Nesta seção você encontra artigos voltados para as práticas e métodos ágeis.

Agile Development: Problemas e Soluções no Scrum

Como lidar com cada um dos ritos e papéis do Scrum

ESTE ARTIGO É DO TIPO MENTORING

SAIBA MAIS: WWW.DEVMEDIA.COM.BR/MENTORING-SAIBAMAIS

Cenário:

Processos de desenvolvimento de software em cascata possuem conhecidos problemas, tais como: entregas tardias com pouco feedback do cliente, excessiva quantidade de documentação, foco demasiado em processos, entre outros. Scrum é uma metodologia que, como um de seus objetivos, enfrenta os problemas citados, proporcionando entregas constantes e iterativas, documentando-se apenas o estritamente necessário, focando no produto e nas pessoas que o desenvolvem, etc. Contudo, o Scrum não é fácil de ser implementado, principalmente em times que estão acostumados com processo em cascata. Esse artigo dá algumas dicas, em cada etapa do framework, para que sua adoção seja feita de forma mais tranquila, evitando-se alguns erros comuns quando se adota o Scrum.

Eduardo Antonio C. Fernandes

eduardofernandes@yahoo.com.br

br.linkedin.com/in/eduardoacfernandes/

Graduou-se em Egenharia de Computação e trabalha com desenvolvimento de software desde então. Atualmente, trabalha no Instituto de Pesquisas Eldorado, atuando como Lider técnico, Aquiteto e desenvolvedor de software, principalmente utilizando Java e Microsoft .NET.

Scrum é um framework para desenvolvimento de produtos. Nesse artigo, o foco será no desenvolvimento de software. É interessante notar a denominação de *Scrum* como framework e não processo. Isso porque, essa metodologia diz o que fazer, mas não como. Por exemplo, as entregas devem ser incrementais e interativas a cada Sprint, provendo real valor ao cliente. Note que não é explicado como se dão essas entregas incrementais, qual o tamanho do incremento por produto, além de não se definir o que é valor para o cliente. Todos esses elementos são deixados em aberto, para que cada *Time* aplique o que achar mais conveniente para esses conceitos.

Scrum vem como resposta aos problemas que processos em cascata geram. Os processos em cascata focam na definição de processos para o sucesso do desenvolvimento de software. Além disso, a entrega do produto é feita após longas fases de análise e desenvolvimento, tendo-se feedback do cliente

meses depois do início do projeto. Além disso, mudanças são encaradas como problemas, pois qualquer modificação no produto ocasiona onerosa análise e alteração em documentações, até que ela seja efetivamente aplicada ao produto. Finalmente uma ampla documentação vem aliada ao software, pois se acredita que isso gere real valor ao produto.

Por outro lado, *Scrum* foca estritamente no produto, sendo associado um processo muito leve a ser seguido. Também, entregas são feitas de forma iterativa e contínua. Isso possibilita que o cliente utilize o produto desde o início do desenvolvimento, provendo feedback constante. Feedback é outro ponto importante: eles são bem vindos e esperados, sendo sempre acomodados no *Sprint* seguinte, caso necessário. Documentação é outro ponto chave: *Scrum* não exige documentação e sim, novamente, foco no produto e por isso deve-se produzir o mínimo necessário de artefatos.

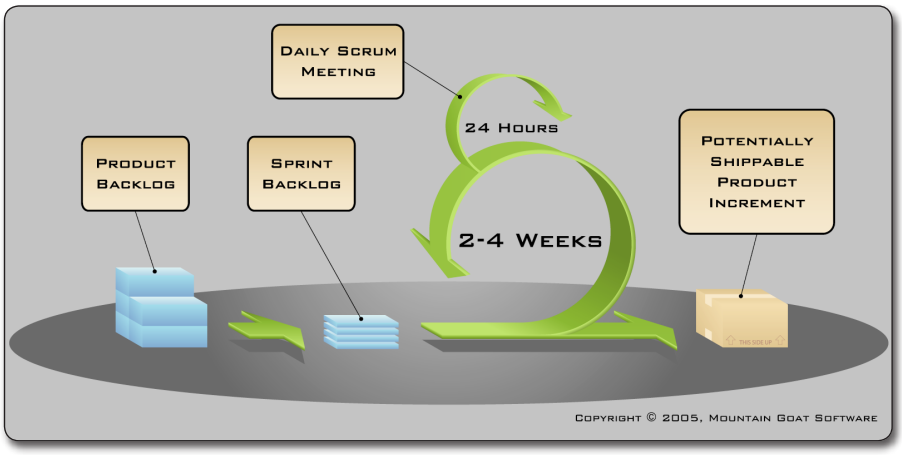

Figura 1. Visão geral do Scrum

Uma questão interessante é a seguinte: "Se o *Scrum* é tão bom e simples de entender, por que é difícil de ser aplicado?". Primeiramente, porque seus principais conceitos são quase que opostos ao que o desenvolvedor está acostumado com processos em cascata. Por exemplo, a ideia de o *Time* ser responsável pelo produto e não pessoas serem responsáveis individualmente por tarefas vai à contra mão do que desenvolvedores estão acostumados. Também, entregas curtas, interativas, com valor e mesmo assim incompletas ou simplificadas geram bastante discordância, pois isso novamente contraria a metodologia empregada em processos em cascata. Finalmente, *Scrum* foca em pessoas e não em processos e, como bem se sabe, relações pessoais, mesmo que de trabalho, podem ser bastante complicadas.

A seguir é dada uma breve descrição de cada etapa e papel do *Scrum*, para que o leitor menos familiarizado com o framework possa entendê-lo um pouco melhor. Entretanto, o foco do artigo são os conselhos a serem aplicados em cada fase dessa metodologia, discutidos depois da apresentação dos mesmos. Finalmente uma discussão é feita sobre aspectos do *Scrum*.

Scrum

Nessa seção é feita uma breve descrição de cada etapa do Scrum, bem como os papéis desempenhados, para contextualizar o leitor. Observe a **Figura 1**: ela representa alguns dos principais conceitos utilizados no Scrum e como eles estão associados.

Sprint

Sprint é o ciclo de desenvolvimento e entregas do *Scrum*. Nele é feito o planejamento, desenvolvimento, testes, entrega e revisão do processo utilizado. O tamanho do *Sprint* varia de projeto para projeto, de *Time* para *Time*.

Alguns duram uma ou duas semanas, outros um mês. O tamanho dele é descoberto, literalmente por tentativa e erro. O *Time* deve testar vários tamanhos de *Sprint* e escolher o ideal. O importante é que, depois de um tempo, ele tenha um tamanho constante, já que o *Sprint* dita o ritmo de desenvolvimento; e um ritmo constante dá dinamismo ao desenvolvimento.

Reunião Diária

Reunião de duração máxima de quinze minutos onde cada membro do *Time* deve responder a três perguntas:

1. O que desenvolvi desde a última reunião até agora?
2. O que planejo fazer até a próxima *Reunião Diária*?
3. Tenho algum impedimento?

Respondendo a essas questões, o *Time* terá uma visão do progresso do *Sprint*, bem como a quantidade e gravidade dos impedimentos. É importante ater-se aos quinze minutos e deixar eventuais discussões para depois. Isso será tratado com mais detalhes em tópicos seguintes.

Reunião de Planejamento

Onde se inicia o *Sprint*, o *Product Owner* mostra a meta e as estórias para o *Time*. O *Time* então negocia tal meta com o *Product Owner*. Após isso, o *Time* estima as estórias e as divide em tarefas. Caso o *Time* perceba que a meta está muito agressiva, uma nova negociação com o *Product Onwer* é feita a fim de combinar algo factível.

Reunião de Revisão do Sprint

Nessa reunião o *Time* faz uma demonstração das funcionalidades desenvolvidas no *Sprint* atual para o *Product Owner* e eventualmente para o cliente. Então, o *Product Owner* aceita ou rejeita o produto entregue no *Sprint*. Também é dado valioso feedback ao *Time* sobre o que foi desenvolvido e sugestões para melhorias em *Sprints* futuros.

Reunião de Retrospectiva

Aqui o *Time*, junto com o ScrumMaster, discutem o que pode ser melhorado em relação ao desenvolvimento do produto.

Aqui se discute tudo, processos, testes, relacionamento com o cliente e entre membros da equipe. A ideia é que sempre se busque e implemente melhorias na prática de desenvolvimento de software.

Papéis

O *Scrum* possui apenas três papéis a serem desempenhados:

• **ScrumMaster**: Pessoa responsável por garantir que o *Time* execute o *Scrum* de forma adequada, bem como resolver impedimentos.

• **Product Owner**: Pessoa responsável entender o produto, bem como planejar releases e *Sprints* para que o software em questão seja implementado com sucesso. O *Product Owner* tem um profundo conhecimento sobre o negócio no qual o produto atua e dá todo o suporte ao *Time*, explicando exatamente o que deve ser desenvolvido. Quaisquer decisões de escopo, prazo e priorização recaem sobre o *Product Owner*.

• **Time**: Grupo de pessoas responsável efetivamente por desenvolver o produto. Aqui se tem Arquitetos, DBAs, Desenvolvedores, Testadores, etc.

Ritos do Scrum: problemas e soluções

Aqui são discutidos, para cada cerimônia do *Scrum*, alguns dos principais problemas, bem como algumas formas resolvê-los.

Reunião de Planejamento

Como mencionado anteriormente, a *Reunião de Planejamento* é onde se inicia o *Sprint*. Essa é uma cerimônia em que alguns cuidados devem ser tomados:

1. Por ser uma reunião de apresentação e posterior discussão, tem-se que ficar atento para que a mesma não se torne muito longa; no máximo um dia.
2. Times imaturos tendem a aceitar as metas sugeridas pelo *Product Owner*, sem realmente verificar se ela é atingível. Deve-se ter muito cuidado para não subestimar o trabalho necessário, principalmente porque durante a apresentação da meta, as tarefas ainda não foram quebradas e talvez sua complexidade ainda não seja clara.
3. O *Product Onwer* não deixa clara a prioridade das estórias, o que consequentemente faz com que o *Time* não as implemente na ordem correta. Caso ele não se atente a isso, *ScrumMaster* deve alertá-lo.
4. A segunda parte da reunião, onde se quebram as estórias em atividades, pode ser excessivamente demorada, pois algumas pessoas tendem a esmiuçar atividades mais do que o necessário. O Time deve acordar na definição do detalhamento máximo de cada atividade. E já respondendo a pergunta que vem na cabeça do leitor: sim, esse detalhamento é um conceito bastante abstrato e difícil de acordar, mas com a prática isso se torna fácil.
5. Alguns membros do *Time* participam pouco da reunião ou não prestam atenção, esperando que posteriormente outros membros os auxiliem nas suas atividades. O *ScrumMaster*, mas principalmente o Time, deve estimular cada membro a participar ativamente da reunião para tudo fique claro para todos.

Provavelmente, muitos dos problemas acima já foram enfrentados pelo leitor que utiliza *Scrum*. Aqui são dados alguns conselhos para que essa cerimônia seja feita da maneira mais proveitosa possível, evitando os problemas mencionados e outros não citados.

Na primeira parte da reunião, onde o *Product Owner* apresenta a meta e estórias, algo que deve ficar claro é: o que, em termos de funcionalidades que agregam valor, quer-se atingir para o próximo *Sprint*. Não basta listar estórias e dizer que as mesmas são as metas, tem-se que criar uma meta que mostre o que o cliente terá de real valor ao seu sistema. Veja o exemplo na **Figura 2** de uma boa meta, juntamente com suas estórias priorizadas.

Nesse exemplo obviamente são mostradas meta e estórias bastante modestas, já que provavelmente mais estórias caberiam no *Sprint*. Note que a meta não dá muitos detalhes do que será implementado. Com ela, o *Time* conhece o que é importante a ser desenvolvido para o *Sprint* que se inicia. Por tanto, caso o *Time* não consiga, por exemplo, implementar a estória de "Enviar o relatório por e-mail", o *Product Owner* ainda assim poderia aprovar o Sprint, já que a meta todavia fora considerada atingida.

Agora, na **Figura 3** observe um exemplo de uma meta ruim e estórias escritas de forma que não agreguem valor ao produto.

Note no exemplo que a meta lista com muitos detalhes o que deve ser desenvolvido no produto. Logo, não é uma meta e sim uma listagem de itens. O problema de se utilizar uma meta muito descritiva é que o *Time* não tem a noção do que é importante para o cliente, em outras palavras, não se tem o senso de que será agregado de importante ao produto. Portanto, logo após a descrição da meta, é importante também dizer ao *Time* como a mesma impacta positivamente o cliente.

Veja as estórias mostradas para a segunda meta, na **Figura 3:** independentemente elas não adicionam funcionalidades ao sistema. Implementar apenas uma tela ou regras de negócio não geram algo útil para o cliente. Uma estória deve ser uma funcionalidade completa, mesmo que seja algo mais simples do que o objetivo final da mesma, mas seja algo que já possa ser usado pelo cliente. Na verdade, as estórias descritas para a segunda meta são ótimos exemplos de tarefas para uma estória maior. Por exemplo, as estórias 1, 2 e 3 da **Figura 3** poderiam ser tarefas de uma estória de criação de Cadastro de Produtos, da **Figura 2**.

Como mencionado anteriormente, alguns times subestimam o tamanho das tarefas, ou ainda, aceitam o que o *Product Owner* propõe por pura pressão. É de suma importância que, para evitar isso, o *Time* entenda bem o que é requisitado nas funcionalidades. Ele deve questionar o *Product Owner* até ter total entendimento das estórias. O *Product Owner*, por sua vez, deve exigir o melhor do *Time*, sendo seu trabalho fazer com que o *Time* desenvolva o máximo possível em um tempo razoável. Em resumo, uma boa negociação deve ser feita para acordar-se uma meta de tamanho exato para o *Sprint*. Isso só

se alcança com experiência; o *Time* e o *Product Owner* errarão por alguns *Sprints* até encontrar a medida exata.

Em relação à segunda parte da *Reunião de Planejamento*, quando se quebram as estórias em atividades, deve-se atentar a algumas coisas para que a reunião termine em um tempo razoável. Para cada estória e atividades, deve-se já propor a solução para as mesmas e quebrá-las em tarefas que durem mais ou menos um dia. Muitas vezes, ocorrem discussões longas a fim de se determinar a melhor solução para cada estória. Para evitar isso, devem-se anotar as soluções sugeridas, quebrar a tarefa em atividades e postergar tais discussões. Com algumas soluções em mãos, as pessoas responsáveis pelas tarefas podem estudar melhor o problema e propor o melhor caminho, durante a fase de desenvolvimento.

A descrição das tarefas é algo importante. Muitas vezes, as mesmas são descritas sem muitos detalhes, o que posteriormente dificulta o entendimento das atividades, pois a pessoa que trabalhará na tarefa não se lembrará de todos os detalhes. Por tanto, o *Time* deve experimentar vários formatos e tamanhos de tarefas até que encontre o tamanho ideal para as mesmas.

As estimativas, geralmente são feitas usando *Scrum Planning Poker*. Tal metodologia, resumidamente, consiste no seguinte: cada membro da equipe fica com um baralho de cartas que seguem a sequência de Fibonacci (1, 3, 5, 8, 13, 21, 34, 55,...). Uma estória base é usada como referência para todo o desenvolvimento do produto. Para ela, um número (tamanho) da sequência de Fibonacci é utilizado. Logo, quando se estima cada estória cada membro deve pensar: "O quão maior ou menor é a estória atual em relação à base?" Caso a estória base tenha o valor de 8 e a estória atual seja desenvolvida em metade do tempo, 5 deve ser o valor escolhido. Caso a estória atual necessite de quatro vezes mais tempo, o valor 21 deve ser escolhido. Para cada estória as ser estimada, as pessoas mostram, como num jogo de Pôquer, o valor que acha correto, e caso haja discrepância, o *Time* discute até chegar a um acordo.

Note a importância das discussões nessas estimativas. Aqui fica claro se todo o *Time* entendeu com clareza o que deve ser desenvolvido. Depois das estimativas, caso a equipe perceba que concordou com uma meta não factível, ela deve chamar o *Product Owner* e renegociar a meta.

Reunião Diária

Como explicado anteriormente, a *Reunião Diária* tem o objetivo de que, cada membro da equipe, saiba exatamente como anda o andamento do Sprint e, caso haja impedimentos, membros do *Time* possam ajudar uns aos outros. Ainda, o *ScrumMaster* pode resolver impedimentos nos casos em que a equipe não possa se ajudar (fora da *Reunião Diária*).

A principal dificuldade que os times encontram nessa cerimônia é ater-se estritamente ao seu propósito, que é: relatar o que foi feito até agora, o que será feito até a próxima *Reunião Diária* e se existe algum impedimento. Essa reunião deve durar no máximo 15 minutos. Nesse rito, as pessoas não "resistem à tentação" e começam a discutir seus

Meta

"Para o final desse Sprint, deseja-se que o usuário possa cadastrar e gerenciar produtos. Além disso, o usuário terá acesso de produtos cadastrados por mês."

Nota do PO: com isso, o cliente consegue armazenar de forma digital seus produtos, eliminando o controle de produtos e estoque em papel, algo bastante demorado e, portanto custoso.

Estórias

1. Como usuário desejo cadastrar e editar produtos.
2. Como usuário desejo listar e apagar produtos.
3. Como usuário, desejo visualizar um relatório que mostra os produtos cadastrados por mês. Nesse relatório deve-se mostrar a quantidade dos mesmos, mês a mês, caso mais de um mês esteja no filtro de seleção.
4. Como usuário, desejo que o relatório da estória de exportação de relatório de produtos cadastrados, possa ser exportado para Excel.
5. Como usuário, desejo que relatório de estória de exportação de relatórios de produtos cadastrados, possa ser envidado por e-mail.

Figura 2. Meta adequada

Meta

Gostaria que o usuário pudesse:

1. Cadastrar Produtos
2. Editar Produtos
3. Listar Produtos
4. Gerar relatórios de Produtos cadastrados mensalmente
5. Exportar relatório anterior para Excel
6. Enviar relatório do item 4 por email.

Estórias

1. Criar tela de cadastros de produtos.
2. Criar camada de negócio de cadastro produtos.
3. Criar camada de persistência de cadastro de produtos.
4. Criar tela de listar e apagar produtos.
5. Criar camada de negócio de listar produtos.
6. Criar camada de persistência de listar produtos.
7. Criar camada de negócio de apagar produtos.
8. Criar camada de persistência de apagar produtos.
9. Criar tela de relatório de Cadastro de Produtos por mês.
10. Criar camada de negócio de relatório de Cadastro de Produtos por mês.
11. Criar camada de persistência de relatório de Cadastro de Produtos por mês.
12. Criar framework de exportação de Excel.
13. Exportar relatório de Cadastro Mensal de Produtos para Excel.
14. Criar framework de envio de e-mail.
15. Enviar relatório de Cadastro Mensal de Produtos via e-mail.

Figura 3. Meta inadequada

problemas e dar soluções sobre questões do projeto, não percebendo que isso não é o objetivo desse encontro. Essas discussões fazem que as *Reuniões Diárias* passem de 15 minutos para uma hora, onerando bastante a performance do *Time*, já que muito tempo é despendido. Portanto, é de suma importância que se execute tal reunião estritamente como ela é concebida, e fica a cargo do *ScrumMaster* doutrinar o *Time* para que isso ocorra. O melhor conselho para que o *ScrumMaster* consiga tal objetivo é o seguinte: seja rígido, interrompa sempre que o foco da reunião for desviado, até que o *Time* se acostume. Talvez leve um pouco de tempo, mas com insistência a reunião flui corretamente.

A **Figura 4** é bastante ilustrativa, em como se executar uma *Reunião Diária*. Veja que as pessoas estão em pé, focadas no quadro de atividades. Isso ajuda bastante o foco nas tarefas em questão e, o fato de ficar em pé, faz com que o *Time* tenha vontade de terminar mais rápido, para voltar a suas cadeiras.

Figura 4. Reunião Diária

Sprint

Na execução do *Sprint*, alguns cuidados devem ser tomados. O problema mais comum é a necessidade de inserção de estórias ou funcionalidades novas no meio do *Sprint*. Muitas vezes, *Product Owners* são pressionados por clientes ou gerentes de projeto a colocar funcionalidades, consideradas emergenciais, para a próxima entrega. O *Time* deve ter autonomia de rejeitar tais funcionalidades, caso elas modifiquem a meta e arrisquem o sucesso do *Sprint*. Caso isso não seja permitido pela empresa, então o sucesso do *Scrum* estará em perigo, pois se sua metodologia não pode ser seguida corretamente, talvez esse não seja o melhor modelo a ser adotado (lembre-se que o *Scrum* não é a "bala de prata" que resolve todos os problemas e não utilizá-lo não significa fracasso). Por outro lado, caso o pedido do *Product Owner* seja razoável, algo que não signifique a mudança de meta e não resulte na falha do *Sprint*, o *Time* deve atendê-lo. Perceba que, caso novas funcionalidades realmente sejam emergenciais e necessárias, o *Sprint* deve ser cancelado e uma nova *Reunião de Planejamento* necessita ser agendada para o começo de um novo *Sprint*.

Outro erro comum que ocorre é a não priorização das atividades. É natural para cada membro do *Time*, executar as atividades que seja mais proficiente. Isso geralmente significa, para muitos, começar por tarefas de prioridade menor. Portanto, algumas atividades de menor prioridade eventualmente serão terminadas antes de tarefas de maior prioridade. Ou ainda, atividades de maior prioridade não estão terminadas ao passo que várias tarefas de menor prioridade já estão começadas. Perceba o principal benefício de se terminar tarefas com prioridade maior: caso o *Sprint* falhe e a questão de prioridade seja atendida, as estórias de maior prioridade já estarão terminadas, o que significa que o cliente receberá o que lhe é de mais valor, mesmo que não receba tudo. Por outro lado, caso as prioridades não sejam atendidas e o *Sprint* falhe, o mesmo pode terminar com todas as estórias começadas, mas nenhuma terminada, portanto sem nada a ser entregue ao cliente final.

A **Figura 5** mostra um exemplo do quadro de atividades de *Scrum*. Note que as atividades são ordenadas por prioridade. Tal apelo visual ajuda bastante o *Time* a organizar-se para implementar tarefas de maior prioridade antes de começar as demais.

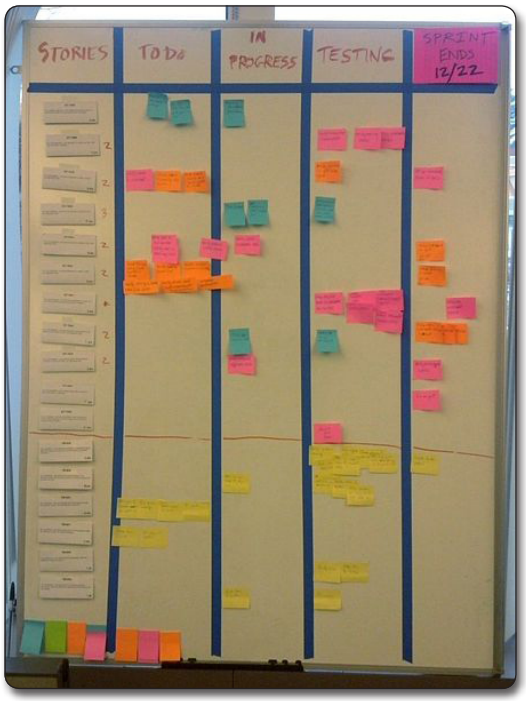

Figura 5. Quadro de atividades

Apesar do *Scrum* não abordar esse assunto especificamente, algo importantíssimo que sempre fica de lado é a questão de testes. É muito comum para desenvolvedores de software postergar testes, sempre que possível. Contudo, é importante lembrar que as entregas de cada *Sprint* devem ser passíveis de utilização pelo cliente, ou seja, devem estar bem testadas e funcionais. Em outras palavras, deve-se evitar entregar estórias não testadas e, ao final do projeto, fazer alguns *Sprints de Testes*. O motivo é simples: até se chegar nesses *Sprints de Testes*, o produto não poderá ser utilizado pelo cliente, o que coloca por terra todo o conceito de entregas iterativas. Entregas só são entregas caso possam ser utilizadas pelo cliente, caso contrário, são apenas protótipos detalhados sem valor agregado.

Reunião de Revisão do Sprint

Essa cerimônia é a que menos se encontram problemas, pois é algo que o *Time* já está acostumado a fazer em outras metodologias. Aqui, como dito anteriormente, o *Time* apresenta ao *Product Owner* o que foi implementado para alcançar a meta. Nessa reunião, deve-se focar no *Sprint* atual, ou seja, nem o *Produt Owner*, nem o *Time* devem discutir funcionalidades anteriores. Além disso, é perfeitamente aceitável uma entrega com bugs, desde que os mesmos não impeçam a utilização do sistema. Aqui, é crucial a honestidade do *Product Owner* em relação ao que pensa sobre o que foi entregue. Nessa reunião, o *Time* recebe feedbacks e deve aceitá-los de forma não pessoal, lembrando que a crítica é ao produto e não aos membros da equipe. É uma reunião que deve ser levada com seriedade, mesmo que o *Product Onwer* tenha acompanhado todo o desenvolvimento durante o *Sprint*. Um fato interessante é que nessa cerimônia, rotineiramente dúvidas e questionamentos aparecem, os quais dificilmente seriam notados se o escopo completo do que foi desenvolvido no *Sprint*, não fosse apresentado diante do *Time* e do *Product Owner*. Uma observação interessante: sempre que possível, é conveniente trazer o representante do cliente para assistir essa reunião. Isso faz com que o *Time* receba feedback imediato do cliente final, além de aproximá-los, devido a estarem no mesmo ambiente interagindo, algo que sempre gera frutos positivos.

Algo que deve merecer atenção na *Reunião de Revisão do Sprint* é a maneira que feedbacks são dados. É importante que o *Product Owner* direcione os comentários para o *Time* e não para a pessoal que implementou a funcionalidade criticada. O mesmo vale para elogios: todo o *Time* deve ser elogiado e não uma pessoa em particular. Isso dá ao *Time* a sensação, que é verdadeira, de que todos são responsáveis pelo fracasso ou sucesso do *Sprint* e, num contexto mais geral, do produto. Além do mais, isso evita que membros da equipe tenham problemas pessoais com o *Product Owner*, e vice e versa.

Reunião de Retrospectiva

Essa reunião, sem dúvida, é a que gera as mais acaloradas discussões. Portanto, deve-se tomar bastante cuidado para que desavenças entre membros do *Time* não ocorram. Logo, alguns conselhos são mostrados a seguir para que essa cerimônia ocorra da maneira mais tranquila possível. O *ScrumMaster* é a pessoa mais indicada para conduzir a reunião da melhor forma possível.

Primeiramente, como na *Reunião de Revisão do Sprint*, sempre que possível, ao fazer-se uma crítica, não direcioná-la a um membro da equipe. Críticas devem ser feitas ao *Time* todo, pois o *Time* é responsável pelo produto. Quando se menciona nomes, a discussão se torna pessoal, o que mina o objetivo da reunião, que é o melhoramento contínuo do processo de desenvolvimento.

Outro ponto é o seguinte: o foco da discussão deve ser o projeto. Quando reuniões em que o objetivo é, entre outros, apontar defeitos, muitas vezes o tópico principal da conversa é desviado; membros do *Time* começam a reclamar da empresa, de outros times, contar histórias pessoais, enfim, discutir qualquer outro assunto que não seja o projeto. Portanto, o *Time* deve se esforçar para focar no assunto em questão fazendo com que a reunião efetivamente tenha resultado e não dure toda uma manhã ou tarde, com poucas questões relevantes discutidas.

Além disso, é importante estimular que todos os membros da equipe falem. Mesmo que seja algo redundante, todos devem dar sua opinião. Isso melhora a integração entre a equipe e muitas vezes pessoas mais tímidas expõem pontos de vista bastante interessantes e diferenciados, pois falam pouco e observam mais.

Outro bom conselho: quando se aponta um problema, deve-se sugerir uma solução. Só reclamar, como todos sabem, não adianta. Portanto, sempre quando uma crítica for trazida, é importante que a mesma pessoa traga uma possível solução. Isso traz dois benefícios: primeiramente, já se tem algo para começar a resolver o problema e segundo, o *Time* vê cada um dos seus pares de uma forma muito mais amistosa, pois se trazendo a solução, a pessoa que a trouxe cria para si a imagem de alguém que quer ajudar a resolver os problemas.

Algo interessante que acontece em times maduros é a aparente falta de necessidades de melhorias. Algumas vezes, depois de certo tempo de trabalho, o *Time* não sugere mais melhorias, pois crê que já está num ritmo bom e nada mais pode ser aprimorado. Isso é um engano. É muito difícil, para não dizer impossível, estar em um grau de desenvolvimento que não haja espaço para aperfeiçoamentos.

Deve-se verificar se algumas metas comuns a desenvolvimento ágil já foram atingidas, tais como: testes automatizados, poucos bugs, integração continua, código com boa qualidade, etc. Poucos times chegam a executar todas as boas práticas citadas. Para os que conseguem, é a hora de "pensar fora da caixa" e sugerir atividades que melhorem ainda mais o que já está bom.

Papéis no Scrum e principais desafios

Nesse tópico são discutidos os principais desafios de cada papel exercido no *Scrum*.

Product Owner

O principal desafio do *Product Owner* é entender o produto. Tarefa que não tem nada de trivial. Imagine-se na situação de criar algo que não existe. Nessa simples sentença, esse é o desafio do *Product Owner*. Para que isso seja possível, aqui vão alguns conselhos.

Primeiramente, estude muito sobre o produto a ser criado. Vá além do papel de um Analista de Requisitos. O negócio em torno do produto a ser desenvolvido precisa ser entendido. O que tal produto trará de ganho deve ficar perfeitamente claro. Em outras palavras, o *Product Owner* precisa se tornar temporariamente especialista no negócio referente ao que será criado.

Além disso, uma estratégia de priorização e *Time to Market* (é o intervalo de tempo, desde a concepção até a disponibilidade de venda de um produto. Esse conceito é importante onde produtos são considerados obsoletos rapidamente) devem ser traçados. É de suma importância que as primeiras funcionalidades do produto sejam as que mais tragam valor ao negócio do cliente. Perceba como isto está intimamente ligado com a questão de priorização, que sempre é enfatizada no *Scrum*. Além disso, um produto tem seu tempo certo para ser lançado, ou o cliente perderá mercado para a concorrência, daí vem a importância do *Time to Market*.

Outro desafio fundamental para o *Product Onwer* é a comunicação. Quem atuar nesse papel deve comunicar-se adequadamente, tanto com o cliente, como com a equipe técnica que desenvolve o produto. Isso significa participar de dois mundos distintos: o do cliente que conhece apenas jargões de negócio do meio onde atua, e do *Time* de desenvolvimento, que possui um linguajar extremamente técnico. Por esse motivo, profissionais que atuam nesse papel geralmente originam de uma carreira relacionada a desenvolvimento de software, já que estão aptos para entender o mundo dos clientes e estão acostumados a lidar com times técnicos.

Time

O *Time* engloba toda a equipe técnica, responsável pelo desenvolvimento do produto. Fazer parte de um *Time* de *Scrum* não é nada simples, principalmente para quem vem do mundo de metodologias em cascata. Profissionais, em geral, não apenas do ramo de desenvolvimento de software, tem o hábito de desenvolverem atividades completas, do início ao fim. No *Scrum*, isso muda bastante. O *Time* tem como o objetivo acabar estória mais prioritária e apenas depois, partir para a próxima. Isso não é natural para profissionais em geral, pois durante todas as suas vidas, pessoas são doutrinadas a trabalharem sozinhas em atividades completas, ou mesmo que trabalhem em grupo, dividem suas atividades de maneira a terem algo de tamanho aceitável, sem o desafio de priorização de estórias.

Na **Figura 2** veja o exemplo da primeira estória: "Como usuário desejo cadastrar e editar produtos.". Para um *Time* de duas pessoas, a divisão de tal estória seria algo trivial. Por exemplo, um dos desenvolvedores criaria a interface com o usuário e outro a camada de negócios. Contudo, imagine agora cinco desenvolvedores. O desafio é maior. O primeiro pensamento é: existe pouca tarefa para muitas pessoas, portanto o *Time* poderia decidir começar mais estórias. Muitas vezes, essa é a melhor solução, pois, nesse exemplo, se fosse um *Time* de nove pessoas, certamente não seria possível quebrar essa estória atividades suficientes para tantas pessoas. Entretanto, será que não é possível quebrá-la para cinco pessoas? A resposta a essa questão depende da arquitetura e da criatividade dos desenvolvedores. Um software desenvolvido em MVC (Model View Controller) poderia ter duas pessoas responsáveis pela tela: uma para o design e outra pelas interações com Javascript, uma pessoa desenvolvendo o *Controller* e mais duas na camada de persistência e negócios que, se tivessem baixo acoplamento poderiam ser divididos.

Outro enorme desafio para os times é a questão de testes. Processos em cascata viciam desenvolvedores a deixarem testes para o final do desenvolvimento. É preciso reeducar o *Time* a sempre testar e entregar produtos estáveis e de boa qualidade a cada *Sprint*. Para se alcançar isso não há segredo: deve-se insistir nessa meta, até que a mesma seja alcançada. Isso não acontecerá no primeiro *Sprint*, mas o Time deve lutar para uma melhora constante até que tal prática seja natural.

ScrumMaster

Esse, sem dúvida, é o papel mais complicado do *Scrum*, basicamente porque o *ScrumMaster* não possui autoridade alguma sobre o Time. Ele deve persuadir a equipe a utilizar essa metodologia sem nenhuma imposição ou poder. Tal argumentação para desenvolvedores não acostumados a isso é longa e penosa. A grande maioria deles só vê valor no *Scrum* depois de alguns meses utilizando o framework. Durante esse período, o *ScrumMaster* deve ter muita paciência e tenacidade para lidar com inseguranças e questionamentos do *Time*.

Outro desafio, que infelizmente, não depende apenas do *ScrumMaster* é a resolução de impedimentos. Muitas empresas não dão autonomia para que a pessoa que desempenha esse papel possa de fato, por exemplo, interagir com clientes ou com a gerência da empresa onde trabalha, para resolver impedimentos. Aí entra a criatividade: o *ScrumMaster* deve ser habilidoso para persuadir outras pessoas a lhe ajudarem. Por exemplo, ele deve convencer o *Product Owner* a conversar com o cliente a fim de resolver alguma pendência (suponha pendências de requisitos para a finalização de uma estória). Ou ainda, ele deve ter um bom relacionamento com outras

áreas para conseguir, por exemplo, recursos materiais para terminar uma estória (por exemplo, PCs com entradas seriais para comunicação com máquinas antigas).

Além disso, *ScrumMaster* deve ter profundo conhecimento do *Scrum* para direcionar o *Time* de modo a segui-lo corretamente. Um concelho importante é: não faça o *Time* adaptar-se ao *Scrum* no primeiro *Sprint*. É saudável que o *Time* adapte-se ao framework paulatinamente, para que se acostume a ele e o processo transitório seja o menos traumático possível. Reuniões de Retrospectiva são sempre interessantes para introduzir novos passos e conceitos a serem seguidos no *Scrum*.

Scrum e uma solução definitiva

Muito se discute se *Scrum* é a solução para todos os problemas de processos em cascata. A resposta é simples: não. *Scrum* resolve vários problemas, como o encorajamento a mudança, a valorização do indivíduo em relação a processos, entre outras questões mencionadas aqui. Contudo, alguns problemas ainda permanecem.

"Estimativas" é um problema sem solução ainda. Apesar das estimativas por pontos no *Scrum*, após uma calibração por vários *Sprints*, mostrar bom resultados, ainda se tem o sério problema de entender-se e estimar um projeto de médio a longo prazo. Mesmo atividades simples são mal estimadas, até hoje. Em outras palavras, no *Scrum* se consegue estimar estórias a serem desenvolvidas no período de um *Sprint*, mas ainda não se consegue estimar com exatidão um projeto qualquer, com duração de um ano, por exemplo.

Lidar com pessoas ainda é um desafio, e provavelmente o será, por muitos anos. O *Scrum* ainda "piora" esse problema, pois tira totalmente o foco de processo e o coloca em pessoas.

Scrum, apesar de fácil de entender, é um processo dificílimo de seguir, como discutido em tópicos anteriores.

Qualidade do código

Em processos em cascata, a qualidade do código era bem controlada se o projeto possuísse um arquiteto com bons conhecimentos e autoridade. Com *Scrum*, como o *Time* é responsável pelo produto e pelo código, não se é permitido ter uma pessoa com autoridade para forçar membros da equipe a codificarem com qualidade. Com frequência, boa parte do *Time* não se importa com a qualidade do código. Isso gera problemas enormes no futuro, pois, com a necessidade de manutenção contínua e com o dinamismo muito grande dos *Sprints*, código mal estruturado começa fazer com que o *Time* diminua sua performance pois é obrigado a interagir com código de má qualidade. Portanto, o Time deve se conscientizar e garantir uma boa qualidade, tanto do produto, como do código. O *ScrumMaster* pode ajudar bastante nisso, mostrando ao *Time* as vantagens de se codificar com qualidade.

Testes automatizados

Testes automatizados, de maneira alguma são obrigatórios para o sucesso de um projeto. Principalmente se o mesmo for de curta duração. Contudo, para a criação de produtos que terão uma longa vida e serão extensos, recomenda-se fortemente a criação dos mesmos. O tempo dispendido na sua criação e manutenção é ganho durante testes de regressão que são feitos automaticamente pelos testes.

Veja o seguinte cenário: imagine um produto onde, caso as funcionalidades atuais "quebrem" algo desenvolvido anteriormente, o desenvolvedor seja imediatamente avisado de tal problema. Nesse mesmo produto, não são necessários testes de regressão nem a preocupação com funcionalidades já desenvolvidas que atualmente estão em funcionamento. Esse produto, necessariamente possui testes automatizados.

Entretanto, não se engane pensando que testes automatizados são a "bala de prata" para problemas de testes de produtos. Mesmo com eles, ainda podem existir erros oriundos de novas funcionalidades que não foram detectados pelos testes automatizados. Contudo, a frequência destes será bem menor.

Scrum, sem dúvida alguma, resolve inúmeros problemas de processos em cascata. Contudo, desenvolvimento de software ainda continua uma atividade complexa e com muitos desafios. Por fim, observe que o *Scrum* deve ser encarado como um framework que ajuda o desenvolvimento de software, mas é apenas uma metodologia. O sucesso do produto criado está nas mãos do *Time* que trabalha nele.

Links:

Manifesto Ágil
http://agilemanifesto.org/