir e se comprometer a respeito do trabalho que será executado, seja a partir de dados históricos de capacidade da equipe, ou a partir da capacidade que a equipe espera atingir no *Sprint*. Em outras palavras, o *Product Owner* prioriza e seleciona, mas só a equipe pode dizer o que será atingido durante o *Sprint*. Se o *Product Owner* ficar insatisfeito, tudo o que ele pode fazer é repriorizar as tarefas e ajustar o objetivo do *Sprint*.

2.4.6.1.2. Sprint Planning II

Na reunião de *Planning* II, participam somente as pessoas que efetivamente irão realizar o trabalho. Aqui já não cabe mais a participação do *Product Owner*, somente da equipe e do *Scrum Master*. O *Product Owner* pode ser consultado durante o *Sprint Planning* II caso ocorram dúvidas a respeito dos requisitos selecionados.

- Participantes
    - Equipe (*Team*)
    - *Scrum Master*
- Entradas
    - *Selected Product Backlog*
    - Objetivo do próximo *Sprint*
- Saídas
    - *Sprint Backlog*

O *Sprint Backlog* é uma expansão do *Product Backlog* selecionado durante a reunião de *Planning* I. A equipe vai ler, de um por um, os itens selecionados, e decompor em tarefas necessárias para considerar o item de *Backlog* implementado, pronto. Este trabalho é feito colaborativamente, com a participação de toda a equipe. É recomendável manter as tarefas num nível de granularidade suficiente para admitir tarefas de 4 ou 8 horas (ou seja, no máximo 1 dia de

trabalho). Estas tarefas podem ser estimadas individualmente, ou pode-se assumir o tamanho do requisito previamente estimado (isso fica à cargo da equipe).

É importante firmar o conceito de "pronto" (*done*), para garantir o entendimento de todos. Pronto significa que o requisito foi implementado e testado, e está num estado entregável, caso seja necessário. Ter isto firmado em mente é importante porque na eventualidade de um item de *Backlog* ter ficado parcialmente implementado, digamos, 80% num *Sprint*, isso significa que ele não está pronto e deverá ser selecionado como item de *Backlog* para o próximo *Sprint*.

No término desta cerimônia, temos a equipe comprometida, uma meta de velocidade pré-estabelecida e o trabalho decomposto em tarefas de granularidade suficiente para serem trabalhadas.

2.4.6.2. Executando o Sprint

Após a reunião de *Planning* I e II, a equipe está pronta para iniciar o trabalho, auto-organizando-se para decidir quem executa o quê e quando. É importante que os membros da equipe se comuniquem para efetivar esta auto-organização.

Durante esta fase, que dura entre 2-4 semanas, é que o trabalho será executado e impedimentos serão levantados. Aqui é que o trabalho do *Scrum Master* será mais evidenciado, agindo como um desobstruidor, protetor e mentor. De maneira a manter todos informados, é feita uma cerimônia de acompanhamento diário, o *Daily Meeting*, sobre a qual vamos conhecer mais adiante. É também durante a fase de execução que o *Scrum Master* deve agir como um escudo para a equipe, evitando que trabalho não planejado ou novos itens de *Backlog* sejam incluídos no *Sprint*. Vamos para um exemplo:

Estamos trabalhando em um *Sprint* de 4 semanas. Uma nova solicitação de negócio chegou para o *Product Owner*, com um grau de importância alto. O *Product Owner* então, apressa-se em comunicar ao time que eles devem começar a trabalhar nesta nova funcionalidade/solicitação agora.

Isso é disruptivo para o time, e introduz instabilidade. O time já gastou tempo planejando o *Sprint*, e está mentalmente focado em atingir o objetivo do *Sprint*, e mais importante, entregar software funcionando. O *Product Owner* deve tentar conversar com o cliente, e explicar como isso irá interferir no trabalho do

time, e priorizar esta demanda no *Product Backlog* para ser trabalhado no próximo *Sprint*.

Se um bug crítico foi encontrado, e a não correção deste bug trará maiores consequências à organização ou a seus clientes (por exemplo, "pagamentos de cartão de crédito estão sendo duplicados"), então é compreensível que um item do *Sprint Backlog* seja retornado ao *Product Backlog* e substituído pela nova demanda.

Se por alguma razão, a situação for muito caótica, então é melhor dissolver o *Sprint* e começar novamente com uma nova reunião de *Sprint Planning*. Garanta que uma reunião de retrospectiva desse *Sprint* prematuramente terminada irá acontecer, para que possam ser encontradas as causas reais desse acontecimento. A prática de interromper um *Sprint* deve ser adotada como último recurso. Ao ser aplicada alguns fatores que possivelmente causariam a aplicação desse recurso devem ser observados, como por exemplo:

- Planejamento pobre
- Priorização não está correta
- Falta de entendimento e de suporte das práticas ágeis
- Ambiente de negócio passou por mudança massiva
- Políticas corporativas
- Não há um acompanhamento cíclico do cliente quanto à evolução do *Backlog*

2.4.6.3. Daily Meeting

A reunião de *Daily Meeting* (ou *Daily Scrum*) é um encontro público da equipe, de curtíssima duração (no máximo 15 minutos). Nesta reunião são admitidos membros do time e interessados no projeto, entretanto tudo que os interessados podem fazer é observar, sem fazer comentários nem perguntas.

Seguem algumas regras:

- Reunião com frequência diária
- Duração: no máximo 15 minutos
- Mesmo local e horário, todos os dias
- Participantes:

- *Scrum Master, Team, Product Owner*
- Interessados (somente como ouvintes, não podem interromper sob hipótese alguma)

• Entradas são geradas a partir de três perguntas:

- O que eu tenho atingido desde a nossa última reunião?
- O que me proponho a atingir até a próxima reunião?
- Quais problemas estão me impedindo ou atrapalhando na realização do meu trabalho?

• Como saída, temos impedimentos e decisões tomadas.

Os principais benefícios que podemos enumerar desta reunião são:

• Visibilidade para todo o grupo: todos sabem o que está acontecendo

• Ajudar o *Scrum Master* a identificar impedimentos, para que os mesmos possam ser resolvidos e proteger a produtividade do time.

É obrigação do *Scrum Master* trabalhar em cima dos itens de impedimento levantados durante a reunião, e escalá-los à alta direção caso ele não seja capaz de resolver. Impedimentos detém que o time realize seu trabalho em um ambiente ótimo, e podem comprometer o objetivo do *Sprint*.

É função do mediador da reunião (normalmente o *Scrum Master*) manter discussões técnicas ou a respeito de solução de problemas de fora, para que sejam por exemplo feitas imediatamente após a reunião. Muitas vezes pessoas podem se alongar demais na discussão de um problema técnico, e isso pode atrapalhar o bom andamento e o propósito da reunião. O mediador da reunião deve declarar claramente para os envolvidos o término do *Daily Meeting*, deixando assim o time à vontade para discutir problemas ou questões técnicas, ou para agendar uma outra reunião para discussão do tema entre os envolvidos.

2.4.6.4. Sprint Review

Esta cerimônia é realizada no último dia do *Sprint*. É uma reunião informal (em geral 4 horas) onde todos são convidados a participar, principalmente os clientes do projeto. Normalmente é realizada numa sala de reunião ou auditório, de portas abertas, e o objetivo desta reunião é dar a oportunidade à equipe para apresentar o resultado do trabalho realizado durante o *Sprint*. Tipicamente acontece como uma demonstração do incremento de funcionalidade, ou da sua arquitetura informal. A equipe tem no máximo 1 hora para se preparar para esta reunião, na qual deve ser apresentada a demo do sistema.

Funcionalidade que não ficou pronta (aqui, mais uma vez, destaco para a importância de alinhar o significado de "pronto" entre todos os envolvidos, pois este conceito pode variar de organização para organização) não pode ser apresentada. Artefatos que não são funcionalidade também não podem ser apresentados, com exceção de quando eles são usados como suporte para entendimento da funcionalidade demonstrada. Artefatos não podem ser demonstrados como produtos de trabalho, e seu uso deve ser minimizado para evitar confundir os clientes, ou requerer que eles entendam como funciona desenvolvimento de sistemas.

O ambiente de demonstração deve ser o mais próximo do ambiente de produção. Na maioria do tempo da reunião, membros do time irão apresentar funcionalidades e responder perguntas dos clientes a respeito dos produtos apresentados. No final desta reunião, os clientes são questionados, um por um, e devem declarar suas impressões, mudanças desejadas e a prioridade dessas mudanças. O *Product Owner* então discute com os clientes e o time sobre a potencial reorganização do *Product Backlog* baseado no feedback recebido.

Adicionalmente, os clientes podem apontar e identificar funcionalidades que não foram entregues, ou não foram entregues conforme esperado, e solicitar que esta funcionalidade em questão seja retornada ao *Product Backlog* para repriorização. Além disso, os clientes também podem solicitar que novas funcionalidades sejam adicionadas ao *Product Backlog*. O objetivo desta reunião é claro: apresentar o incremento de funcionalidade atingido durante o *Sprint*, e receber feedback dos clientes, de forma a repriorizar o *Product Backlog* e adicionar novos itens ao mesmo.

2.4.6.5. Sprint Retrospective

Esta é uma reunião formal, fechada (geralmente 3 horas). Participam dela somente Time, *Scrum Master* e *Product Owner*, sendo a presença deste último opcional.

Todos os membros do time devem responder à três perguntas:

- O que foi bem?
- O que pode melhorar?
    - Quem tem controle do que pode melhorar? (Time/organização)

• Neste ponto, é preciso tomar cuidado. A função da retrospectiva não é apontar culpado. Deve ser definida somente a responsabilidade da implementação da melhoria: do time, ou da organização.

Após isso, as oportunidades de melhoria detectadas devem ser priorizadas. A partir dessas perguntas, e da priorização, serão inferidas as seguintes saídas:

• *Team Backlog*, o que o Time pode mudar no processo de forma a melhorá-lo.

• Impedimentos abertos que não estão no controle do time, devem ser trabalhadas pelo *Scrum Master*.

Ambos ordenados por prioridade. O *Scrum Master* não está nesta reunião para prover respostas, mas para facilitar a busca do time por melhores maneiras de fazer com que o processo do Scrum trabalhe para o Time. Itens viáveis podem ser adicionados à próximo *Sprint*, como *Product Backlog* não-funcional de alta prioridade.

2.4.7. Artefatos do Scrum

No Scrum, existem somente 3 artefatos requeridos:

• *Product Backlog*

• *Sprint Backog*

• *Burnup/Burndown Charts*

2.4.7.1. Product Backlog

O *Product Backlog* é uma lista de todas as funcionalidades desejadas no produto, estimadas pelo time e priorizadas pelo *Product Owner*. Quando um projeto é iniciado, não é possível escrever todos e quaisquer requisitos que porventura um dia serão incorporados ao sistema. Tipicamente, escrevem-se primeiro os requisitos que são mais óbvios, o que é normalmente mais do que suficiente para o primeiro *Sprint*.

O *Product Backlog* então deve crescer e mudar à medida em que se aprende mais sobre o produto, seus clientes e o negócio, ou seja, é emergente. Os itens de maior prioridade são geralmente mais detalhados, e o mesmo é mantido de uma forma visível para todo o time e o *Scrum Master*. Geralmente, qualquer pessoa pode contribuir com o *Product Backlog*, mas as solicitações devem ser sempre priorizadas pelo *Product Owner*.

Exemplo de um *Product Backlog*, já estimado, priorizado e com valor de negócio definido:

#	Prioridade	Descrição	Estimativa	Valor de negócio
1	Muito Alta	Como gestor, quero controle de acesso por login e senha para acessar o sistema, para controlar o acesso	13	100
2	Muito Alta	Como usuário, quero poder cadastrar fornecedores no sistema, para poder encontrar facilmente seus contatos	5	80
	Alta	Como usuário quero poder cadastrar notas fiscais no sistema, associando-as ao fornecedor, para poder manter registro das notas	8	90
	Baixa	Como operador, desejo tirar backup dos dados do sistema a qualquer momento, para poder garantir a segurança dos dados	5	30
	Média	Como vendedor, desejo poder registrar as vendas que faço sobre meu login, para poder administrar minha comissão.	3	60

Tabela 1 – Exemplo de Product Backlog.

2.4.7.2. Sprint Backlog

O *Sprint Backlog* é a lista de histórias que o time se comprometeu com o *Product Owner* a implementar durante o *Sprint*, após a reunião de *Sprint Planning* I e II. Dependendo do tamanho de uma história em particular, ela pode ser quebrada em duas ou mais histórias menores para facilitar a entrega em etapas.

O *Sprint Backlog* é escolhido pelo time dado o objetivo e prioridades descritos pelo *Product Owner*. Cabe somente ao time decidir quais histórias implementará durante o *Sprint*. Como é o time que se compromete com o objetivo do *Sprint* é ele que decide quais histórias deverão ser feitas para cumprir esse objetivo. O *Sprint Backlog* pode ser mantido da maneira como for mais conveniente para a organização: pode ser numa planilha de Excel, ou num sistema de controle de tarefas, ou até somente no quadro do time.

2.4.7.3. Burndown/Burnup Charts

Os gráficos de *Burndown* ou *Burnup* são as melhores ferramentas do time para manter registro da velocidade atual do trabalho. Esses gráficos geralmente ficam no quadro do time, ou perto dele, e podem ser mantidos manualmente ou via Excel.

Existem várias formas de se representar um gráfico de *burndown* ou *burnup*, a única diferença em ambos é que, no gráfico de *burndown*, a linha representa o quanto de trabalho ainda falta para concluir o *Sprint*, e no gráfico de *burnup* a linha representa quanto de trabalho já foi feito no *Sprint*, considerando que o montante de trabalho é contabilizado pelo fator de complexidade adicionado a cada história durante as estimativas (mais detalhadas a diante). Os gráficos são ferramentas fundamentais para o time se auto-gerenciar, pois permitem de uma certa forma prever o sucesso de um *Sprint*. O gráfico do *Sprint* deve ser mantido pela própria equipe, e atualizado diariamente.

Exemplo de gráfico de *Sprint Burnup*:

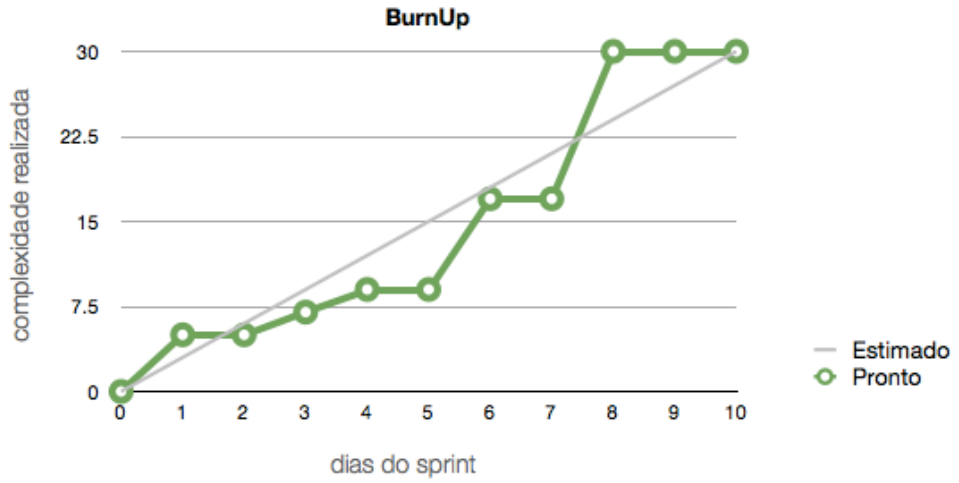

Figura 2 – Exemplo de Sprint BurnUp

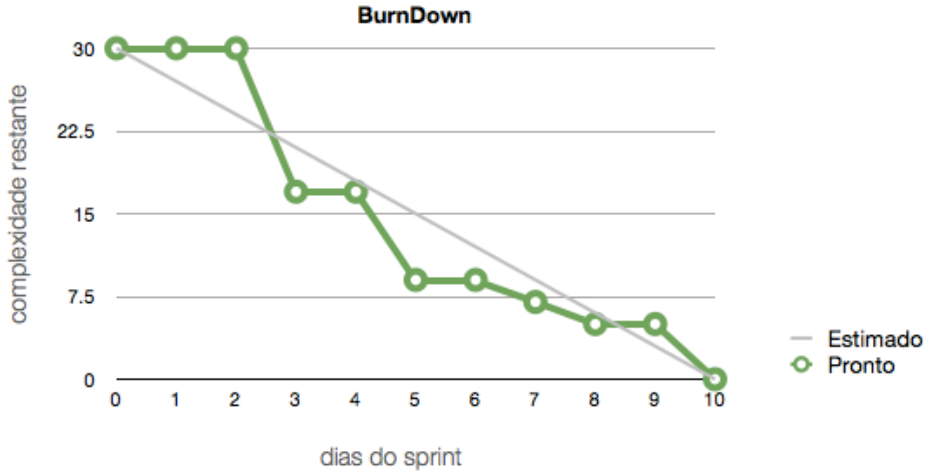

Figura 3 – Exemplo de Sprint BurnDown

Além dos gráficos do *Sprint*, a equipe também pode decidir monitorar sua velocidade entre *Sprints*, utilizando-se de um gráfico de *Product Burndown* ou *Burnup*. Neste caso, a diferença entre *Burndown* e *Burnup* é a mesma, a única diferença é que neste caso, será monitorado o tamanho do trabalho que a equipe consegue entregar a cada *Sprint*. Este gráfico também dá visibilidade ao *Product Owner* e os *Stakeholders* que estão patrocinando o projeto sobre a velocidade do time por *Sprint*.

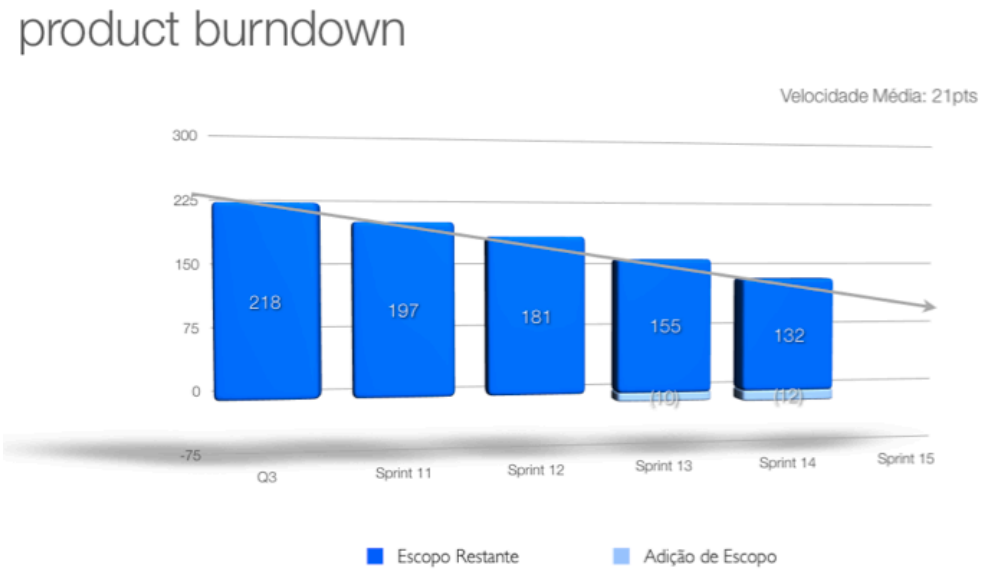

Figura 4 – Exemplo de Product Burndown

2.4.8. Estimativas – Planning Poker

Antes de começar a falar sobre estimativas, gostaria de chamar a atenção para o significado da palavra "Estimativa".

O que é uma estimativa?

1. Um cálculo aproximado
2. Uma declaração por escrito do preço aproximado que será cobrado para um trabalho específico.
3. Um julgamento ou avaliação

Fonte: Oxford Dictionary

Mais importante, estimativas são inerentemente erradas. Estimativas não são exatas, e não refletem a realidade. A única maneira de saber o tempo exato que uma atividade vai durar, ou quanto esforço será necessário despender para executar algum trabalho, é após a sua realização. Dito isso, vamos prosseguir a respeito do *Planning Poker*.

Uma das maneiras mais eficientes e recomendadas de estimar o tamanho de requisitos em times que adotam métodos ágeis (XP/Scrum) é o jogo de cartas

conhecido como *Planning Poker* [Cohn, 2005]. Este método de estimar combina opinião de especialistas, analogias e desagregação em uma aproximação eficiente para estimar de maneira rápida, porém confiável. É uma variação do método de estimativa *Wideband Delphi* (1940).

As estimativas acontecem normalmente em uma reunião (geralmente 4 ou 8 horas, dependendo da quantidade de requisitos a ser estimada). Os participantes do jogo de poker são todos os membros do time do Scrum. O *Product Owner* normalmente comparece a esta reunião para prestar esclarecimentos a respeito dos requisitos, porém não pode opinar e estimar junto com o time. O *Scrum Master* registra os resultados da reunião, e assim como o *Product Owner*, não interefe nas estimativas do time.

No início de uma reunião de *Planning Poker*, cada membro do time recebe um deck de cartas, que contém uma variação da sequencia de Fibonacci, e uma última carta que representa a não possibilidade de estimativa: 0, ½, 1, 2, 3, 5, 8, 13, 20, 40, 50, 75, 90, 100, ?. Ainda no início, todos os itens a serem estimados são lidos pelo *Product Owner* ou *Scrum Master*, e a equipe conversa para decidir qual é o menor item de *Backlog* disponível.

Após essa estimativa inicial, esse item é marcado como "2" pontos, e a equipe prossegue para estimar os demais itens de *Backlog*. Isso serve para definir uma referência de tamanho e complexidade para ser utilizada nas demais estimativas. Isso só precisa ser definido na primeira reunião de Estimativa, e deve ficar registrado, para uso nas futuras reuniões. Em casos excepcionais, o time pode decidir mudar a história de referência por uma outra história. Aqui também é importante usar o bom-senso.

Seguindo adiante, para cada história a ser estimada, o *Scrum Master* ou *Product Owner* lê a descrição e os critérios de aceitação da mesma. O *Product Owner* responde a quaisquer questionamentos que o time possua a respeito da história, entretanto procurando manter o nível da discussão em alto nível, e não entrar em detalhes. É comum definir também um *timebox* para esse tempo de perguntas e respostas, de modo a não se estender demais e perder muito tempo em uma única história.

Depois da apresentação e discussão inicial, cada pessoa seleciona sua estimativa (uma carta), baseada em sua opinião pessoal a respeito do tamanho e complexidade da história, e a coloca em cima da mesa, com a face voltada para

baixo. Quando todos os participantes selecionarem uma carta, as mesmas serão viradas ao mesmo tempo.

Se houver uma grande variância entre a maior e menor carta, o time vai discutir o que cada membro estava pensando quando selecionou o tamanho. A ideia é que todos entendam as razões, que mais uma ou outra pergunta sejam respondidas pelo *Product Owner*, e a equipe volta a pensar sobre a história, e faz uma nova estimativa. O ciclo deve ser repetido até que todas as estimativas estejam próximas, ou que a equipe chegue a um consenso. É comum definir um limite de rodadas de poker, para evitar que as coisas fiquem muito arrastadas. Repetir até que todas as histórias estejam 100% estimadas!

2.4.9. Escrevendo Histórias

Uma história de usuário, ou *user story*, é um requisito de sistemas de software formulado com uma ou duas sentenças em linguagem natural. Cada *user story* é limitada e pequena, de forma a caber perfeitamente em um pequeno papel de *post-it*. Isso é feito para de forma a garantir que histórias muito grandes sejam sempre quebradas e granularizadas.

User stories são uma maneira rápida de lidar com requisitos do cliente, sem ter que elaborar documentos de requisito formais e vastos, e sem executar tarefas administrativas super-dimensionadas para mantê-lo. A intenção com a história é ser capaz de responder mais rápido e com menos overhead as mudanças nos requisitos voláteis do mundo real.

Uma história é uma declaração informal do requisito, que em segue normalmente o seguinte formato:

Como um <papel do usuário/ator> quero <funcionalidade> para <valor de negócio>

Na parte de trás da história, normalmente são escritos os critérios de aceitação. Critérios de aceitação funcionam como um balizador de entendimento do que é “pronto” entre o desenvolvedor e o cliente, para aquele requisito específico. É importante frisar que histórias não são requisitos do IEEE nem Casos de Uso (*use cases*).

Exemplo de uma história:

"Como Gestor, quero que as informações pessoais dos clientes fiquem gravadas em formato criptografado no banco de dados, para garantir a privacidade e a segurança dos dados dos meus clientes

Critérios de aceitação:

- Ter os dados armazenados no banco de dados e arquivos de troca do sistema usando algoritmo de criptografia do tipo chave publica/chave privada. "

2.4.10. Scrum of Scrums

"*Scrum of Scrums* é uma importante técnica para escalar Scrum em grande times e projetos. Essas reuniões permitem agrupar os times para discutir seus trabalhos, focando especialmente em área de sobreposição e integração". [Cohn, 2005]

Essa reunião tem por objetivo alinhar informações técnicas entre os times de um mesmo projeto, bem como discutir eventuais problemas ou decisões técnicas.

Imagine um projeto contendo sete times, cada um com sete membros, cada time irá conduzir seu próprio *Daily Meeting*. Cada time elege uma pessoa para fazer parte da reunião do *Scrum of Scrums*, essa deverá ser uma decisão do próprio time, por reconhecerem um membro capaz de dar posição técnica das decisões do time e opinar nas decisões dos demais. Não é aconselhável que o *Scrum Master* ou *Product Owner* sejam os eleitos a participarem dessa reunião, exatamente pelo fato das decisões técnicas deverem ser tomadas pelo Time. Uma vez que o representante foi escolhido, ele deverá ter uma sequência de participações, mas não está preso a essa reunião, eventualmente outro membro do time poderá acompanhá-la. Caso o time já saiba que existe uma determinada necessidade específica de conhecimento dos outros times, como arquitetura ou banco de dados, poderá previamente solicitar a presença de membros dos outros times que tenham o conhecimento específico do problema para facilitar a solução e o alinhamento entre os Times.

O *Scrum of Scrums* pode ser feito também de forma recursiva, principalmente quando existem muitos times envolvidos no projeto. Sendo

possível dividir em diversos grupos por temas, componentes ou funcionalidades. A reunião inicial ocorre somente dentro dos temas e na sequência cada representante do tema se reúne em um novo *Scrum of Scrums*.

Caso o número de participantes na reunião seja pequeno, tanto por poucos times envolvidos no projeto, ou por uma recursão do *Scrum of Scrums*, poderá ser enviado mais de um representante por time, dessa forma é possível enriquecer mais a reunião. Essa é uma medida que deverá ser tomada caso a caso, porque muitas pessoas participando dessa reunião poderá também fazer com que deixe de ser efetiva.

Outra reunião similar ao *Scrum of Scrums* também muito importante é o agrupamento frequente por especialidade, a fim de manter um nivelamento de conhecimento, melhores praticas e padrões. Dessa forma podemos manter a arquitetura, design, componentes *client-side*, banco de dados mais alinhados entre times. Isso não é somente interessante do ponto de vista de unicidade no projeto como também facilita a transição de membros entre times, por manter o mesmo contexto base.

A frequência dessa reunião deverá ser determinada pelo time, mas é importante lembrar que quanto maior a distância entre reuniões fatalmente aumentará seu tamanho, e eventualmente algumas informações podem ser perdidas.

Principalmente no início do projeto, recomenda-se que o *Scrum of Scrums* seja executado diariamente, ao fim de todos os *Daily Meetings*. [Schwaber, 2007]

É importante que os times foquem na colaboração entre si durante o *Scrum of Scrums*, e no alinhamento das expectativas de integração entre componentes correlacionados.

Se fizermos um paralelo do *Scrum of Scrums* com o *Daily Metting* [Cohn, 2005], as perguntas diárias deverão ter uma leve mudança, além do acréscimo de uma muito importante para essa reunião:

• O que o seu time fez desde a última reunião?

• O que o seu time irá fazer depois desta reunião?

• Algo está diminuindo a velocidade o interrompendo o trabalho do seu time?

• Seu time está a ponto de fazer (ou deixando de fazer) algo que de alguma forma irá impactar outros times?

A última pergunta pode ser extremamente útil no que diz respeito a coordenação de múltiplos times em um projeto, e da expectativa entre times. Eventualmente um time não tem plena visão da profundidade do impacto que poderá causar em outro time por uma decisão técnica, que em algumas situações poderia tomar outra solução.

A idéia das perguntas, como no *Daily Meeting* é apenas para dar um ritmo rápido e efetivo a reunião. Questões mais profundas deverão ser endereçadas após todos passarem por esse pequeno processo. Dessa forma pode ser otimizado o tempo daqueles que eventualmente não serão impactados por um determinado problema. É importante no segundo momento, aonde as discussões tomam profundidades, levantar ações práticas e seus responsáveis para que esses problemas possam ser resolvidos ou que os levantamentos necessários possam ser feitos até o próximo encontro.

2.5. Kanban

Em 2004, David Anderson adaptou o modelo Toyota de controle enxuto de produção utilizando como base a "teoria das restrições", que de maneira bem simplificada diz que se for possível identificar um gargalo em seu processo, dever-se-ia focar toda a atenção para aliviar esse gargalo [Anderson, 2003]. Dessa forma, Kanban se utiliza de filas de estágios (ou etapas) de processo com limites determinados e o acompanhamento da evolução de cada item em cada uma desses estágios, com o intuito de expor gargalos nesse fluxo para que imediatamente sejam trabalhados.

Existem 5 propriedades no Kanban [Anderson, 2010].

2.5.1. Visualização do Fluxo

Visualizar o fluxo de trabalho e fazê-lo visível constantemente é fundamental para o entendimento de como funciona o trabalho e seu fluxo. Torna-se mais difícil fazer mudanças sem o entendimento correto desse fluxo. Uma maneira comum de visualizar o fluxo é utilizando cartões e um quadro com colunas. As colunas no quadro representam os diferentes estágios deste fluxo e os cartões brancos as funcionalidades ou histórias. Ainda na representação abaixo é possível observar figuras que representam o desenvolvedor responsável pela história em questão. Os marcadores em amarelo representam tarefas detalhadas das histórias. Podem ser adicionadas outras representações visuais nas histórias dependendo do seu cenário, como uma estrela para identificar uma história urgente, ou uma marcação vermelha para indicar um impedimento.

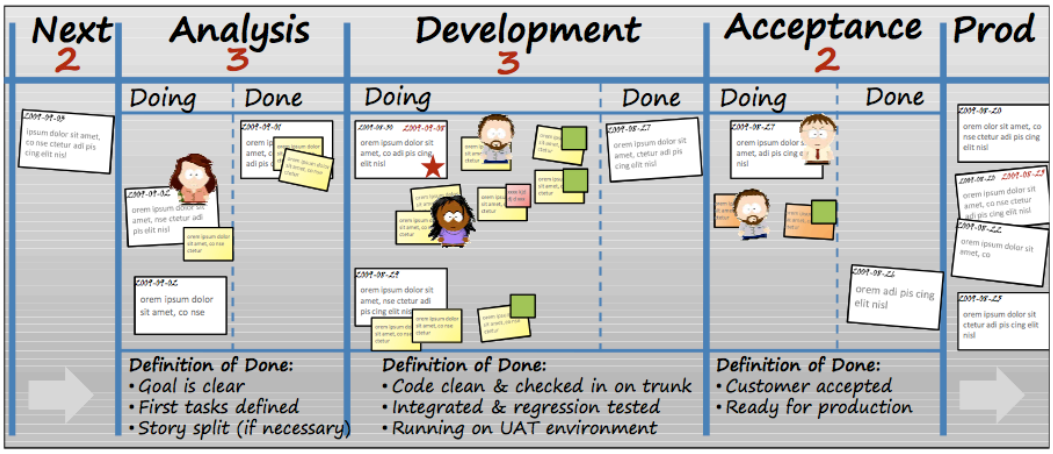

Figura 5 – Exemplo de um quadro Kanban

[Kniberg, 2009]

2.5.2. Limite de trabalho em andamento

O número em vermelho, abaixo da descrição de cada etapa no exemplo de quadro acima, mostra a quantidade de cartões máxima para cada etapa de desenvolvimento. Uma vez observado que uma etapa não pode receber novos

cartões, deverá ser observado o que está evitando o andamento correto do fluxo, para que seja trabalhado pelos membros do time.

2.5.3. Administração do Fluxo

O Fluxo de trabalho por cada estágio deve ser monitorado, medido e exposto. Ao medir o tempo de um cartão a cada estágio, poderá ser notado de forma clara o impacto (negativo ou positivo) de eventuais mudanças no processo.

Uma forma comum de monitorar esses tempos é anotar no verso dos cartões a data que a história foi criada, passou por cada estágio e finalmente foi concluída.

2.5.4. Processo e políticas explicitas

Todo o sistema de trabalho é explicito no quadro. Os estágios de desenvolvimento, a definição de pronto, quem está fazendo qual atividade e principalmente os limites de trabalho em andamento de cada estágio.

2.5.5. Melhoria colaborativa

Kanban encoraja pequenas mudanças de forma incremental e contínua. O time, tendo uma visão compartilhada de seu trabalho, riscos, fluxo e processo, deve sugerir mudanças assim que perceber uma oportunidade de melhoria. Essas mudanças devem ser feitas em consenso, aonde todo o time participa da decisão.

3 Boas práticas para o emprego de Scrum

Dean Leffingwell no seu livro *Scaling Software Agility*, descreve sete práticas para escalar o uso de Scrum.

3.1. Definir. Construir. Testar

Para construir código funcionado em um curto período de tempo, as equipes precisam se organizar para conter três capacidades, fundamentais para entregar software.

• Capacidade para agir nas definições e como ponte entre o cliente e o próprio time, tarefa ao qual o *Product Owner* é incumbido.

• Criar código com o mínimo de distração e troca de contexto entre projetos. E para esse caso desenvolvedores capacitados e protegidos por um *Scrum Master*.

• Testes, de forma integral, automatizados além de capacidade em testes manuais.

Todo método ágil divide grandes conjuntos de trabalho em pequenos pedaços, representados como histórias, casos de uso, itens de *Backlog*, enfim, uma lista de "coisas" a serem feitas. O objetivo do time é definir, construir e testar cada uma dessas "coisas" em um tempo pré-estabelecido (*time box*).

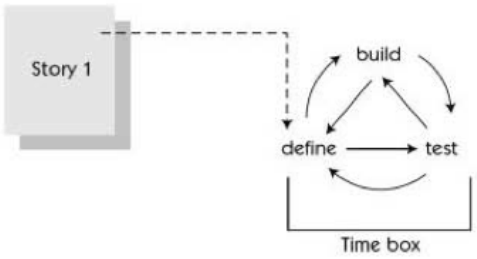

Figura 6 – Definir. Construir. Testar

[Leffingwell, 2007]

3.1.1. Definir

Não importa o quão bem elaborada está uma história, os desenvolvedores irão interagir com o *Product Owner* para entender o que ela quer dizer. Durante esse processo algum design já estará presente na imaginação de cada desenvolvedor, e não estando certamente será criada.

Nesse processo não somente o design de código começa a surgir, os requisitos também podem ser modificados para facilitar o desenvolvimento no período de tempo disponível. É possível inclusive acrescentar novas idéias, enriquecendo o requisito original.

Figura 7 – Retro alimentação de requisitos e design

[Leffingwell, 2007]

3.1.2. Construir

A construção propriamente dita. Mesmo durante o desenvolvimento a equipe se mantém em contato com o *Product Owner* para validar eventuais questões de definição, havendo inclusive a possibilidade de redefinir a própria história ou parte dela.

3.1.3. Testar

A história não estará completa até que esteja testada e aceita. Em geral criam-se testes automatizados para garantir que eventuais desenvolvimentos no entorno da história em questão não gerem quebra. O próprio teste servirá também como documentação técnica do funcionamento de cada componente gerado pela a história em questão.

Se considerarmos a prática de TDD (*Test Driven Development*) Construir e Testar não serão necessariamente sequenciais, uma vez que, para cada parte do código, os testes são criados antes do código em si.

Para que exista esse fluxo de forma constante, é necessário que o time seja composto por membros capazes de desenvolver integralmente a funcionalidade ou história em questão.

"Um time deve ser multifuncional, contendo todo o conhecimento necessário para atingir o objetivo daquela iteração" [Schwaber / Beedle, 2001].

Uma das características comuns às empresas e que dificulta a adoção de métodos ágeis, é a separação de funções em silos que são:

- Otimizados para comunicação vertical através de gerência.
- Gera facilmente fricção entre silos interdependentes
- Seus profissionais são alocados e agrupados por função
- Naturalmente gerando barreiras políticas entre funções

Com métodos ágeis a empresa precisa se reorganizar para que cada time tenha as habilidades para definir, desenvolver, testar e entregar cada história. Nesse caso é fundamental quebrar esses silos e gerar um time, comprometido com a entrega, contendo representantes de cada uma das partes envolvidas [Leffingwell, 2007].

É possível ainda organizar cada time por responsabilidade, componentes, funcionalidades ou serviço.

3.2. Dois níveis de planejamento

Por mais que o desenvolvimento ágil se afaste de planejamento detalhado muito a frente, não descarta a possibilidade de criar um plano a longo prazo, pensando em entregas em pequenas iterações e de uma forma flexível, isso é, o plano deve ser contínuo sendo revisado a cada iteração para, caso necessário ajustá-lo.

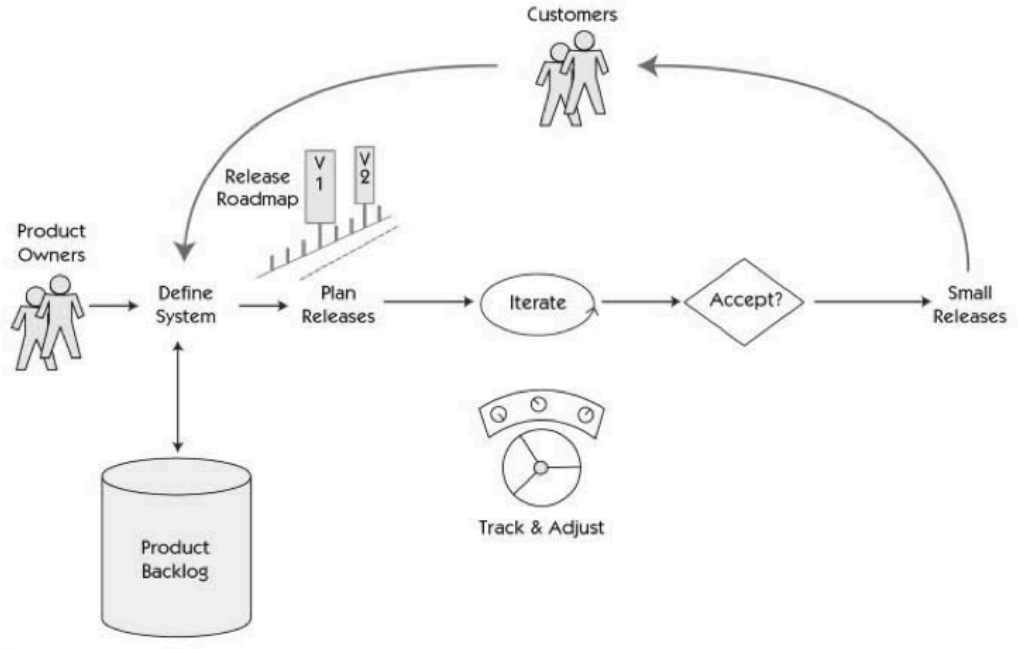

Figura 8 – Planejamento contínuo

[Leffingwell, 2007]

Para garantir esse fluxo de desenvolvimento precisamos dividir o planejamento em duas partes, divididas em Release e Iteração.

A primeira parte é no nível de release, em que se destacam as macro-funcionalidades ou temas, observadas a partir do *Product Backlog*. Ela abrange um grupo de iterações, tendo como entrega macro-funcionalidades.

A figura abaixo representa uma parte do acompanhamento da evolução do *roadmap* trimestral de um dos times de desenvolvimento da globo.com.

Nele podemos observar temas e macro funcionalidades (*features*) e a sua evolução. Por exemplo:

Um dos temas que podemos observar é a "Página de Tópico", para que esse tema seja desenvolvido serão necessárias quatro macro funcionalidades, "Página Básica", "Template Padrão", "Template Pessoa" e "Otimização de montagem de página".

Cada uma das macro funcionalidades pode representar uma ou diversas histórias no contexto do time.

No relatório a evolução é ilustrada por uma barra verde dentro da caixa da macro funcionalidade. A observação dessa evolução dada eventuais dificuldades de desenvolvimento podem levar com que outra macro funcionalidade perca a prioridade visto a necessidade de ajuste do *roadmap*.

Figura 9 – Acompanhamento do roadmap trimestral

A segunda fase é no nível de iteração, que deverá ser quebrado em um período menor. Nesse nível o planejamento é mais detalhado chegando ao nível das tarefas.

A iteração é guiada por um objetivo, logo, as histórias selecionadas para essa iteração deverão somadas serem capazes de atingir a esse objetivo.

Cada iteração tem inicio meio e fim. Organizados inicialmente por uma fase de planejamento, histórias contendo definição, construção e teste. Uma revisão das histórias agrupadas espelhadas no alcance ou não do objetivo e finalmente uma retrospectiva para eventuais ajustes para a próxima iteração.

Ao fim de cada iteração o primeiro planejamento (de releases) deverá ser revisitado e eventualmente reajustado.

3.3. Dominando a iteração

Uma das diferenças cruciais entre as metodologias tradicionais e ágeis é a habilidade de construir código funcional, testado em um pequeno espaço de tempo. Nesse caso cada ajuste ou nova funcionalidade torna-se potencialmente uma entrega.

Mas para que isso aconteça, é necessário que o time tenha a capacidade de dominar a iteração, dividindo suas histórias ou funcionalidades em partes que são capazes de serem produzidas no tamanho de sua iteração.

A primeira questão que os times enfrentam no que diz respeito à iteração é: Qual o tamanho ideal de uma iteração? A maioria dos livros concorda que independente do seu tamanho, iterações devem manter o mesmo formato durante um projeto, por dar melhor visibilidade de progresso ao time e facilitar a manutenção do *Release Plan*. Uma iteração muito curta pode forçar o time a paralelizar histórias, enquanto uma iteração muito longa pode não dar margem para ajustar o planejamento de forma rápida.

Em geral, duas semanas de iteração força quebrar histórias em pequenas partes, possibilita tempo suficiente para gerar código funcionando, oportunidade de ter sucesso ou falha cedo e replanejar mais rápido.

Independente de seu tamanho, toda iteração deverá ter o mesmo padrão, isso é parte da disciplina do desenvolvimento ágil.

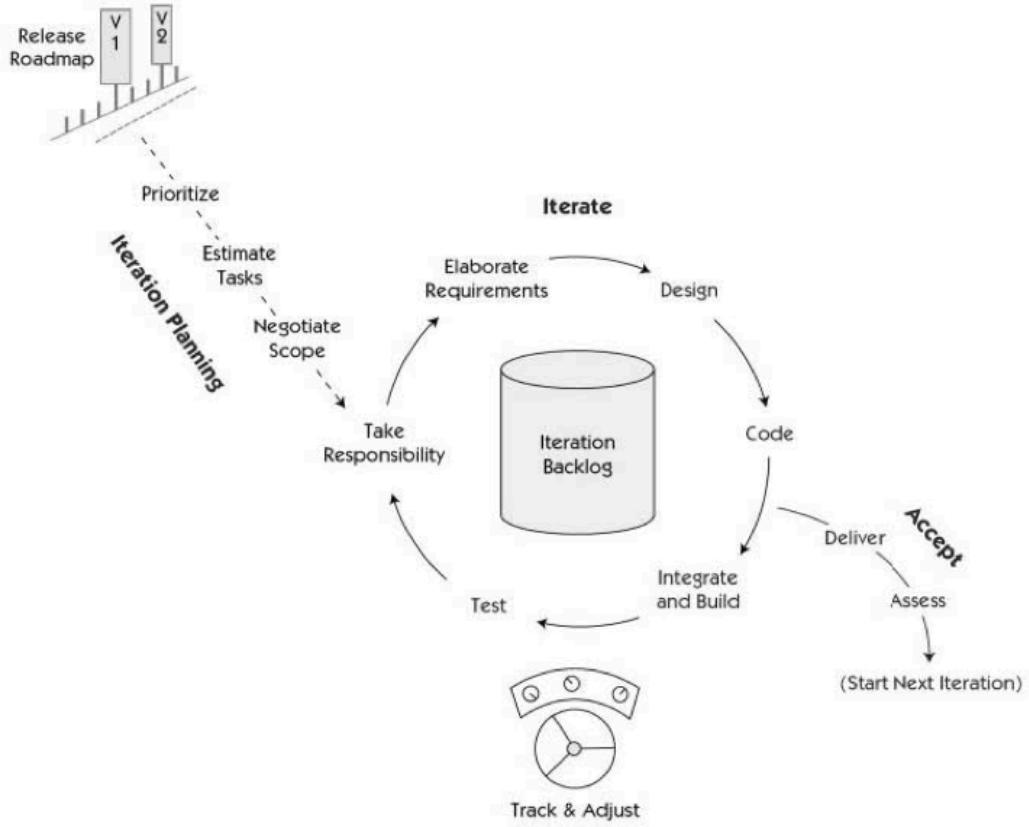

Figura 10 – Ciclo de Roadmap e Iterações

[Leffingwell, 2007]

3.4. Entregas pequenas e frequentes

Outro princípio requerido nos métodos ágeis é a participação constante dos clientes ou usuários. A melhor forma de obter essa interação é entregando software funcionando constantemente. Dessa maneira podemos observar se o caminho que o time está percorrendo precisa ou não de ajustes. Quanto mais rápido forem feitas as entregas, mais rápido absorve-se a resposta do cliente. O risco desenvolver algo não alinhado com a expectativa do cliente é

substancialmente reduzido porque ele passa a ter a habilidade de prover feedback rápido a respeito da evolução do desenvolvimento, se esse está ou não endereçado a resolver suas necessidades.

Além do cliente prover feedback rápido daquela entrega, ele também poderá trazer novos problemas, eventualmente prioritários àqueles que estão previstos nas próximas entregas.

Fazer entregas de forma frequente é uma das bases do desenvolvimento ágil, da mesma forma que as iterações, releases também precisam ser planejadas, acompanhadas e entregues aos usuários finais. Contudo esse planejamento deve ser feito de uma forma mais abstrata, considerando no máximo duas ou três releases com uma quantidade limitada de funcionalidades, exatamente para mantê-la curta e de fácil mudança.

Uma vez sendo considerada uma release baseada em data, dois pontos não deverão ser modificados: data e qualidade. Resta apenas o escopo a ser modificado dentro de uma release: redução da quantidade de funcionalidades ou simplificação de algumas delas. Considerando que os ajustes necessários durante cada interação estejam alinhados a solucionar as necessidades do cliente.

Com podemos notar, o Scrum é direcionado a ciclos datados. No modelo em cascata e modelos baseados em planos, requisitos foram fixados e times fizeram o possível para estimar os recursos necessários e a data para a qual o desenvolvimento de uma funcionalidade pré-determinada fosse atingida com esses recursos. Mas essas datas estimadas não são muito precisas e isso causa grande dificuldade para nós e organizações, que dependem desses compromissos. Na verdade essa incapacidade em prever datas realizáveis é um dos principais motivadores do desenvolvimento ágil. Reconhecemos que, para os fins mais práticos, recursos são fixos e é difícil adicionar recursos a um projeto em andamento sem reduzir sua velocidade [Brooks, 1975]. Em um modelo aonde recursos, data e qualidade são fixas, a lista de funcionalidades deverá ser variável.

O *Release Plan* é uma representação de como e quais *features* deverão ser implementadas, culminando em um *Product Release*. O plano é dividido em iterações e associado a funcionalidades de alto nível, ou histórias. Esse plano

deverá ser criado durante uma reunião a parte das tradicionais do fluxo do Scrum mas deverá ser direcionada a ser executada dentro dos fluxos das iterações.

Esse plano também inclui informações sobre a arquitetura e qualquer dependência ou outras questões que eventualmente poderiam afetar o caminho das iterações.

O resultado de um *release plan* deverá conter:

• *Goal* ou tema

• Lista priorizada de funcionalidades ou histórias

• Uma lista de iterações necessárias para entregar o release em uma visão macro

• Dependências, suposições e preocupações, ouvidas pelos membros do time que precisarão ser endereçadas

• Compromisso de todo o time

• O plano deverá estar exposto de maneira visível em uma área comum ao time.

Quando estamos falando de um projeto grande, com múltiplos times, o *Release Planning* torna-se ainda mais critico, pois existe a necessidade de alinhar todos os times em um objetivo comum. Tendo sido feito esse alinhamento, todos os times passam a trabalhar por um objetivo, suas interdependências podem ser datadas e acompanhadas a partir do progresso do *Release Plan*. Nesse caso puderam ser observados não só o progresso como a necessidade de repriorização ou simplificação de uma necessidade.

3.5. Testes simultâneos

Em se tratando de métodos ágeis, todo código deve ser um código testado. Seja unitariamente ou em forma de aceitação, performance, testes necessários a cada trecho de código devem ser produzidos e automatizados dentro de cada iteração. Em alguns modelos ágeis como XP, os times são encorajados a produzir

os testes antes mesmo do código, além de facilitar o teste, isso ajuda também a entender melhor os requisitos daquela funcionalidade.

Princípios de testes em um ambiente ágil

• Todo código é um código testado.
• Testes são escritos antes em conjunto com o próprio código.
• Teste é um esforço do time. Ambos testadores e desenvolvedores deverão escrever testes.
• Automação é uma regra, não uma exceção.

Existem diversas estratégias de testes, a maioria dos times no ambiente ágil adotam basicamente os testes a seguir:

3.5.1. Teste Unitário

Ou teste de unidade, em que se entende por unidade a menor parte testável de um sistema, é o método pelo qual desenvolvedores testam o código como um módulo único, independente. Teste unitário é um teste de baixo nível, aonde o teste deverá ter acesso ao mais interno do objeto, método ou interface que está sendo testada. É um teste de isolamento, em que não deve ser levada em consideração a sua relação com outros objetos.

Geralmente testa-se em um teste unitário diversos tipos de entrada e saída de um único método, para que possa ser verificado se a resposta do método em questão está em acordo com o esperado, e no caso de entradas inválidas que seja feito um tratamento adequado.

3.5.2. Teste de Aceitação

Refere-se à reprodução do teste feito por um membro do time, uma área responsável por testes de qualidade, *Product Owner* ou o próprio cliente. Deve validar se o novo código está em acordo com os requisitos. Diferente do teste unitário, os testes de aceitação deverão tocar os componentes ao seu entorno, deverá servir como um teste "caixa-preta" em que o que está sendo testado é o comportamento do novo código na sua relação com os demais componentes.

Como o teste unitário deverá ser adicionado a cada nova funcionalidade ou nova ação de uma funcionalidade existente.

3.5.3. Teste de Performance

Testes unitários e de aceitação fazem referencia ao comportamento de cada componente, classe ou método, mas não testam o quanto efetivo esse novo trecho de código é no que diz respeito a performance.

O teste de desempenho mostra a escalabilidade e precisão do novo componente e seu impacto na adição ao sistema.

Poderá nos dar uma visão desde consumo de CPU, memória, disco, uso de sistemas associados, como também tempo de carregamento de uma página e seu tamanho, em se tratando de desenvolvimento web.

3.6. Integração contínua

Essa não é uma pratica nova, mas torna-se mais desafiadora quando falamos em times ou projetos grandes, uma vez que a quantidade de código e mudanças a serem integradas e testadas torna-se maior.

Martin Fowler define integração contínua como:

"Integração Contínua é uma pratica de desenvolvimento de software em que os membros de um time integram seu trabalho frequentemente, geralmente cada pessoa integra pelo menos diariamente – podendo haver múltiplas integrações por dia. Cada integração é verificada por um build automatizado (incluindo testes) para detectar erros de integração o mais rápido possível. Muitos times acham que essa abordagem leva a uma significante redução nos problemas de integração e permite que um time desenvolva software coeso mais rapidamente". [Fowler, M. 2006]

Integração contínua então, é um processo suportado por ferramentas, que resulta em *builds* frequentes do sistema, considerando cada novo código adicionado. Com a integração contínua a inspeção do sistema por compiladores e testes automatizados é frequente. A resposta à adição do novo código é imediata.

Defeitos podem ser identificados e corrigidos pouco tempo depois de terem sido injetados no código ou mesmo no que diz respeito a desconformidade de especificação, havendo teste para ela.

Alguns benefícios da integração contínua podem ser vistos abaixo:

• Menos tempo gasto examinando bugs causados pelo código de uma pessoa causar reflexo no código de outra.

• Descoberta rápida de falha de entendimento entre desenvolvedores trabalhando em um mesmo componente ou em componentes relacionados

• Defeitos são descobertos enquanto o trabalho está fresco na cabeça dos envolvidos

• Todo o código no repositório (GIT, CVS...) está testado e existe uma execução de um estado conhecido

• Todo novo código está compilado e testado

• Segurança em relação a mudanças

• O progresso do sistema como um todo pode ser medido

3.7. Reflexão e adaptação constante

Um dos princípios do Manifesto Ágil é "Em intervalos regulares, o time observa reflete sobre como tornar-se mais efetivo, podendo ajustar e otimizar seu comportamento de acordo com as observações."

Essa é a melhor maneira de dar poder ao time, observar e corrigir tudo o que da mínima forma atrapalha ou impede a melhoria do time. Fazer uso constante dessa prática faz com que a cada iteração o time possa produzir mais e melhor.

A reunião de retrospectiva, executada ao final de cada iteração, provê uma das primeiras oportunidades para que o time possa fazer essa reflexão e implementar melhorias em seu processo, seu produto e a relação desses com o objetivo final.

O ciclo rápido e frequente de reflexão e adaptação faz que não só problemas sejam corrigidos como facilita a experimentação de formas diferentes de corrigir um determinado problema. Ao fim dessa iteração analisa-se inclusive se a forma que determinado problema foi atacado foi a melhor e se está sendo eficaz.

No que diz respeito a essa reflexão, não devemos nos limitar apenas a iteração em si, mas também ao ser atingido o final de um ciclo maior, como o fechamento de um *Release Plan*, ou de uma meta anual, ou até da entrega de um sistema como um todo.

É interessante analisar as métricas da iteração e como o time pode fazer para melhorá-las. Alguns exemplos de métricas que podem ser analisadas estão a seguir:

• Número de histórias adicionadas no início de iteração e completadas a seu fim.

• Número de histórias aceitas

• Número de histórias endereçadas para o próximo *Sprint*

• Histórias adicionadas durante a iteração (deveria sempre ser zero)

• Defeitos conhecidos no início da iteração

• Defeitos conhecidos ao fim da iteração (e sua relação com o item anterior)

• Novos casos de teste (preferencialmente automatizados)

• Percentual de cobertura de testes

Outro ponto primordial a ser observado é a qualidade das histórias escritas pelo *Product Owner* e a relação do entendimento delas com o Time. Se estão escritas de forma a dar pleno entendimento do problema que está sendo atacado e o seu objetivo. Em alguns times o próprio *Product Owner* é responsável por escrever o teste de aceitação antes do desenvolvimento para guiar o time no progresso da história em questão.

Além de levantar quais foram os problemas e de que forma eles podem ser endereçados, é muito importante destacar os sucessos da iteração, não somente para que o time tenha a percepção de suas vitórias mas para que continuem fazendo bem (e cada vez melhor) aquilo que já é positivo.

4 Scrum em grandes projetos

Um dos maiores desafios em projetos de grande porte é manter a comunicação na dose certa. De acordo com o *Bull Survey* 57% dos projetos falham por comunicação inadequada entre as partes importantes do projeto. [Bull Survey, 1998]

Quando falamos em grandes projetos, é necessário observar os canais de comunicação dentro de um time grande. Quanto maior o time, mais canais de comunicação são abertos e obviamente aumenta o tempo desprendido para alinhar a comunicação entre os membros.

A fórmula abaixo, exemplifica de uma maneira simplista e não exata o aumento dos canais de comunicação visto o aumento das pessoas diretamente envolvidas no projeto.

Communication channels = N*(N-1)/2, da ordem de n**2.

Havendo 3 pessoas, 3 canais de comunicação. 5 pessoas, 10 canais, 7 pessoas 21...

Pode-se concluir que um dos fatores decisivos no sucesso de um projeto é o tamanho do time, diretamente conectado à quantidade de canais de comunicação. Ao iniciar um novo projeto usando Scrum, uma das primeiras decisões é como organizar indivíduos em times, considerando seu tamanho e as habilidades necessárias para execução do projeto.

Alguns problemas comuns de comunicação que ocorrem em projetos são: não entendimento completo dos requisitos, decisões incorretas de arquitetura não comunicada entre os envolvidos, expectativa gerencial não claramente comunicada, falta de transparência na comunicação de riscos, etc.

4.1. Aplicação em times pequenos

O tamanho ideal de um time de Scrum é aproximadamente sete pessoas (7 ± 2), [Schwaber, K. 2007]. Com essa quantidade de pessoas, normalmente é possível agregar todas as especialidades necessárias em um projeto de desenvolvimento de software.

Times maiores do que nove pessoas têm sua produtividade reduzida e os mecanismos de controle propostos pelo Scrum tornam-se pesados. Liderar e manter um *Daily Meeting* efetivo e dentro de um tempo determinado torna-se difícil para o *Scrum Master*.

É altamente recomendado quebrar em dois ou mais times pequenos caso se tenha mais de nove pessoas disponíveis. Recomenda-se compor um time, com as habilidades necessárias para resolver o problema proposto, e selecionar um *Sprint Backlog* que esse time reduzido consiga se comprometer. O mesmo processo deverá ser feito para o outro time resultante da divisão, com o restante do *Backlog* que o primeiro time não pode se comprometer.

Existem outras vantagens na manutenção de times pequenos, tais como:

Gerenciamento dos pares: com um time pequeno, o próprio time observa facilmente o comportamento e a codificação produzida pelos seus pares, sendo claro identificar pontos de melhoria no dia-a-dia, pois todos sabem o que cada um está fazendo, e podem ajudar caso observem que alguma atividade não esta sendo realizada como deveria.

Relação mais próxima: A proximidade das pessoas facilita a construção de confiança mutua e interdependência. "*The Five persistent feelings of Superior Work Teams: inclusion, commitment, loyalty, pride and trust.*" Os cinco sentimentos persistentes num time de trabalho superior: inclusão, compromisso, lealdade, orgulho e confiança [Kinlaw, D. C. 1990].

Especializações ficam mais visíveis: um problema comum dos times de desenvolvimento é quando somente um desenvolvedor detém o conhecimento de uma parte do sistema. Isso faz com que o time torne-se dependente do conhecimento desse desenvolvedor, uma possível ausência como férias, doença ou um eventual atraso em um dia comum de trabalho, faria com que o time não pudesse prosseguir em um eventual ajuste dessa parte do sistema. Com o time pequeno, essa especialização torna-se visível, durante as reuniões de planejamento e inclusive durante os *Daily Meetings*. Nesse caso o próprio time consegue visualizar o problema e diluir esse conhecimento.

A comunicação é mais quente: Como num time pequeno as pessoas têm maior facilidade para se reunir e se comunicar face a face, a efetividade da comunicação é muito maior, pois se dispensam artifícios "frios", tais como comunicação documental.

Como pode ser visto no gráfico abaixo (*Richness of Communication Channel*), quanto maior a impessoalidade na comunicação, menor a efetividade dessa comunicação. O gráfico baseia-se na teoria da riqueza da mídia, que analisa um meio de comunicação por sua capacidade de reproduzir as informações enviadas sobre ele. Um exemplo disso é uma informação enviada por email, mesmo que seja redigido de forma clara ele exclui tom de voz, gestos e expressões faciais que eventualmente poderiam trazer mais informações facilitando o entendimento do que está sendo dito [Daft; Lengel, 1984].

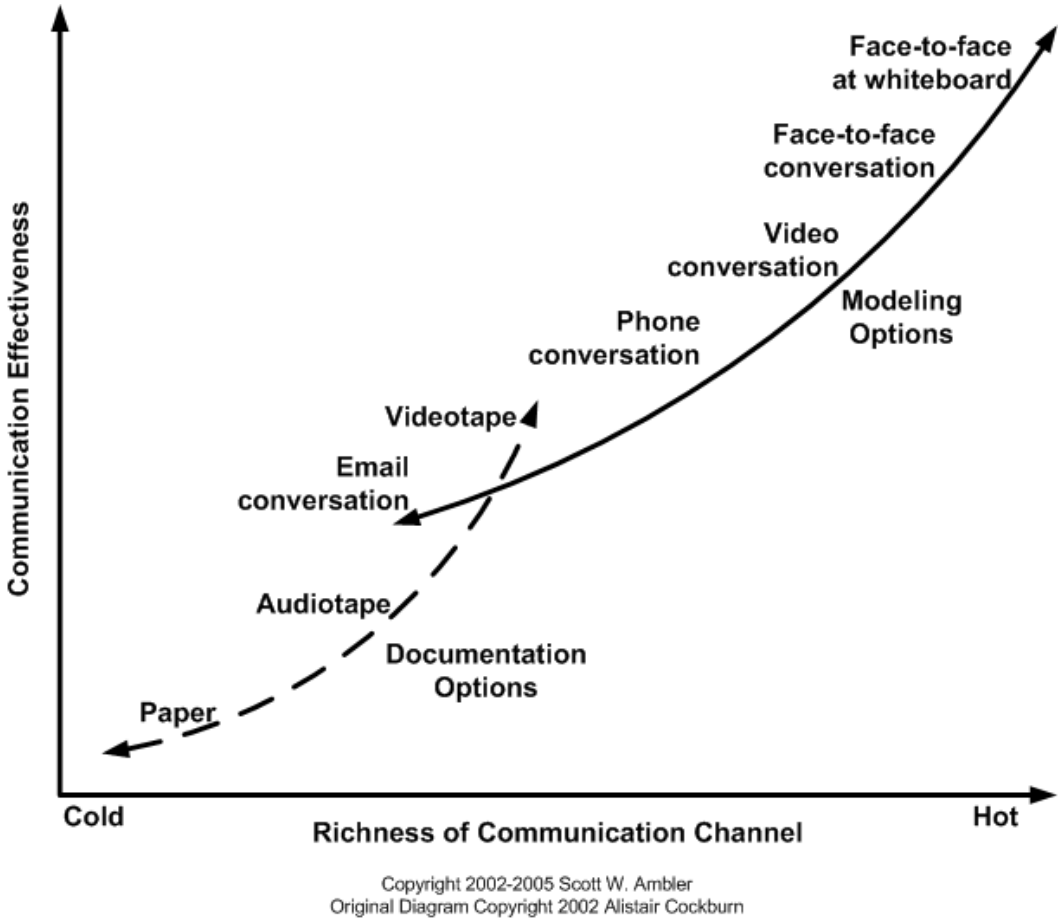

Figura 11 – Riqueza dos canais de comunicação

[Ambler, 2005 / Cockburn, 2002]

4.2. Aplicação em times grandes

No seu livro "*The Enterprise and Scrum*", Ken Schwaber [Schwaber, 2007] descreve uma tentativa de criar um time de 17 pessoas enquanto implementava Scrum em uma determinada empresa.

No primeiro momento ele achou que poderia funcionar, que o time iria se organizar de forma a trabalhar com esse tamanho, mas não aconteceu assim. Durante as reuniões, era difícil escutar as pessoas. O número de pessoas era grande, logo, era muito fácil surgirem conversas paralelas. Muitas coisas eram

ditas e muitas idéias surgiam, e era impossível manter todo o time focado no que estava acontecendo naquele momento.

Instintivamente o time começou a se dividir em pequenos grupos, e depois de algum tempo acabou se dividindo em dois times completos, cada um com seu PO e SM. Após a divisão, os times criaram representantes que se reuniam diariamente após os seus devidos *Daily Meetings* para a uma reunião conhecida como *Scrum of Scrums* [Schwaber, 2007].

Em contrapartida Martin Fowler em seu artigo "*The New Methodology*" [Fowler, 2005], comenta sucesso com times de 100 pessoas distribuídas em vários continentes quando exemplifica a possibilidade de uso de metodologias ágeis em projetos grandes, ressaltando que para que possa ser realizado, a empresa e os times já devem ser proficientes nessas metodologias.

4.3. Casos de empresas conhecidas

4.3.1. Caso SirsiDynix e StarSoft Development Labs

Jeff Sutherland, Peter Vaihasky e Anton Victorov descrevem em seu artigo "*Hyperproductivity In Large Projects Though Distributed Scrum*" a experiência no projeto complexo chamado *Horizon 8.0* que pode ser comparado com um sistema de gestão empresarial contendo um portal público acessado por mais de 200 mil pessoas, em uma pareceria da *SirsiDynix* e *StarSoft Development Labs* que continha 56 desenvolvedores, distribuídos entre Estados Unidos, Canadá e Rússia [Suthelrand; Vaihasky; Victorov, 2006].

Nesse caso além de termos a aplicação de Scrum em um time grande, podemos observar outra dificuldade no uso naturalmente conhecido de Scrum, os times estão distribuídos em três países distintos.

Uma das práticas destacadas no artigo foi uma maneira dos times das diversas regiões sincronizarem uma vez por dia através de uma reunião com representantes dos times (*Scrum of Scrums*).

Qualquer desenvolvedor de qualquer time em qualquer uma das localidades poderia desenvolver qualquer tarefa designada para o projeto, mapeadas através de uma ferramenta de controle de tarefas, independente de qual local fisicamente esse time estivesse.

Algumas práticas utilizadas nesse projeto foram:

Daily Meeting para todos os desenvolvedores de um mesmo time, considerando a diversidade de localidades.

• Reuniões diárias dos *Product Owners*.
• *Builds* automatizados em um repositório central a cada hora.
• A não distinção entre desenvolvedores (por país ou por time).
• Algumas práticas de XP como programação em pares e *refactoring* agressivo.

O projeto utilizou uma ferramenta única de acompanhamento de tarefas, isso deu a todos os envolvidos no projeto uma visão em tempo real do estado de cada *Sprint* além de dar relatórios gerenciais de status.

Uma das maiores dificuldades encontradas durante o projeto era a sincronia entre times, já que existia uma diferença de dez horas entre St. Petersburg e Utah. Para solucionar esse problema o time enviava por email tudo o que havia sido discutido no seu *Daily Meeting* e os demais times liam antes que o seu fosse iniciado.

Além das notificações por email, o time Utah passou a se reunir mais cedo, as 7:45h, 17:45h no horário de St. Petersburg, dessa forma eles podiam fazer teleconferências para tirar eventuais dúvidas.

Além do *Scrum of Scrums* diários dos desenvolvedores, *Scrum Masters*, *Product Owners* e Arquitetos, faziam toda segunda-feira pela manhã o seu *Scrum of Scrums* para garantir a coordenação dos requisitos e performance consistente entre os times.

No inicio do projeto a SirsiDynix trouxe um dos melhores seniores para St. Petersburg para passar uma semana treinando seus engenheiros em algumas peculiaridades do projeto. Depois disso fizeram o mesmo com um engenheiro de Utah, após algum tempo, alguns desenvolvedores chave de St. Petersburg vieram para os escritórios da SirsiDynix para treiná-los.

Ter esses desenvolvedores circulando entre US e St. Petersburg ajudou a atingir dois objetivos. Compartilhar conhecimento e criar união entre esses grupos. A partir disso, trabalhar em componentes ou tarefas inter-relacionadas se tornou muito mais fácil.

4.3.2. Caso Boeing 777

Nesse caso em particular não foram utilizadas as metodologias abordadas na dissertação até o momento, mas o caso mostra algumas práticas aonde podemos fazer paralelo com práticas ágeis conhecidas.

Diante de uma crise no setor de aviação, agravada pela grande competição entre os fabricantes de aviões, a Boeing se viu na necessidade de criar um novo avião para suprir a demanda da aviação comercial. Um avião com maior capacidade de passageiros e carga do que os existentes com o consumo inferior do que as aeronaves de grande porte existentes. Nenhuma fabricante possuía o exato avião necessário pelas empresas de aviação e para se manter a frente de seus concorrentes a Boeing precisava aproveitar essa oportunidade, e assim se manter a frente por mais algumas décadas [BPS/BBC, 1996].

United Airlines, uma das empresas de aviação comercial parceira da Boeing de longa data tinha em suas mãos propostas de construção de diversas fábricas distintas, entre elas a Boeing e a Airbus. A escolha direcionaria a linha de aeronaves que a empresa trabalharia pelos próximos trinta anos

De uma forma inédita na história da United Airlines e a Boeing, decide-se construir em conjunto um avião plenamente funcional de forma a atender não só as necessidades e expectativas do cliente mas dos passageiros, tripulação, mecânicos e todos envolvidos na utilização da aeronave.

Com isso a United Airlines se comprometia investir 3.5 bilhões de dólares em um avião que ainda não existia. Comprando antecipadamente 34 aviões e a opção de comprar outros 34 após três anos.

Em 1992 a Boeing começou a agrupar e treinar um grupo de 10 mil funcionários para o desenvolvimento do 777, sob uma nova filosofia aonde se encorajava e até forçava a colaboração entre as pessoas.

A mudança cultural era guiada por Alan Mulally, vice presidente executivo da Boeing na época, hoje CEO da Ford, que não somente trouxe uma nova filosofia para o projeto do 777 mas também uma mudança na maneira de organizar as 10 mil pessoas envolvidas no projeto. De alguma forma eles se remetiam à maneira que trabalhavam no passado, nas primeiras construções de aeronaves, onde a produção e engenharia ficavam muito próximos, em um mesmo prédio, divididos apenas por um lance de escadas. Qualquer problema que surgisse na produção imediatamente consultavam um engenheiro que eventualmente modificava o projeto para resolver o problema, o mesmo acontecia caso um engenheiro descobrisse uma falha no projeto, no mesmo momento interrompia a produção para que o projeto fosse ajustado [Sabbagh, 1996].

Com o crescimento da empresa esse processo se tornou rígido, criando silos entre as áreas e perdendo a comunicação, com isso criou-se a filosofia do “eles” e “nós” reforçando a separação entre as áreas.

Alan Mulay criou então o que foi chamado "*Design Build Teams*" times multifuncionais responsáveis por componentes específicos da aeronave, contendo todas as pessoas necessárias para definir e construir o componente em questão. Pessoas que nunca haviam trabalhado em conjunto. A comunicação passou a ser mais aberta e transparente, franca, deixando inclusive que alguns conflitos que, eventualmente se tornavam, passarem a ser visíveis e dessa forma poderiam ser resolvidos mais rápido e de forma efetiva [Lewis, 2006].

Essas formações de times e a maneira de lidar com sua comunicação interna, faz um paralelo com a sugestão de formação dos times em projetos ágeis e seu comportamento.

A partir disso a Boeing passa nesse projeto a ter engenheiros e mecânicos trabalhando em conjunto, evitando diversas falhas comuns de comunicação

gerando uma drástica redução dos erros por falha de comunicação entre essas partes.

Alem da formação das equipes similar aos apresentados em processos ágeis, outras práticas similares foram adotadas como as listadas abaixo:

Review

Antes mesmo de construir as partes mecânicas da aeronave, foi criado um protótipo em tamanho real do interior e exterior da aeronave, para que os investidores pudessem ver a diferença do espaço interno proposto para o 777 em comparação aos seus concorrentes.

Protótipo testável

Durante o design da porta da aeronave, surgiu uma dúvida a respeito do seu comportamento diante de uma nevasca, ou em baixíssima temperatura. A porta deveria abrir, considerando o esforço de uma pessoa comum mesmo com uma camada de gelo na porta e seu entorno. Além do baixo esforço para abrir a porta, deveria ser considerada a possibilidade do rompimento de seus componentes diante de fadiga do uso.

Integração contínua

Um dos problemas comuns nos projetos anteriores da Boeing era que os projetos bidimensionais não consideravam perfeitamente o espaço necessário para os componentes, surgindo situações aonde, no momento da primeira montagem, surgiam conflitos entre algumas partes do avião. Era difícil, por exemplo, observar que a altura de um determinado duto de ar condicionado conflitaria com o espaço necessário para ele na fuselagem.

Mesmo considerando os protótipos o processo tornava-se custoso e demorado, e para reduzir esse custo construíram um software aonde poderiam validar o novo componente do projeto contra o projeto já adicionado nesse sistema, muito similar ao que temos nos projetos ágeis como a Integração Contínua. Essa validação poderia chegar ao nível da cabeça de um parafuso conflitando com o espaço necessário para seu encaixe.

4.3.3. Caso Globo.com

4.3.3.1. Histórico

Desde seu surgimento, a Globo.com utilizou uma conhecida ferramenta de mercado para a administração e publicação de suas matérias e páginas editorias de seus diversos produtos. Essa ferramenta permitia customizações de layout e adição de novas funcionalidades para o internauta, quanto à administração e edição por partes dos jornalistas, editores e desenvolvedores. Além de um custo alto de manutenção desenvolvimento para aplicar essas customizações, a ferramenta não trazia toda a flexibilidade e facilidade de uso desejadas aos nossos clientes. Aumentar a facilidade de uso para os editores é fundamental quando existe a necessidade de acelerar a publicação de uma notícia. Reduzir o custo e aumentar a flexibilidade de customização e facilidade criação de novas funcionalidades pelos desenvolvedores é fundamental para que cada novidade seja desenvolvida com velocidade.

Diante desse cenário surgiu a necessidade de descontinuar a ferramenta antiga de publicação. Mas qual ferramenta teria a capacidade de suprir as necessidades atuais além da evolução desejada?

A partir desses questionamentos, foi alocado um time, composto por um *Product Owner*, um *Scrum Master* e cinco desenvolvedores para estudar ferramentas de mercado para esse fim, que atendesse a todas essas necessidades, esse time ficou conhecido como Core. Seu *Product Owner* tinha conhecimento prévio de manutenção e customização da antiga ferramenta, e em conjunto com o time selecionou uma série de outras ferramentas de publicação para um estudo comparativo.

Nesse estudo foram encontradas diversas ferramentas que atendiam parcialmente as necessidades, mas nenhuma ferramenta no mercado naquele momento as atendia por completo. O time então decide unir as boas ideias de cada

ferramenta estudada e criar uma ferramenta interna, contendo a cobertura de todas as necessidades e desejos.

Após alguns estudos e algumas provas de conceito, iniciaram o desenvolvimento dessa nova ferramenta. Mas a entrega não estava de acordo com a velocidade esperada pela diretoria, após alguns meses, surge a necessidade de acelerar o desenvolvimento da ferramenta, nesse momento outros cinco times extraídos de outras áreas de desenvolvimento são movidos de seus projetos atuais e alocados no projeto da nova ferramenta de publicação.

Os times não estavam familiarizados com a tecnologia e com o projeto alguns eram novos em processos ágeis e não tinham habilidades firmadas em testes automatizados, programação em par e outras práticas ágeis.

4.3.3.2. Cenário 1: Cinco times pequenos, trabalhando em paralelo

Nesse cenário, foram adicionados cinco novos times. Juntos totalizavam 49 pessoas: 28 desenvolvedores, 4 designers, 4 arquitetos da informação, 3 especialistas em infra estrutura/bando de dados, 5 *Product Owners* e 5 *Scrum Masters*.

Durante esse cenário não eram feitos *release plans* e no momento inicial não havia um CI centralizado, mas era possível executar localmente os testes feitos previamente pelo primeiro time alocado ao projeto.

A partir da entrada desses novos times ao projeto, decidiu-se diluir os membros do time Core nos demais times, adicionando um de seus membros a cada um dos novos times de forma que pudessem agregar conhecimento do projeto a cada um dos novos times.

A primeira ação tomada foi reunir todos os times e mostrar o propósito e detalhes do projeto, de forma que todos tivessem pleno conhecimento daquilo que estavam desenvolvendo e sua importância.

Na sequência, os Sprints de todos os times foram sincronizados, para que o planejamento e revisão pudessem seguir em paralelo.

Para que a medida de pontos de complexidade (considerando o *Planning Poker*) fosse equalizada entre os times, foram feitas sessões de estimativas em conjunto, usando uma mistura de uma prática exemplificada por Ken Schwaber em seu livro *The Enterprise and Scrum* [Schwaber, 2007] e por Mike Cohn em seu artigo *Establishing a Common Baseline for Story Points* [Cohn, 2008] :

Em um auditório os cinco times foram agrupados de forma a ocuparem uma mesa cada um. Com o intuito de definirmos a história base para as demais estimativas, um *Scrum Master* mediador leu algumas histórias e todos os membros de todos os times levantando as mãos elegeram a que seria a mais simples de todas.

Na sequência uma única história era lida com seus casos de aceite, em um intervalo de tempo pré-estabelecido cada mesa discute entre si e decide um valor de complexidade.

No segundo momento cada mesa descreve soluções e dificuldades para aquela história e a complexidade estimada por seu grupo.

Após ouvir cada um dos grupos, cada mesa faz uma nova rodada de estimativa para a mesma história. O processo é repetido até que os times encontrem uma estimativa alinhada para aquela história.

Considerando que as estimativas de cada história são relacionadas, ou seja, se uma história tem 5 pontos de complexidade outra tem 8, a segunda história é quase duas vezes mais complexa que a primeira.

Após algumas rodadas de equalização, os times se sentem confiantes de que uma estimativa feita por outro é coerente com o que seu próprio time estimaria.

Em uma segunda fase, apenas para confirmação da afirmativa anterior, cada time estima uma história diferente, escolhida aleatoriamente a partir do backlog, e apenas confirma a sua estimativa com os demais times, se tudo ocorreu como esperado a estimativa está coerente com que cada time estimaria.

A partir desse momento todos os times passaram a ter o *mesmo Product Backlog*, mas *Sprint Plannings* e *Sprint Backlogs* distintos a serem desenvolvidos

em um mesmo tamanho de *Sprint*, uma vez que todos os *Sprints* estavam sincronizados.

Os *Daily Meetings* de cada time ocorriam no mesmo horário, cada time focado nas histórias de seu próprio quadro. Na sequência do *Daily Meeting* um representante do time se reunia com os demais representantes dos times no *Scrum of Scrums* para falar a respeito da história que o time estava trabalhando naquele dia e suas dificuldades, eventuais integrações ou generalizações de soluções que poderiam ser aproveitadas pelos demais times. No retorno dessa reunião o membro do time que havia participado comunica ao time qualquer ajuste da história, além de comunicá-los a respeito do que os demais times estão produzindo.

Uma vez por semana os especialistas de cada time (Arquitetos da informação, Desenvolvedores *Client-side*, Designers) se reuniam em um *Scrum of Scrums* por especialidade técnica, para discutir decisões aplicadas nos respectivos times e para manter seus padrões.

Além do *Scrum of Scrums* e as reuniões semanais por especialidade técnica, havia também uma reunião diária entre os POs de cada time, aonde discutiam as histórias que sofriam alguma alteração durante o *Sprint*, criavam novas histórias e seus casos de aceite além de planejar quais histórias eventualmente seriam separadas para os próximos *Sprints*, observando as dependências entre times.

Antes da conclusão do *Sprint*, os membros do time Core que estavam diluídos nos demais times se agrupavam para fazer uma inspeção no código e integrar as histórias de cada time que já havia concluído sua funcionalidade. Esse se tornava um momento critico para os times que ainda não haviam concluído sua funcionalidade, porque passavam a não contar com aquele que mais conhecia o projeto, membro do time original do Core.

Outro problema era a integração de cada funcionalidade ao repositório. Como cada time trabalhava em uma ramificação do repositório (*branch*), era necessário, após a inspeção do código, a integração no repositório principal, o que dificultava o próximo time da fila, uma vez que passava a ser necessário atualizar

sua ramificação antes que o time Core pudesse inspecionar seu código. Esse processo se tornava mais doloroso para os times que demoravam mais para concluir suas histórias, uma vez que cada aprovação dos times da "fila" fazia com que fosse necessária uma nova atualização de seu código, e verificação de eventuais quebras.

Ao termino do *Sprint* todos os times se reúnem novamente em um auditório para um *Review* coletivo, aonde cada time em formato de apresentação de auditório, exibe para os outros times, POs e clientes tudo o que foi desenvolvido durante esse período.

Como a cultura de testes não era difundida dentre os membros de todos os times, alguns limitadores foram estipulados no início do projeto como: Percentual mínimo de cobertura de testes e automatização de casos de aceite criados pelos POs.

Uma forma de dar visibilidade as possíveis quebras de código, foi a adição de um servidor de integração contínua (CI) que executava todos os testes automatizados do sistema a cada nova entrada de código no repositório.

O principal problema nesse cenário foi a competição entre os times. Mesmo que a colaboração fosse explicitamente estimulada, a necessidade de finalizar primeiro a funcionalidade para diminuir problemas com a integração acabou por gerar correria, bugs, testes ineficientes e falta de colaboração.

Outro grande problema era a diferença de conhecimento entre os times e o foco no conhecimento do membro do time Core alocado em cada um desses times.

Envolvidos no projeto: 49 pessoas

Duração do Cenário: 6 sprints de duas semanas.

Fatores positivos

- Visão alinhada do propósito do projeto

- Construção de uma visão única de medida de complexidade
- *Product Backlog* único
- Adição de práticas ágeis no dia-a-dia

Fatores negativos

- Muitas pessoas novas adicionadas ao projeto de uma única vez
- *Scrum of Scrums* ineficiente
- Time Core diluído sem considerar multidisciplinaridade
- Pouco conhecimento comum das tecnologias aplicadas ao projeto
- Competição entre os times

4.3.3.3. Cenário 2: Um único time, grande, composto por membros de todos os times

Como observado no caso anterior, havia uma diferença de conhecimento entre os times, principalmente focado no membro do time Core alocada no time, decidimos então juntar todos os times em um único e grande time, aonde o planejamento era feito em conjunto.

Esse cenário resolveu alguns problemas, como a integração entre desenvolvedores e distribuição do conhecimento técnico e de negócio. Uma vez que todos trabalhavam juntos perdeu-se a divisão em silos dos times originais.

Outro problema que foi diluído do cenário anterior foi a revisão e integração do código. A revisão passou a ser feita durante o desenvolvimento e as integrações eram feitas diretamente no ramo principal do repositório. Caso alguma nova entrada quebrasse os testes executados pelo servidor de integração contínua o código poderia ser revertido, sendo removido do repositório principal até seu ajuste.

Para evitar que novas funcionalidades fossem para algum ambiente antes de finalizadas, existia um sistema de permissões que controlava o acesso a qualquer

funcionalidade, logo uma funcionalidade ainda não finalizada ficaria escondida dos usuários.

Foi implantado um servidor de integração único para o projeto. Ao invés de cada desenvolvedor se preocupar somente com a sua funcionalidade, precisava, a partir de agora, preocupar-se com a integração desta no projeto como um todo. Caso sua nova funcionalidade demandasse alguma modificação estrutural, imediatamente os demais desenvolvedores saberiam da mudança e poderiam discutir a necessidade e adaptar seu código.

Mesmo tendo resolvido alguns problemas, o cenário de um time único e grande gerou alguns efeitos colaterais. A reunião de planejamento de *Sprint* ficou grande, durante o detalhamento de tarefas muitas pessoas perdiam o interesse, deixando de participar das discussões de problema. O *Daily Meeting* ficou longo, desfocado, era difícil saber o que cada um estava fazendo, e os pares não se cobravam. Facilmente um ou outro desenvolvedor perdia o *Daily Meeting* e acabava refazendo alguma tarefa que outro desenvolvedor já tinha iniciado.

Outro problema que passou a existir foi na formação de pares. Alguns desenvolvedores com mais conhecimento acabavam fazendo dupla com outros de conhecimento semelhante, o que acabava deixando desenvolvedores com menos conhecimento desenvolvendo juntos, esse segundo grupo acabava gerando código de menos qualidade.

Envolvidos no projeto: 49 pessoas

Duração do Cenário: 2 sprints de duas semanas.

Fatores positivos

- Planejamento único
- Difusão de conhecimento
- Quebra da competição
- Revisão contínua
- Servidor único de integração contínua

Fatores negativos

- Planejamento muito demorado
- Detalhamento de tarefas ficou pobre
- Dispersão nos *Daily Meetings*
- Formação de pares fixos

4.3.3.4. Cenário 3: Um time grande subdividido em frentes de trabalho

Ainda utilizando o conceito de um grande time para resolver alguns dos problemas gerados no cenário anterior mantendo a soluções que essa iniciativa trazia, misturando o cenário de time grande com o de times pequenos, essa modificação foi discutida com os times antes do início do *Sprint Planning* e implantada na sequência.

A reunião de planejamento continuava sendo feita com todos os desenvolvedores reunidos como um único time, aonde em consenso destacavam cinco ou seis problemas ou temas a serem trabalhados naquele *Sprint*. A partir disso, selecionávamos as histórias do *Backlog* que faziam sentido para resolver cada problema em questão. Uma vez os temas levantados, e as histórias agrupadas por esses temas, os desenvolvedores se ofereciam para compor um grupo responsável por resolver aquele problema. Os grupos deveriam ser mistos, agregando suas habilidades para resolver o problema em questão. Um grupo não poderia conter muito mais membros que outros, para prevenir que um problema tivesse mais atenção que outro problema levantado. Um grupo poderia "solicitar" a presença de um desenvolvedor que de acordo com eles tivesse mais conhecimento sobre um determinado assunto ou tecnologia a ser aplicada em uma das histórias. Essa formação conseguiu diluir definitivamente a questão da integração entre desenvolvedores e conhecimento técnico que já haviam obtido razoável melhoria no cenário anterior.

Como eram times dinâmicos, construídos no momento do planejamento, o grupo durante o *Sprint* poderia ceder um de seus membros caso existisse necessidade do seu conhecimento em outro time e, obviamente, caso isso não impactasse no seu desenvolvimento. O tamanho em dias do *Sprint* mantinha-se o mesmo. Caso um dos grupos concluísse suas histórias antes do previsto, poderia puxar outra história ou diluírem-se nos demais times para ajudá-los a concluir sua história.

Dessa forma, o *Planning* II tornou-se mais focado, pois o grupo responsável por resolver o problema estava comprometido em resolvê-lo e entendia do assunto. Perde-se a necessidade do *Scrum of Scrums* para que um time saiba o que o outro está desenvolvendo porque esse conhecimento foi adquirido durante o *Planning* I feito em conjunto. Outro fator que colaborou com a integração foi a possibilidade de um desenvolvedor se desagrupar para ajudar o outro time, automaticamente os desenvolvedores durante o dia conversavam entre si, principalmente entre seus times originais, para saber se poderiam ajudar de alguma forma. Nota-se nesse momento um destaque dos desenvolvedores mais proficientes, acabam passando por diversos temas no decorrer do *Sprint*.

Durante esse cenário surge o time de suporte. Composto pelos desenvolvedores que se destacaram no decorrer do projeto e parte dos desenvolvedores do time Core. Esse time passa a dar suporte para os demais times, dado a sua facilidade de transitar entre os times e o conhecimento das tecnologias empregadas no projeto. As principais demandas desse time eram levantadas pelos demais times durante seus planejamentos ou em tempo de desenvolvimento. A previsibilidade do planejamento não era clara, poderiam surgir demandas no decorrer do *Sprint* dos demais times. Dada essas circunstâncias esse time deixa de adotar Scrum para utilizar Kanban. O quadro Kanban era alimentado pelos POs dos demais times em conjunto com o PO do time Core, que moderava em conjunto com os demais POs a priorização de entrada na fila de desenvolvimento.

Ainda nesse cenário outros três times da empresa contendo 4 desenvolvedores, um PO e um SM cada, foram inseridos ao projeto, com intuito de absorver conhecimento para prepará-los para a próxima fase, a divisão dos times por áreas de negócio.

Envolvidos no projeto: 67 pessoas

Duração do Cenário: 10 sprints de duas semanas.

Fatores positivos

- *Planning* em conjunto
- Frentes de trabalho dinâmicas
- Maior integração e colaboração entre desenvolvedores
- Todos sabiam o que cada grupo estava desenvolvendo

Fatores negativos

- Ainda restava diferença de conhecimento entre as frentes de trabalho gerando não uniformidade de código.
- Necessidade de surgimento de um time de suporte

4.3.3.5. Cenário 4: times por área de negócio.

A partir desse cenário, os times são separados respeitando as áreas de negócio da empresa, no que diz respeito a produtos de publicação, Esporte, Jornalismo e Entretenimento e Publicação. Cada time torna-se independente, seguindo com o planejamento de sua área, sem a participação dos demais times, exceto em mudanças estruturais quando necessitam de alguma demanda da base do sistema, caso em que torna-se necessário o envolvimento do time Core.

Representantes de área são eleitos pelos desenvolvedores e passam a ser incluídos em discussões sobre arquitetura e tudo aquilo que de certa forma poderia impactar o seu produto.

Mantém-se o time Core como o desenvolvimento da base da ferramenta de publicação e mantenedor de seus padrões. O time é responsável não somente pelo suporte aos demais times como também por manter a plataforma base de desenvolvimento, enquanto os demais times constroem seus projetos utilizando como framework o projeto original, esse time fica também responsável pela correção de bugs e atendimento a demandas dos times.

O projeto base, chamado publicação-core passa a ser distribuído em *releases* planejadas pelo time Core e acompanhadas pelos demais times. A cada iteração do time Core ou a cada ajuste na base uma nova *release* é lançada e para que os times não fiquem muito distantes da release mais recente é criada uma política de atualização em comum acordo com as áreas. A política de atualização passa a servir não somente para manter os produtos atualizados bem como para manter o suporte em temas mais recentes.

Fatores positivos

- Um time focado na ferramenta de publicação
- Times focados na construção de produtos utilizando a ferramenta de publicação como base

Fatores negativos

- Foco na evolução do sistema deixando débitos técnicos em segundo plano

4.3.3.6. Cenário 5: Time de engenharia e time de suporte

Nesse cenário, os times continuam distribuídos por área, as modificações indicadas passam a ser feitas dentro dos times que continuam trabalhando na manutenção e evolução do sistema de base para os times das demais áreas.

Observando que a demanda de suporte e evolução do sistema em diversos momentos se tornava conflitante, decidiu-se dividir o time Core em dois times, com focos distintos.

O primeiro, time de Engenharia passa a trabalhar na reestruturação da arquitetura de componentes, visto que um dos efeitos colaterais dos primeiros cenários foi a não uniformidade de codificação e arquitetura. Esse time passa a refatorar alguns componentes com o intuito de melhorar o desempenho do sistema e a velocidade de desenvolvimento dos demais times.

Uma de suas primeiras ações foi reunir representantes dos diversos times e levantar com esses quais os principais gargalos de desenvolvimento, tudo aquilo que demandava tempo e poderia ser ajustado de alguma forma para poupar tempo de desenvolvimento e tornar a codificação mais fácil.

Dada essa participação dos times, foram levantadas histórias e traçada uma meta de melhoria de performance de desenvolvimento dos demais times, que deveria ser atingida uma vez que tais histórias fossem implementadas.

O segundo, time de Suporte tem como premissa ajudar na questão do aprendizado do uso da plataforma, tanto para desenvolvedores quanto para editores. Como o time de Engenharia a partir desse cenário passa a criar novas funcionalidades para reduzir o tempo de desenvolvimento, o time de Suporte passa a ser o primeiro time a integrar suas novas releases, não somente para validar a entrega quanto para adquirir conhecimento e repassar para os demais times.

É criado um produto, nos formatos dos produtos de publicação para facilitar no suporte, com a experimentação dessas novas funcionalidades além de matérias voltadas para o desenvolvedor e editor, contendo um conteúdo rico e visual ensinando, por exemplo, como iniciar o desenvolvimento nessa plataforma, ou como desenvolver um determinado tipo de componente. Além de vídeos explicativos, gerados também pela própria equipe de suporte.

Fatores positivos

- Um time focado na evolução da ferramenta
- Um time focado na reestruturação da arquitetura e melhoria de performance

Fatores negativos

- Necessidade do constante alinhamento entre os dois times.

4.3.3.7. Lições Aprendidas

Antes de pensar em escalar o uso de Scrum é necessário analisar se a equipe está preparada para utilizar uma metodologia ágil.

"Métodos ágeis diferem substancialmente de métodos tradicionais. Ágil não é uma correção ou ajuste daquilo que vínhamos fazendo. É uma mudança radical na pratica e cultura." [Leffingwell, 2007].

É fundamental que os *Product Owners*, *Scrum Masters* e principalmente o Time sejam treinados em seus papéis para que exista um fluxo contínuo de desenvolvimento.

Um dos fatores que facilitou a comunicação entre os membros do time na Globo.com, principalmente no que diz respeito à programação em par foi a mudança do mobiliário. Inicialmente utilizavam-se baias fechadas, aonde cada desenvolvedor tinha uma área "protegida" dos demais membros da mesma equipe, muda-se para um único mesão por equipe, nesse caso os desenvolvedores passam a ficar lado a lado com seus pares, facilitando a comunicação verbal e principalmente a visual. Os equipamentos também foram substituídos, de desktops para notebooks, aumentando a flexibilidade de movimentação nos mesões.

Outro fator que deve ser levado em consideração antes de adicionar novos times ao projeto é a preparação e estruturação do código fonte para receber os

novos componentes [Schwaber, 2004], além da preparação dos desenvolvedores nas tecnologias, ferramentas e padrões e relacionados a ele.

5 Conclusão

No início da dissertação observamos a afirmativa *"Scrum is the first process with well-documented linear scalability. When you double team size in a well-implemented Scrum, you can double software output, even when the teams are distributed and outsourced".* [Vaihansky, P. / Sutherland, J. / Victorov, A. 2006]. Se destacarmos o *"... in a well-implemented Scrum ..."* podemos dizer que times maduros na metodologia Scrum poderiam ser incorporados ao projeto, efetuando trabalho paralelo aos times já alocados. Isso é confirmado no artigo *"The New Methodology"* [Fowler, 2005] que destaca a necessidade da proficiência dos envolvidos no projeto na metodologia aplicada. Isso faz com que a atenção dos times seja mais focada no produto e na comunicação entre os times, do que no processo em si.

Além do conhecimento na metodologia empregada ser de completo domínio e uso frequente das pessoas envolvidas no projeto, é extremamente importante que as tecnologias utilizadas também sejam dominadas pelos novos times, de forma que facilite a normalização dos componentes, e o eventual intercambio entre membros de times.

"Prior to scaling any project, an appropriate infrastructure must be put in place. For instance, if a project will employ multiple collocated teams, a mechanism for frequently synchronizing their work must be devised and implemented. Also, a more detailed product and technical architecture must be constructed so that the work can be cleanly divided among the teams. If these teams are to be geographically distributed, high-bandwidth technology for source code sharing, synchronized builds, and alternative communications such as instant messaging must be employed." [Schwaber, 2004]. O texto de Ken Schwaber reforça que, antes de adicionar novos times em qualquer projeto, a estrutura para comportar a adição desses times deve ser organizada. Tanto no aspecto físico, e nesse caso, mesas, quadros, salas, rede, etc. Quanto no aspecto de sistema, ou seja, a arquitetura deverá estar em um estado que seja possível dividir claramente componentes para distribuir o trabalho entre novos times.

Nos casos descritos nessa dissertação podemos observar a aplicação de múltiplos times e o emprego da paralelização do *Backlog* na construção de funcionalidades ou componentes. Um fator crítico observado foi o fluxo de comunicação entre esses times. Uma forma encontrada para facilitar a essa comunicação foi a utilização da prática do *Scrum of Scrums*, tanto no aspecto da sincronia entre os times, quanto da manutenção do *Backlog* por parte dos diversos *Product Owners* envolvidos no projeto. Além do acompanhamento da evolução dos times, é importante que as especialidades (designers, dbas, desenvolvedores) frequentemente se reúnam para discutir e manter os padrões do projeto e compartilhar com os demais as soluções aplicadas dentro do time.

Como se trata de um único projeto é importante que o objetivo seja único e alinhado entre os times. Cada *Goal* de cada *Sprint* de cada time deve estar em conformidade com o objetivo geral do projeto, para que todos observem o valor da contribuição de cada time para atingir o objetivo final.

Promover entregas frequentes e *Review* com a participação dos clientes, e aqueles que de alguma forma serão impactados pela entrega, permite um acompanhamento claro da evolução do projeto e da necessidade de eventuais ajustes no sistema, integrações, ou no planejamento de entregas.

De todas as práticas, a que considero mais importante e que acredito que tenha feito a diferença, é a reflexão e adaptação constante. Tendo essa última pratica sendo aplicada a cada iteração, é possível encontrar novas maneiras de melhorar tanto o processo, quanto o projeto, de forma a entregar mais valor em menos tempo.

6 Referências

[Highsmith, 2001] HIGHSMITH, J. **History: The Agile Manifesto** http://agilemanifesto.org/history.html

[Highsmith, 2001] HIGHSMITH, J. **Principles behind the Agile Manifesto** http://www.agilemanifesto.org/principles.html

[Highsmith, 2001] HIGHSMITH, J. Manifesto for Agile Software Development http://www.agilemanifesto.org/principles.html

[Fowler, 2005] FOWLER, M. **The New Methodology** - http://martinfowler.com/articles/newMethodology.html

[Suthelrand; Vaihasky; Victorov, 2006] SUTHERLAND, J; VAIHANSKY, P; VICTOROV, A. **Hyperproductivity In Large Projects Though Distributed Scrum** http://www.agilejournal.com/articles/columns/column-articles/190-hyperproductivity-in-large-projects-though-distributed-scrum

[Cohn, 2008] COHN, M. **Establishing a Common Baseline for Story** http://blog.mountaingoatsoftware.com/establishing-a-common-baseline-for-story-points

[Cohn, 2007] COHN, M. **Advice on Conducting the Scrum of Scrums Meeting** http://www.mountaingoatsoftware.com/articles/35-advice-on-conducting-the-scrum-of-scrums-meeting

[Kniberg, 2010] KNIBERG, H; SKARIN M. - **Kanban and Scrum - making the most of both**

[Anderson, 2003] ANDERSON, J. D. - **Agile Management for Software Engineering: Applying the Theory of Constraints for Business Results**

[Anderson, 2010] ANDERSON, J. D. - **Kanban: Successful Evolutionary Change for Your Technology Business**

[Kniberg, 2009] KNIGERG, H. - **Kanban kick-start example** - http://www.crisp.se/kanban/kanban-example.pdf

[BPS/BBC, 1996] PSB; BBC. **21st Century Jet - The Building of the 777**

[Sabbagh, 1996] SABBAGH, K. - **Twenty-First-Century Jet: The Making and Marketing of the Boeing 777**

[Lewis, 2006] LEWIS, P. J. - **Working Together: 12 Principles for Achieving Excellence in Managing Projects, Teams, and Organizations**

[Schwaber / Beedle, 2001] SCHWABER, K.; BEEDLE, M. - **Agile Software Development with Scrum**

[Schwaber, 2004] SCHWABER, K. **- Agile Project Management with Scrum**

[Schwaber, 2007] SCHWABER, K. - **The Enterprise and Scrum**

[Mountain Goat, 2005] Mountain Goat Software. – **Scrum Flow** - http://www.mountaingoatsoftware.com/scrum/figures

[Cohn, 2005] COHN, M. **Agile Estimating and Planning**

[Leffingwell, 2007] LEFFINGWELL, D - **Scaling Software Agility: Best Practices for Large Enterprises**

[Brooks, 1975] BROOKS, F - **The Mythical Man-Month**

[Daft; Lengel, 1984] DAFT, R; LENGEL, R - **Information richness: a new approach to managerial behavior and organizational design**

[Ambler, 2005 / Cockburn, 2002] AMBLER, S. - **Disciplined Agile Delivery** / COCKBURN, A - **Agile Software Developement**