As possibilidades de criar um software são quase infinitas: é possível criar um programa de computador ou um aplicativo sobre quase qualquer coisa.

Se alguém quiser criar o software de um jogo que seja uma mistura de damas com xadrez, pode criar.

Se alguém quiser criar um software que calcule o tamanho de sua sombra de acordo com o movimento do sol, pode criar.

Se alguém quiser criar um software que calcule a velocidade média dos carros que passam na Avenida Paulista, pode criar.

Provavelmente os softwares dos exemplos acima não teriam nenhuma utilidade. Ninguém ia usar, não iam servir pra nada e seriam pura perda de tempo. E de dinheiro.

Mas eu dei esses exemplos escalafobéticos justamente pra mostrar que qualquer software pode ser desenvolvido. E justamente por isso, muitos software inúteis foram criados, gerando perdas milionárias.

Por que isso acontece?

Porque, historicamente, o desenvolvimento de software era baseado em um método conhecido como “cascata”.

O método cascata era baseado em etapas pré-definidas e sequenciais.

Vamos pegar o exemplo da figura acima.

Primeiro, se estuda um problema a ser resolvido. Depois, se analisa qual é a melhor forma de resolver esse problema e se desenvolve o projeto do software. Aí vem o desenvolvimento dos códigos, seguidos pelos testes e pela implantação do projeto.

Todo o calendário e orçamento dessas etapas são definidos antes da execução. E as etapas são sequenciais: uma só começa depois que a anterior termina.

Esse método cascata é muito eficiente em dar errado.

Porque ele prevê as etapas como se tudo fosse dar certo no caminho. Só que o caminho é incerto e tortuoso; vários problemas surgem.

Imagine, por exemplo, se o time muda após fechar o projeto.

Ou o prazo para fazer o código estoura (o que é bem comum).

Ou se descobre, na hora dos testes, que alguns códigos não estão funcionando.

Ou, na etapa da implantação, se percebe que o código desenvolvido tem uma incompatibilidade com o sistema.

Pior de tudo: imagine se todas as etapas foram completas e se descobre que o software é horrível, que não resolve o problema dos usuários.

A quantidade de softwares desenvolvidos que não deram certo é altíssima ao longo da história. E o método cascata quase sempre dá errado.

Foi por isso que algumas pessoas começaram a se perguntar se não haveria uma forma melhor de desenvolver software. Assim surgiram os métodos ágeis de desenvolvimento.

O método ágil mais famoso é o Scrum. Seus fundadores partiram de duas perguntas:

- por que não verificar, em intervalos regulares, se estamos no caminho certo e se é isso mesmo que o usuário precisa?

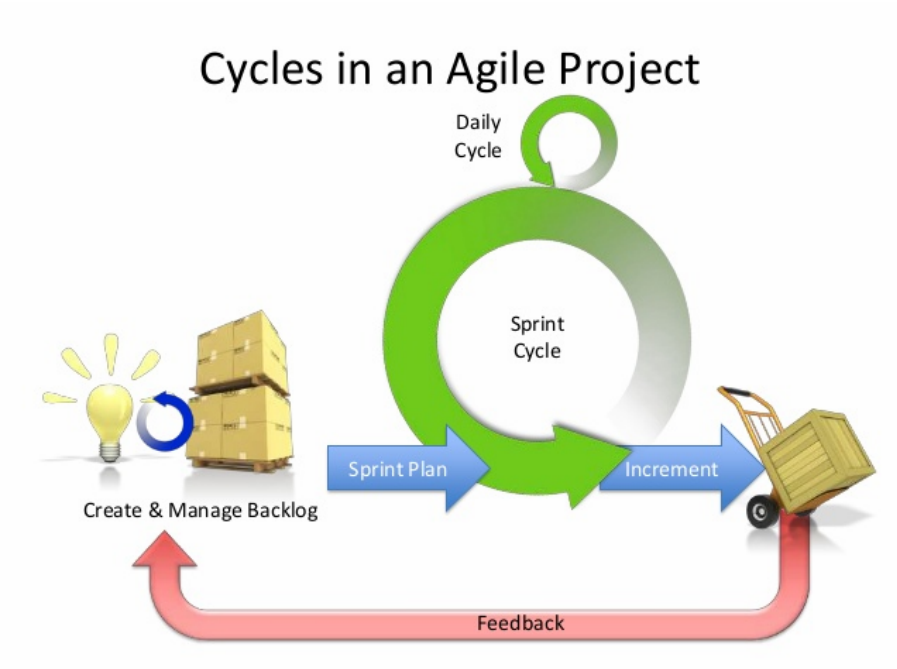
- por que não perguntar se é possível melhorar a forma de fazer ou o que está impedindo de fazer melhor e mais rápido?

Da primeira pergunta surgiu a ideia da inspeção. Da segunda pergunta surgiu a ideia da adaptação.

A inspeção serve pra melhorar o software; a adaptação, pra melhorar o desempenho do time de desenvolvimento.

Assim, o Scrum propõe que, ao invés de fazer tooodo o software de uma vez e lançá-lo para os usuários, seria melhor dividir o projeto em ciclos curtos e incrementais. Ao final de cada ciclo, um entregável que gera valor aos usuários é lançado.

Por exemplo: imagine um software cujo projeto previsse um ano para a finalização. Ao invés de esperar um ano pra entregar tudo, seriam entregues partes menores que já tivessem utilidade para usuários em tempos menores.



Esses tempos menores são chamados de sprint. Uma sprint pode ter de duas a quatro semanas de duração.

O objetivo de uma sprint é um entregável.

Scrum

O Scrum possui 3 elementos:

- Personagens
- Cerimônias
- Ferramentas

Personagens

Há 3 personagens no Scrum:

- Scrum Master
- Product Owner (PO)
- Time de desenvolvimento

O primeiro personagem é o Scrum Master. Ele é o responsável por garantir a aplicação do método e remover os impedimentos do Time de Desenvolvimento.

Para garantir a aplicação do método, o Scrum Master organiza os rituais e atualiza as ferramentas.

Para remover os impedimentos do Time de Desenvolvimento, o Scrum Master se vira nas 30: tem que ser pau pra toda obra.

Se a internet cair, o Scrum Master corre atrás pra fazer a internet voltar. Se o Time de Desenvolvimento precisar falar com o PO, o Scrum Master vai atrás do PO. Se um notebook der pau, o Scrum Master vai consertar ou arranjar outro notebook. Se acabar o café, o Scrum Master vai na padaria comprar café. Em resumo: se chover dentro do escritório, o Scrum Master arranja e segura o guarda-chuvas.

O Scrum Master é o servo do Time de Desenvolvimento: ele tem que resolver os problemas pra que o Time concentre-se no projeto.

O segundo personagem é o **Product Owner (PO)**. Como o nome já diz, ele é o dono do projeto, o responsável pelo produto e pelo backlog.

Assim, o PO coloca no backlog os itens do projeto e diz o que vai ser priorizado em cada sprint.

Depois das sprints, ele movimenta o backlog, indicando o que já foi feito e o que falta fazer.

O PO deve estar disponível para o Time de Desenvolvimento caso esse tenha alguma dúvida e precise de algum esclarecimento.

O terceiro personagem é o **Time de Desenvolvimento**. São os programadores que vão colocar a mão na massa e desenvolver o projeto de fato.

Cerimônias

Há 4 cerimônias no Scrum:

- Planning
- Daily
- Review
- Retro

A **planning** pode ser a planning do projeto ou de cada sprint.

A planning do projeto vai criar o backlog e determinar todos os entregáveis do projeto.

A planning da sprint vai definir o calendário da sprint. O Time de Desenvolvimento vai dizer o que pode ser entregue e vai estimar a duração de cada tarefa.

A **daily**, como o nome diz, é a reunião diária para verificar em que pé está o desenvolvimento do projeto.

Tem que ser rápida, no máximo 15 minutos. Normalmente, é feita com todo mundo em pé para simbolizar que vai ser rápida.

Na daily, cada membro do Time de Desenvolvimento responde a 3 perguntas: o que fez ontem, o que fará hoje e quais impedimentos tem encontrado.

Assim, todos ficam a par de tudo e o Scrum Master sabe que impedimentos tem que atacar.

A **review** e a **retro** ocorrem ao final da sprint.

Na review, o Time de Desenvolvimento apresenta ao PO tudo o que foi feito na sprint.

Na retro, os personagens vão discutir que problemas que enfrentaram na sprint e o que precisa ser melhorado para aumentar a velocidade do time.

Ferramentas

Há 2 ferramentas no Scrum

- Backlog
- Burn down ou burn up

O backlog é o quadro de tarefas do projeto, que mostra todos os itens que devem ser entregues pelo Time de Desenvolvimento.

Pode ser feito na parede com stickers ou em um software como o Trello.

A planilha de burn down ou de burn up mostra o progresso das tarefas na sprint.

A diferença é que o burn down mostra o que falta ser feito na sprint, enquanto que a planilha de burn up mostra o que já foi feito.

Exemplo

Vamos imaginar um projeto que desenvolva um programa para gerentes de um banco X.

O gerente precisa de um programa que permita-o atender seus clientes com eficiência.

Para isso, o programa tem que permitir ao gerente ver algumas informações do cliente: os dados cadastrais, o histórico de compras, os investimentos, a fatura do mês do cartão, o limite de crédito e os produtos disponíveis para compra.

Agora vamos comparar como esse programa seria desenvolvido no método cascata e no método ágil.

Cascata

1 mês para estudos e análises do escopo do projeto
1 mês para definição do projeto
8 meses para codificação
1 mês de testes
1 mês para implantação

No primeiro mês, analistas estudariam a demanda dos gerentes, os problemas e as possibilidades de solução.

No segundo mês, um arquiteto e um engenheiro de software desenhariam o projeto, com cronograma, orçamento e time.

Do terceiro ao décimo mês, o time de desenvolvimento desenvolveria o programa.

No décimo primeiro mês, os testers testariam.

No décimo segundo mês, o time de desenvolvimento integraria o programa com o sistema do banco.

Esse cronograma baseia-se no fato de que tudo vai dar certo (mesmo que todo mundo saiba que vai dar errado e o prazo não será cumprido).

Ao final do ano, será entregue o software completo (quá, quá, quá).

Ágil

O ponto de partida é a planning pra montar o backlog.

O backlog teria:

- dados cadastrais
- histórico de compras
- investimentos
- fatura do mês do cartão
- limite de crédito
- produtos disponíveis para compra

O PO, então, decidiria o que é mais prioritário. No caso, vamos supor que o gerente precisa confirmar os dados cadastrais logo no início da reunião com o cliente para evitar fraudes.

Assim, a primeira sprint criaria um programa que mostraria os dados do cliente na tela do gerente. Como uma sprint vai de 2 a 4 semanas, no primeiro mês o gerente (usuário) já teria algo que melhorasse o seu trabalho.

A sprint começaria com a planning. Nesta reunião, o time de desenvolvimento diria o que tem que ser feito para entregar esse programa com os dados cadastrais do cliente.

Digamos que o programa tenha 4 funcionalidades:

1. Se conectar com a página do gerente
2. Mostrar um campo onde o gerente digita o CPF do cliente
3. Localizar esse CPF no banco de dados do banco
4. Trazer os dados cadastrais daquele CPF até a página do gerente

O desenvolvedor A criaria os códigos do item 1 e 4, porque ele cuida da conexão do programa com a página do gerente.

O desenvolvedor B criaria os códigos do item 2, porque ele cuida da alteração na página do gerente.

O desenvolvedor C criaria os códigos do item 3, porque ele cuida da conexão do programa com o banco de dados do banco.

Aí cada um “quebraria” suas tarefas pelos dias da sprint. Vamos supor que a sprint tenha 3 semanas e 15 dias. Então, cada desenvolvedor quebraria sua grande tarefa (item 1, 2, 3 ou 4) em 15 tarefas diárias. (O ideal é quebrar em 30, pra que cada tarefinha dure um turno).

Na medida em que os desenvolvedores entreguem suas tarefas, o Scrum Master vai atualizando a planilha de burn down ou burn up, para que todos saibam se a sprint está atrasada, adiantada ou no prazo certo.

Todo dia, no início da manhã, os personagens participam da daily, onde os desenvolvedores dizem o que fizeram no dia anterior, o que farão no dia e quais impedimentos estão encontrando.

Ao final da sprint, os personagens apresentam ao PO review tudo o que foi feito naquela sprint.

Depois, na retro, os personagens discutem quais problemas encontraram e o que precisa melhorar. Isso é fundamental para o progresso do time e para aumentar a capacidade e eficiência da equipe na próxima sprint.

Os problemas apontados na retro podem ser os mais diversos: dificuldade em obter chaves de acesso ao sistema do banco, daily muito longa, erro na estimativa de duração das tarefas, desequilíbrio na divisão de tarefas (um desenvolvedor com muita tarefa e outro com pouca), dificuldade em conversar com o PO etc.

O importante aqui é listar todos os problemas, elencar os piores e focar em corrigi-los na próxima sprint.

Dessa forma, a próxima sprint terá menos problemas e o time vai caminhar mais rápido.

Terminada a primeira sprint, o PO diz que a segunda priorizará os produtos disponíveis para compra. (Porque o gerente precisa vender!).

Aí o time de desenvolvimento vai incrementar o programa de forma a permitir que o gerente tenha em sua tela os produtos que poderiam ser comprados por aquele cliente que está em sua frente.

E assim iria todo o projeto, de sprint em sprint.

Na medida em que o gerente use o programa, ele vai informando ao PO o que funciona bem e o que poderia melhorar. Com essas informações, o PO pode rearrumar o backlog pra incluir uma funcionalidade ou retirar outra. Ou pode propor alterações em um item.

O que importa aqui é que o progresso em ciclos curtos permite a) incrementar o projeto (inspeção) e b) aumentar a performance do time (adaptação).