# O que são métodos ágeis?

Olá, seja bem-vindo ao curso de Planejamento Ágil da Alura. Eu sou o André Faria e, nesta primeira parte, nós vamos introduzir um pouquinho dos conceitos de métodos ágeis, de desenvolvimento de software, e por que eles apareceram, pra que eles surgiram, em que contexto e o que existia antes dos métodos ágeis. Isso vai ser uma pequena introdução para que a gente possa fazer referência ao que a gente aprender agora nessa primeira parte nas próximas partes do treinamento.

Então nós vamos falar um pouquinho de tudo isso: os problemas, métodos ágeis em geral, Scrum, XP e Kanban, que são três métodos - eu diria que são os três métodos mais conhecidos e mais utilizados pela comunidade - e também alguns resultados que a comunidade já atingiu.

Quais eram os problemas com os chamados métodos tradicionais de desenvolvimento de software, que era o que existia antes? Basicamente, eles seguiam esse modelo, conhecido como *Waterfall*, o modelo **Cascata**, em que você tinha diversas fases no projeto. Era feito um levantamento muito bem detalhado de tudo o que tinha que ser desenvolvido, uma fase de requisito inicial enorme, de levantamentos de requisitos com o cliente, até porque muitos contratos eram fechados em cima disso - chamados de Contrato de escopo fechado. Então, primeiro se define qual o escopo do projeto, e pra isso a gente precisa ter uma ideia de tudo o que o cliente precisa, o cliente precisa se decidir o que ele quer e o que ele não quer logo de início. Muito embora ele ainda não tenha a oportunidade de ter experimentado, de ter visto nada ainda, ele precisa tomar uma série de decisões nesse momento. É um momento em que ele ainda sabe muito pouco sobre o que ele mesmo quer; tem aquela frase "Ele não sabe o que quer".

Uma equipe de análise faria esse trabalho, depois segue para uma nova etapa, de desenvolvimento de software, em que, por exemplo, uma outra equipe vai fazer a análise de todos esses requisitos. E cada etapa dessa costumava demorar muito tempo. Um exemplo, um projeto que demoraria um ano, dois anos para ser concluído, cada etapa dessa teria vários meses, teria alguns meses de duração.

Depois de analisar, passava-se para outra equipe que fazia o design, fazia a modelagem, utilizava todos aqueles recursos da UML para modelar. Depois disso, depois de muito tempo, começava-se a codificar: uma outra equipe codificava, depois uma outra equipe provavelmente ia fazer os testes - se fisesse -, porque como os testes estão ali no final, é muito comum aquela famosa história de *"Poxa, não dá tempo de testar, vamos fazer a implantação porque o prazo está acabando, a gente vai estourar o prazo."* Então o prazo dos testes geralmente acabava apertando porque estava lá no final. Muitas vezes isso acabava gerando muitos problemas de qualidade também.

Perceba que são várias etapas bem definidas, geralmente realizadas por pessoas com skills diferentes, por equipes, inclusive, diferentes. Só quando uma etapa está completa, passa-se para a próxima etapa - a gente não vê aí, por exemplo, que naturalmente as coisas poderiam voltar de uma etapa pra outra; a ideia é que vá caindo simplesmente de uma etapa dessa cascata para a próxima.

O grande problema dessa abordagem é que, quando o cliente geralmente via o software no final, era aquela famosa frase que ele dizia: *"Não foi isso que eu pedi, não era isso que eu queria. Não era isso que eu precisava, isso não me atende."*

Se, por alguma razão o cliente precisasse parar o projeto no meio do caminho, ele não teria nada funcionando, ele não teria nada que agregasse valor a ele, porque ele só receberia no final do projeto.

Além disso, ele só veria o projeto mesmo, alguma coisa palpável, no final. Então o *ROI* dele, o retorno sobre o investimento que ele fez, ele só vai receber no final do projeto. Além disso, como eu disse antes, no começo, sabe-se muito pouco sobre o projeto ainda, você não teve muita oportunidade de fazer experimento, de fazer testes. Você não conhece ainda muitas vezes aquela equipe, aquele contexto do negócio, do projeto que está fazendo. Então é um dos momentos em que você vai tomar as piores decisões no início do projeto. Todas as decisões importantes são tomadas nesse momento, são tomadas de antemão. Isso é um problema, que acabava causando o que é conhecido na comunidade como *Big Requirements Up Front*, que é a ideia de você levantar todos os requisitos de antemão, e de *Big Design Up Front*, de você tentar fazer todo o design de antemão.

Nós vamos ver que, com métodos ágeis, possivelmente, como vamos falar, isso é feito de uma maneira bem diferente, é feito de uma maneira contínua e tudo isso que a gente comentou aqui (levantar requisitos, analisar, modelar, codificar e testar) também é feito nos métodos ágeis. A grande diferença é que é feito em ciclos, com intervalos muito menores. Isso resolve grande parte desses problemas. A cada ciclo desse, o cliente recebe um incremento no produto e ele já pode ver aquele produto funcionando e dar feedback pra equipe sobre aquele produto.

Então, nessa abordagem, os requisitos pouco claros acabavam criando uma série de problemas justamente por essa falta de conhecimento no início do projeto, de ter que se levantar tudo de antemão, com medo de ir pra próxima fase sem ter levantado tudo detalhadamente. Como não se podia voltar idealmente de uma etapa pra outra, tudo tinha que ser muito bem feito, muito bem levantado, e você acabava levando muito tempo.

A resistência a mudanças também era grande, justamente porque, como o escopo é fechado, se o cliente quisesse mudar alguma coisa, sempre tinha que ter uma negociação, um *change request* que muitas vezes ia aumentar o custo do cliente. Era complicado, era uma coisa difícil. O processo em si, o método em si, resistia à mudança. Ele não incentivava o cliente a mudar para que aquele sistema atendesse melhor ao negócio dele, mas pelo contrário, ele não queria que ele mudasse o negócio dele de maneira nenhuma. O que foi decidido no início tinha que ser mantido até o final.

Perceba que esse "até o final" geralmente era um ciclo bastante grande de tempo, então as mudanças iam se tornando cada vez mais caras. Os projetos eram longos demais, tinha pouquíssimo feedback do cliente ao longo do processo e não respondia às mudanças. E o mercado está mudando, o cliente está mudando de ideia, o mundo está mudando o tempo inteiro, e esse processo era pouco aderente a essas mudanças.

Os testes deixados para o final, como nós falamos, e como o [Standish Group] apresenta no Chaos report 50% desses requisitos eram implementados, então muita coisa acabava, muitas vezes nem dava tempo e acabava ficando de fora. Mas perceba que foi feito o levantamento de tudo isso detalhadamente lá no começo, e acaba que aquilo nunca foi implementado. O que é isso? É desperdício, é dinheiro, é tempo jogado fora. E 64% daquilo que foi implementado raramente era utilizado. Então era mais desperdício ainda. E com métodos ágeis a gente vai tentar justamente endereçar todos esses problemas e evitar que esses desperdícios aconteçam.

Vamos falar um pouquinho de métodos ágeis aqui. Eu gosto bastante de mostrar esse desenho do Jurgen no Management 3.0, que é um livro - tem também um treinamento, criado pelo holandês Jurgen Appelo. Ele

mostra esse desenho, que mostra o seguinte: lá em 1950, a gente tinha uma coisa chamada Programação de computadores, que era meio caótica, não tinha ordem, não tinha processo nenhum. Era uma coisa muito nova. Então, à medida que a computação foi evoluindo, houve a necessidade de se ordenar melhor o processo de se desenvolver software. E o que a gente conhecia muito bem eram as engenharias mais tradicionais, a engenharia civil, engenharia mecânica.

Trouxemos muito das ideias dessas engenharias para software. Chegamos à engenharia de software, que tentou deixar o software muito ordenado, assumindo que as coisas eram muito previsíveis. Por exemplo, você poderia tomar todas as decisões de antemão, e manter essas decisões num ciclo grande de anos. E os métodos ágeis são justamente o meio termo, que estaria na complexidade. Você tem uma certa previsibilidade, mas não uma previsibilidade total. Por isso, você trabalha em iterações em ciclos de feedback contínuo, pra que você possa rapidamente ir respondendo às mudanças que você não pode prever desde o início do projeto.

Esse foi o Manifesto ágil, que deu origem ao Manifesto Ágil. Um grupo de pessoas se reuniu numa estação de esqui - estão aqui os nomes das pessoas, são pessoas muito respeitadas na comunidade de desenvolvimento de software em geral, não só por competências em questões de processos, mas também por competências em desenvolvimento mesmo, em programação, em design, nas várias áreas de desenvolvimento de software. E eles disseram o seguinte:

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 a interação entre eles** (o relacionamento entre eles) mais do que processos e ferramentas. Então perceba que esse padrão vai ser seguido sempre de uma coisa em oposição à outra, mas aqui a questão é que não é que processos e ferramentas não são importantes - eles são importantes - mas as pessoas são mais importantes. Essa é que é a ideia, elas têm que ser colocadas em primeiro lugar.
O *Software em funcionamento** é mais importante que a documentação abrangente. O cliente não recebe valor diretamente pela documentação em si, mas ele recebe pelo software em funcionamento, que é o que ele precisa no fim das contas. Não é que nós não precisamos escrever documentação - muito pelo contrário, documentação que agrega valor tem que ser escrita -, o que nós não precisamos é daquela documentação que apenas vai ser um desperdício.
A **Colaboração com o cliente** é mais importante do que a negociação contratual. Não vale ter um contrato com um cliente e forçar para manter aquele contrato, aquele escopo daquela maneira fechada até o final, se o software que eu vou entregar para o cliente não vai atendê-lo, não vai servir de nada pra ele. Nós queremos construir software que agregue valor reealmente ao cliente, por isso nós temos que colaborar com o cliente.
**Responder a mudanças** mais que seguir um plano. Temos que estar em constante estado de inspeção, de adaptação, de responder às mudanças do mundo, às mudanças das necessidades do nosso cliente.

Então, basicamente, aqui a grosso modo, esse é o Manifesto Ágil. À medida que nós formos avançando no treinamento, nós vamos voltando pra essas ideias e vamos mostrando como que na prática o desenvolvimento ágil transforma esses princípios em realidade, transforma esses princípios em ação.

Além do Manifesto, nós temos mais 12 princípios ágeis:

- Satisfazer o cliente.

- Dar boas-vindas a mudanças.
- Entregar com frequência.
- Trabalhar como um time, não apenas um grupo de pessoas reunidas, mas um time, uma equipe de verdade.
- Motivar as pessoas.
- Comunicação face a face é preferida em relação a outros tipos de comunicação, como comunicação escrita, a troca de e-mail, telefone. Então a gente potencializa a comunicação face a face das pessoas.
- Medir o software em funcionamento.
- Manter um ritmo sustentável. Equipes exageradamente ágeis não são equipes que fazem milhares de horas extras, viram madrugadas, esse tipo de coisa, não. É um ritmo sustentável que possa ser mantido ao longo do tempo, com qualidade de vida pras pessoas. Isso é um princípio do desenvolvimento ágil.
- A qualidade tem que ser uma coisa bem vista, muito bem preparada, uma coisa muito importante também. Então aquela coisa de "deixa os testes para o final, faz se dar tempo" não acontece com métodos ágeis, os testes fazem parte do dia a dia, não é uma fase final.
- Manter as coisas sempre simples.
- Os designs têm que ser evolutivos. Então não é uma coisa também resolvida de antemão, é uma coisa que está sendo constantemente adaptada.
- Refletir regularmente sobre o processo, sobre o que se está fazendo, e analisar de que maneira nós podemos melhorar continuamente. Então, é a melhoria contínua aqui nessa ideia de refletir regularmente.

E aí, para implementar essas ideias todas, existem vários métodos ágeis, vários processos ágeis. Alguns desses processos já existiam até antes do manifesto. Aquelas pessoas que se reuniram no manifesto, algumas delas já eram responsáveis por esses métodos que seguem esses mesmos princípios. Esse é o guarda-chuva ágil.

Não existe uma única maneira de se utilizar Ágil, de se fazer Ágil. Na verdade, existem várias abordagens. A essência é a mesma, mas a forma pode mudar e aí você vai ter que, provavelmente, conhecer algum desses pra que você entenda qual deles vai se adaptar melhor às suas necessidades, ao seu contexto, sua equipe, seu time.

Nós vamos falar um pouquinho do Scrum, do Kanban e do XP, mas existem outros também aqui listados, que você pode pesquisar depois.

É importante ver esses métodos como ferramentas. Muitas vezes, na comunidade, as pessoas acabaram, a um tempo atrás, discutindo coisas como "Scrum é melhor que XP", "XP é melhor que Scrum", "Kanban é o

melhor". Na verdade, são métodos diferentes, são ferramentas diferentes. Então, dependendo da sua necessidade, você vai utilizar essa ferramenta de uma maneira diferente.

Métodos podem ser mais ou menos prescritivos. Então quanto mais prescritivo, mais coisas estão bem definidas no método. RUP é extremamente prescritivo - ele define vários papéis, define vários artefatos, várias práticas que têm que ser realizadas. São 120 no total. O XP só tem 13, o Scrum tem 9, o Kanban tem 3. Então aqui a grande questão é o seguinte: o Kanban, por exemplo, é um dos métodos mais adaptativos que existem, portanto, de acordo com a empresa que estiver utilizando kanban, provavelmente se você for analisar a realidade daquela empresa, o que elas fazem no dia a dia, elas têm menos coisas em comum do que parece, porque elas vão adaptar bastante o dia a dia delas, o processo delas. Já uma empresa que usa RUP vai ter muita coisa em comum.

A grande questão aqui é que quanto mais adaptativo o método for, mais ele pode ser aderente à realidade do seu contexto, porém mais difícil porque você vai ter que encontrar as suas próprias formas de fazer as coisas, para endereçar certos problemas.

Aqui a gente percebe que os métodos aqui vão variando dentro dessa linha de prescrição - de ser mais prescritivo ou menos prescritivo, e naturalmente, mais adaptativo. Qualquer ferramenta pode ser mal utilizada, então muitas vezes o insucesso de um projeto não é mérito da ferramenta, ou falta de mérito da ferramenta, a mesma coisa com o sucesso do projeto. Você pode estar utilizando, por exemplo uma serra elétrica como se ela fosse um machado, então você vai bater, por exemplo, a serra elétrica em uma árvore para tentar cortar como se fosse um machado, você vai quebrar a serra elétrica e não vai cortar a árvore.

É um exemplo que o Henrik Kniberg da comunidade ágil sempre dá. Se você usar a serra elétrica como uma serra elétrica, muito provavelmente você vai obter resultados melhores. Então nós podemos muitas vezes usar os métodos, a ferramenta certa para resolver o problema errado, ou a ferramenta errada para resolver um problema certo. Enfim, então muito cuidado com isso.

São ferramentas, você pode usar todas elas para resolver problemas diferentes. Uma não é melhor que a outra: elas são apenas ferramentas diferentes.