Métodos Ágeis em TI

UNIDADE 07

Escolha

*Nesta unidade, faremos a **escolha** e responderemos às perguntas: qual é o melhor método? Com qual método devo começar?*

Escolha

Agilidade é compreender que todas as metodologias têm sua função e aplicabilidade. Com certeza, após conhecer algumas com bastante propriedade, você poderá customizá-las para chegar no processo que mais se adequa a sua realidade.

Introdução

Muitos agilistas, experientes ou não, acreditam que apenas seguir os processos e métodos da linha de pensamento ágil será suficiente para atingir os objetivos de escalar ou melhorar um time de desenvolvimento de *software*. Infelizmente, a adoção bem-sucedida dificilmente acontece dessa maneira, e armadilhas e desafios não são vistos até que seja tarde demais.

Acreditamos que o uso das melhores práticas é a coisa certa a se fazer. *Scrum*, XP e *kanban* são conhecidos como métodos excelentes para lidar com a maioria dos problemas que enfrentamos em nossas vidas diárias de desenvolvimento de *software*. A expectativa é que, com o uso dessas ferramentas, possamos encontrar uma forma rápida, boa e barata de agregar valor.

Para entender o potencial da linha de pensamento ágil, precisamos primeiramente explorar alguns conceitos por trás dela; vamos começar com eficiência e eficácia.

Eficácia e eficiência

A maioria das pessoas considera eficiência e eficácia como sinônimos, mas Peter Drucker, pai da administração moderna, mostra que há uma enorme diferença entre eles.

> **“Eficiência: fazer as coisas da maneira certa.”**

Executando mais trabalhos com menos, com baixo índice de defeitos, pouca variabilidade e escala. No campo da eficiência, precisamos estabelecer processos, regras, automação, ferramentas etc.

Uma ferramenta importante para nos ajudar a aumentar a eficiência é a automação. Existem algumas decisões unânimes sobre como começar:

- Testes unitários.
- Compile, *build* e *deploy*.
- A configuração do ambiente de desenvolvimento.

> **“Eficácia: fazer as coisas certas.”**

Muito alinhada ao valor da simplicidade, fazendo somente o que deve ser feito e evitando fazer o que não deve ou precisa ser feito. Capacidade de descobrir quais recursos agregam mais valor e o melhor momento para implementá-los.

Outro valor importante são os ciclos de aprendizado curtos. Quanto menor o tempo entre a criação de uma hipótese e a sua validação, mais eficaz se tornará o resultado geral.

A razão é que cada vez que você valida uma hipótese, aprende um pouco mais sobre as necessidades do cliente e o contexto para o seu trabalho, bem como o comportamento do *software*. Com este novo conhecimento, você pode adaptar o desenvolvimento de *software*, recursos, processos, abordagens de gestão etc.

Ainda existem equipes de desenvolvimento de *software* que não sabem ou não se importam muito com o motivo pelo qual um aplicativo de *software* está sendo desenvolvido. Podemos dizer que eles pensam muito sobre eficiência e muito pouco sobre eficácia. Notamos esse comportamento mesmo em equipes ágeis: eles são incríveis em fazer as coisas rapidamente e com baixas taxas de defeitos, mas geralmente não estão cientes dos problemas de negócios que estão tentando resolver.

Os desenvolvedores de *software* devem sempre pensar em desenvolver o recurso certo e no momento certo: sempre se concentre no que são as coisas mais importantes a serem feitas em um determinado momento.

Agora, vamos dar um passo adiante combinando esses conceitos.

Para fins de compreensão, nos limitaremos a apenas três cenários possíveis:

1. Faça a coisa errada da maneira certa: este é o cenário mais prejudicial, pois muito esforço é gasto em vão. Não há valor implementado.
2. Faça a coisa certa da maneira errada: este não é o melhor cenário, mas é melhor do que o primeiro. Erros são cometidos, mas há valor implementado.
3. Faça a coisa certa da maneira certa: este é o melhor cenário e deve ser o objetivo da equipe. O valor implementado é o mais alto possível.

Assista ao vídeo a seguir, no qual retomaremos as perguntas que fizemos nesta Unidade e revisaremos os conceitos já estudados para formular nossas respostas.

Métodos ágeis

Neste vídeo, vamos revisar esses conceitos e olhar novamente para as perguntas que queremos responder nesta Unidade.

Case

Uma história interessante é a da Objective Solutions, que escalou sua equipe de desenvolvimento de *software* de 40 desenvolvedores para mais de 250, trabalhando de forma colaborativa em três sedes diferentes: São Paulo, Curitiba e Maringá. A ideia naquele momento era escalar a equipe, explorando os desafios e as soluções deste processo sem perder os valores da linha de pensamento ágil. Ao fundar sua terceira sede em Maringá, começaram usando *extreme programming*, com práticas como histórias de usuário, programação pareada e desenvolvimento orientado a testes.

Após algumas observações, perceberam certa sobrecarga nas cerimônias devido ao fluxo como as atividades chegavam até essa unidade. Demandas urgentes aconteciam com frequência, e mesmo com iterações curtas de uma ou das semanas, logo após o início da iteração já havia novas histórias para serem estimadas e planejadas, muitas vezes atrapalhando todo o planejamento já feito para a iteração. Isso também gerava um descontentamento no time, ao ver todo um trabalho de planejamento ser interrompido pelas urgências.

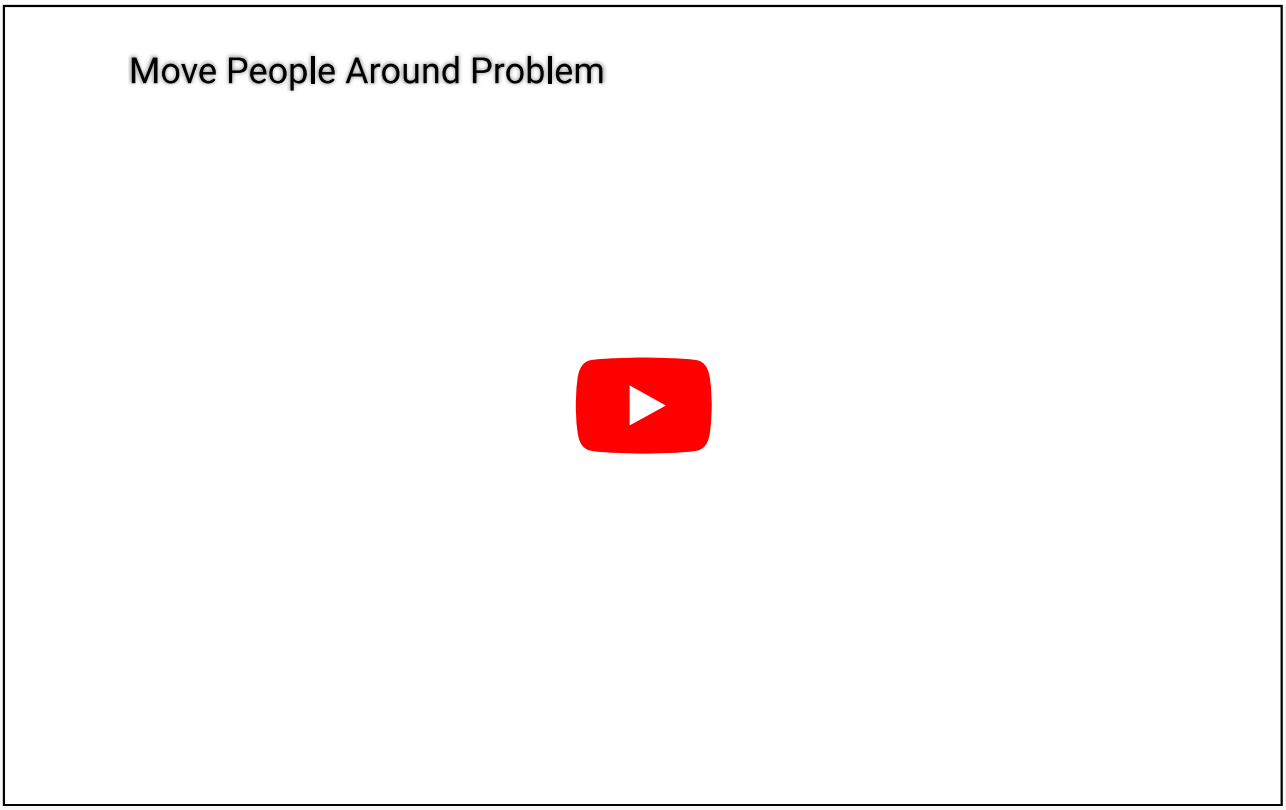
A solução foi que não havia espaço naquele ambiente para se trabalhar em lotes, no caso as iterações. Assim, todas as histórias iam sendo trabalhadas de acordo com a sua chegada, conforme demonstrado na animação a seguir.

A animação demonstra como funciona um fluxo contínuo de atividades em um time de desenvolvimento de software, sem iterações ou sprints definidas.

Adicionalmente à troca de iterações para fluxo contínuo de atividades, outras ações foram feitas. O tempo investido no planejamento e na estimativa das atividades se mostrou desnecessário, pois essa unidade era responsável exclusivamente pelo desenvolvimento técnico, outras atividades, como atendimento ao cliente, orçamento de atividades e gestão do produto, eram realizadas em outras unidades da empresa. Quando a tarefa era recebida por essa unidade de desenvolvimento, o cliente já havia recebido uma estimativa de tempo e custo da atividade, gerados pelos gestores do produto e pelo time comercial, baseados no histórico de tempo e esforço tomados por atividades similares nesse ou em outros projetos. Assim, o time passou a não estimar novamente e confiar no que já estava definido.

Aplicar programação pareada também exigiu esforço. Surgiram personagens como “o solitário”, com quem ninguém gostava de parear; “o herói”, com quem todos gostavam de parear; e “os amigos para sempre”, duplas viciadas que sempre pareavam juntas, conforme demonstrado na animação apresentada a seguir.

A animação demonstra possíveis problemas que podem ocorrer ao adotar programação pareada do *extreme programming*.



O defeito de um cenário assim é que isso não promove a disseminação do conhecimento, além de poder gerar desmotivação em certos membros do time. Isso foi tratado fortalecendo a prática *move people around*, do próprio *extreme programming*, adotando uma tabela de rodízio de pares, cruzando todos os desenvolvedores e estabelecendo como regra que os pares deveriam seguir o algoritmo. Programação pareada é uma prática que exige muita disciplina e atenção às regras do jogo para funcionar corretamente, como demonstrado na animação apresentada a seguir.

A animação demonstra como a prática *move people around* do *extreme programming* favorece a disseminação do conhecimento, fortalecendo a programação pareada.

Desenvolvedores são otimistas. Se você pergunta para um desenvolvedor como está o andamento de uma atividade na qual ele está trabalhando, a resposta quase sempre é 80%. Desenvolvedores gostam de desafios e sentem que sempre estão a um passo de atingir sua solução. Também dificilmente pedem ajuda, mesmo em pares, costumam achar degradante solicitar ajuda de alguém mais experiente. Isso prolonga o tempo investido na solução das atividades e pode minar a previsão de prazo prevista para o projeto.

A situação oposta também tem suas desvantagens. Desenvolvedores com menor senioridade podem exigir atenção de seus líderes muito frequentemente, tirando tempo do líder para investir em apoiar o grupo mais otimista. Esses dois riscos são demonstrados na animação apresentada a seguir.

A animação demonstra como determinadas situações podem passar despercebidas mesmo a um líder atento.

Confused Coach Solution

Para minimizar esse risco, o time usou desde bandeirolas sobre os computadores, indicando necessidade de apoio, até uma nova prática, chamada de ronda ativa, em que o líder passava diariamente por todos os pares, verificando o andamento de suas atividades e auxiliando como podia, independentemente de o par ter pedido ou não por ajuda, conforme demonstrado na animação apresentada a seguir.

A animação demonstra como uma ronda ativa do líder ajuda muito na fluidez das atividades em um time de desenvolvimento de *software*.

Flag Solution Nurse Round

Ou seja, mesmo times maduros precisam de certo controle. Assista ao vídeo a seguir, em que será entrevistado um profissional sobre o caso apresentado anteriormente.

Entrevista sobre a escolha

Neste vídeo, será entrevistado um profissional de métodos ágeis. Discutiremos o case e as escolhas que fizeram parte dele, com uma visão atual e prática de mercado.

Conclusão

Por um lado, temos as metodologias tradicionais, que são aquelas que tratam do desenvolvimento de *software* com uma abordagem de engenharia, buscando constantemente o controle, a alocação de recursos e planos de longo prazo. A ideia é planejar todos os aspectos do projeto antes de iniciar sua implementação. Você pode passar meses em um projeto tradicional sem implementar nenhum código.

Por outro lado, temos as metodologias ágeis, que tratam do desenvolvimento de software como um processo de aprendizagem. O contexto é que cada interação entre o cliente, a equipe e o *software* codificado produze um novo entendimento sobre os problemas e as soluções dadas. Este conhecimento adquirido pode mudar drasticamente o escopo do projeto, prazos e custos, direcionando o projeto para implementar mais valor.

Podemos dizer que as metodologias ágeis são eficazes, encontrando o *software* certo no momento certo de ser desenvolvido.

O que aprendemos ao longo dos anos sobre metodologias de desenvolvimento:

As metodologias tradicionais estão focadas na eficiência, enquanto as ágeis estão focadas na eficácia.

Desenvolvimento ágil é isso. Não é um método, limitado pela visão, mesmo que brilhante, de um determinado autor. Pois no próprio manifesto, um trecho muito ignorado é o cabeçalho, que diz:

Estamos descobrindo maneiras melhores de desenvolver *softwares*, fazendo-os nós mesmos e ajudando outros a fazê-lo.

Portanto, vale ressaltar que você não precisa (e nem deve) ficar preso a um método só, ou segui-lo nos mínimos detalhes. Há muitos meios de se melhorar os processos de desenvolvimento de *softwares*. Você deve levar em consideração os aspectos únicos do seu negócio para escolher qual mais se adequa ao seu contexto.

Referências

AGILE TOUR 2014. **The Spice Must Flow - Ramon Tramontini e Marcelo Walter**. YouTube, 2014. Disponível em: https://www.youtube.com/watch?v=i38Pu_JVoW4. Acesso em: 18 abr. 2021.

OBJECTIVE. **Como escalar times de alta performance com o mindset ágil e ambidestria organizacional**. YouTube, 2016. Disponível em: https://www.youtube.com/watch?v=VJk3cBKysCE. Acesso em: 01 jun. 2021.