



# Desenvolvimento de Sistemas

---

## Metodologias de desenvolvimento de *software* Introdução e aplicabilidade

O desenvolvimento de um projeto de *software* exige a parceria de diversas áreas de conhecimento. Por meio dessa necessidade, é comum que se tenha uma equipe de pessoas trabalhando em conjunto.

Sendo assim, como projetar e acompanhar o desenvolvimento de um projeto de *software*, para que este seja concluído com sucesso?

Para iniciar seus estudos sobre metodologias de desenvolvimento de *software*, conheça um ditado muito popular no setor de tecnologia da informação: “Na tecnologia da informação, não existe bala de prata”. Em outras palavras, não existe uma solução que atenda a todos os tipos de projetos e necessidades. Logo, é essencial conhecer as diversas estratégias existentes, seus pontos fortes e seus pontos fracos, para, dessa forma, entender a melhor abordagem para os diversos cenários que você encontrará no dia a dia de um projeto de *software*.

## Metodologias tradicionais

### Modelo cascata



Figura 1 - Exemplificação do modelo cascata

Fonte: <<https://www.treinaweb.com.br/blog/ciclo-de-vida-software-por-que-e-importante-saber>>. Acesso em: 11 out. 2021.

O modelo cascata, também conhecido como modelo Waterfall, trabalha com etapas predeterminadas, executadas sequencialmente. Dessa forma, é necessário finalizar todas as atividades de uma etapa para depois seguir para a próxima. As etapas, já na ordem certa, são: levantamento de requisitos, projeto, implementação, realização de testes e manutenção do sistema.

## Levantamento de requisitos

Nessa etapa, a equipe de desenvolvimento definirá, junto ao cliente, as expectativas para o projeto, montando o escopo de requisitos. É fundamental a excelência nesse momento, para que o projeto cumpra com as expectativas do cliente, sendo um projeto de sucesso.

Como visto anteriormente, serão levantados os requisitos de um projeto em requisitos funcionais e não funcionais. O ponto-chave aqui é não tentar apenas identificar o que está dentro do escopo do projeto, mas, além disso, entender o que

**não** está dentro do escopo. Dessa forma, será possível manter uma linha mais próxima das expectativas do cliente para a entrega do projeto.

## Projeto

Com a lista de requisitos em mãos, você entrará na segunda etapa: o projeto para desenvolvimento do *software*. É nesse momento que você planejará como serão as próximas etapas, criando o cronograma do desenvolvimento do projeto, a definição escrita do escopo, a estimativa para cada etapa, a montagem do time de desenvolvimento e, em alguns casos, a modelagem da arquitetura, dos protótipos e das estruturas para o *software*.

Entenda essa etapa como a documentação de tudo o que está por vir. O resultado da etapa de projeto será o mapa para o desenvolvimento do *software* e também a sua principal comunicação com o cliente.

## Implementação

A etapa de implementação é a etapa de desenvolvimento do *software*. Baseado nos requisitos e nas atividades montadas na etapa de projeto, a equipe desenvolverá todo o projeto. A duração dessa fase está diretamente ligada à quantidade de pessoas na equipe e o conhecimento técnico desta.

## Realização de testes

Com o código totalmente desenvolvido, a equipe testará o *software*, validando o resultado da etapa de implementação e garantindo que o desenvolvimento tenha cumprido o objetivo do projeto.

Quaisquer falhas encontradas nesse momento deverão ser corrigidas pela equipe de desenvolvimento antes de partir para a próxima etapa.

## Manutenção do sistema

A etapa de manutenção do sistema envolve a implantação do *software* para o cliente, sendo definida como a entrega do projeto e, em alguns casos, a manutenção contínua. Entretanto, no modelo cascata não há espaço para novos requisitos no escopo, sendo necessário o reinício desse modelo. Nesse modelo, a manutenção reflete apenas suportar aquilo que já foi desenvolvido e entregue.

## Metodologias ágeis

### Conceitos básicos

As metodologias ágeis surgem como uma proposta de atender com flexibilidade e agilidade às características do mercado corporativo nos dias de hoje. Com a competitividade do mercado exigindo das empresas agilidade para responder a mudanças, as metodologias ágeis buscam prover um processo estruturado e confiável para reagir rapidamente a mudanças de escopo e redirecionamentos nas estratégias.

Quando se fala do desenvolvimento de um projeto de *software*, é natural que se planeje uma lista de atividades que compõem a solução em desenvolvimento.

Com a conclusão dessa lista, conclui-se o projeto.

Considerando um projeto cujo desenvolvimento levará cerca de um ano, como se pode garantir que o escopo inicial ainda é válido para o cliente após esse longo período?

Ainda há outro cenário que pode ser analisado: e se algum dos requisitos solicitados pelo cliente não ficou plenamente entendido entre todos os envolvidos?

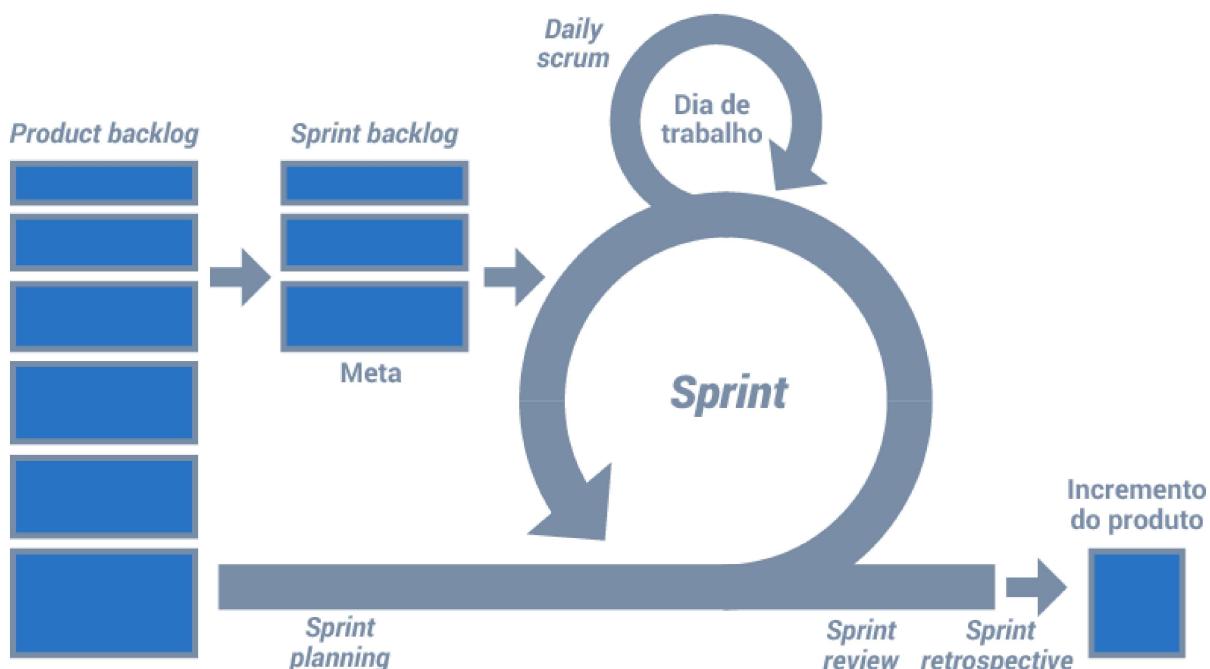
Isso se torna ainda mais grave se for o caso de um requisito que sirva como base para outros requisitos (uma dependência). Nesse cenário, o erro só seria descoberto após a entrega do projeto (ou seja, um ano depois). Todas as partes impactadas pelo “requisito incompreendido” também precisam de modificações.

Descobrir problemas em projetos com uma única entrega, em um longo período de tempo, é um grande risco para o sucesso dos projetos. Baseadas nesses problemas, as metodologias ágeis trabalham com o conceito de **incrementos**. Com pequenas evoluções e pequenas entregas para o cliente, é possível avaliar se o desenvolvimento do projeto está na direção correta e, caso contrário, redirecioná-lo.

Mas as metodologias ágeis (sim, no plural) não se resumem a isso. Elas têm suas diferenças e peculiaridades. Por isso, é preciso abordar duas metodologias que são amplamente utilizadas no mercado de trabalho e conhecer os seus processos.

Você verá que muitos termos são compartilhados entre as diferentes metodologias, o que facilitará seu aprendizado. Em muitos casos, apenas haverá uma aplicação diferente para o mesmo conceito.

## Scrum



### Figura 2 – Exemplificação do processo *scrum*

Fonte: <<https://kbase.com.br/2018/11/05/scrum-vantagens-da-metodologia-agil-no-desenvolvimento-de-software/>>. Acesso em: 5 out. 2021.

O *scrum* é uma das metodologias mais utilizadas no mercado de trabalho. Com grande popularidade, é essencial conhecer o seu processo e seus pontos fortes fracos. Observe a seguir os artefatos, os papéis e as cerimônias do *scrum*.

## Artefatos do *scrum*

### *Milestone*

Um *milestone* é um agrupamento de atividades similares. Esse agrupamento pode ocorrer por diversos critérios, de acordo com as necessidades do projeto.

Você pode agrupar atividades relativas a um mesmo “módulo” do seu projeto. Nesse critério, por exemplo, você teria um *milestone* chamado de “módulo de autenticação de usuários” e agruparia todas as atividades relativas à autenticação de usuários.

Você também pode agrupar atividades por sua entrega, o que é muito comum em projetos que tenham uma etapa de pesquisa e validação de conceitos. Nesse contexto, o *milestone* seria o agrupamento necessário para conseguir finalizar a entrega. Por exemplo, caso o seu projeto seja uma nova tecnologia para uma plataforma de *streaming* de vídeos e filmes, nunca usada por nenhuma outra empresa, é comum que você faça uma etapa de validação desse conceito, isto é, um pequeno projeto rápido, não relativo ao seu produto final, desenvolvendo apenas o que é necessário para comprovar a viabilidade do projeto. Nesse cenário, é possível ter um *milestone* “Validação – Streaming Inovador” e agrupar, nesse *milestone*, todas as atividades relacionadas ao desenvolvimento dessa validação.

Nesse tipo de projeto, validar um conceito é conhecido como “prova de conceito” e utiliza-se a sigla POC (originada do termo em inglês *proof of concept*, que, traduzindo do inglês, significa “prova de conceito”).

## User story

Uma *user story* é um artefato utilizado para descrever objetivos e funcionalidades de um *software*. Nesse artefato, você responderá “para quem?”, “o quê?” e “por quê?”. Respondendo a essas três perguntas, você conseguirá direcionar o desenvolvimento das atividades, sem definir como elas devem ser realizadas (dando flexibilidade para a equipe de desenvolvimento do projeto).

Originalmente, as *user stories* devem ser escritas em primeira pessoa, assumindo o papel do ator da *user story* (nesse caso, refere-se a quem foi a resposta para a pergunta “para quem?”). Veja um exemplo de *user story* a seguir:

**Eu, como um** médico,  
**Quero** conseguir registrar os laudos que preenchi  
**De modo que** consiga acessá-los depois fácil e rapidamente.

Essa flexibilidade para desempenhar a atividade pode ser um ponto negativo para algumas equipes, parecendo pouco descritivas ao desenvolvimento. Existem diversas outras práticas e conceitos que podem ser utilizados para a escrita de uma atividade. Algumas delas são:



- ◆ Critérios de pronto: uma lista apresentando o que deve existir para que a atividade seja considerada como “pronta” para ser testada e preparada para a entrega.
- ◆ Critérios de aceite: uma lista apresentando o que deve existir para que a atividade seja aceita como concluída e aprovada e autorizando a fazer parte do incremento a ser entregue.
- ◆ Expectativas: uma lista apresentando quais as expectativas a respeito da atividade. Quando a atividade for revisada, ela deverá atender a todos os itens da lista de expectativas.
- ◆ Objetivo: um parágrafo descrevendo qual é o objetivo (e, logo, o valor para o ator dessa atividade) de desenvolver a atividade.

Existem diversas práticas, estratégias e conceitos a respeito da melhor forma de descrever uma atividade. Como foi citado, não existe bala de prata para esse tópico também. Assim como em todo o objetivo das metodologias ágeis, entender a real necessidade do seu time de desenvolvimento e adaptar-se a essa realidade é essencial. A comunicação e a decisão em equipe são parte do processo.

## *Product backlog*

O *product backlog* é a nossa lista total de tarefas para o completo desenvolvimento da solução. A partir do *product backlog*, as atividades serão selecionadas para desenvolvimento e entregues por meio dos incrementos construídos. Dentro do que você já leu até agora, deve ter entendido as tarefas como uma lista de *user stories*.

Idealmente, você deve mapear todas as atividades necessárias para a conclusão do seu projeto. Porém, isso nem sempre é aplicável (e, constantemente, são encontrados pequenos novos refinamentos que são mapeados durante o desenvolvimento do projeto). Dessa forma, deve-se sempre tentar deixar o *product backlog* o mais completo e bem preenchido possível e disponível para utilização pelo time de desenvolvimento do projeto.

Dito isso, considere que o *product backlog* deve estar sempre ordenado por prioridade. A definição de prioridade, entretanto, é subjetiva. Mais uma vez, deve-se lembrar de que não existe “bala de prata” no desenvolvimento de projetos de *software*. Identificar ao seu cliente o que é prioridade para o sucesso do projeto é essencial.

O *product backlog* é responsabilidade do *product owner*. Você conhecerá mais sobre esse papel em seguida.

## *Sprint*

O *sprint* é uma parte essencial da maioria das metodologias ágeis. Como já citado em diversos momentos, a metodologia ágil busca entregar incrementalmente um projeto de *software*, de modo que a equipe de desenvolvimento consiga redirecionar o projeto de acordo com as expectativas do cliente. Para se ter incrementos, são necessários diversos inícios, meios e fins. Para isso existem as *sprints*.

Entenda a *sprint* como uma janela de tempo, na qual a equipe planejará quais atividades serão feitas e qual é o objetivo desse incremento, para que, tendo isso em mente, você possa desenvolver as atividades. Com o fim dessa janela de tempo, a equipe revisará os itens desenvolvidos, verificará se o incremento foi um sucesso e, juntos, identificará possíveis melhorias para otimizar a eficiência de desenvolvimento da equipe. Todo esse processo é uma *sprint*.

As *sprints* são iterativas, isso é, um processo contínuo. Ao encerrar uma *sprint*, outra é iniciada. A duração de uma *sprint* também é algo variável de acordo com as necessidades do projeto. O *scrum*, entretanto, recomenda que *sprints* tenham duração entre duas e quatro semanas.

*Sprints* são partes essenciais do *scrum*, pois, por meio delas, é possível definir os incrementos que são entregues. Veja alguns detalhes importantes sobre as *sprints* ao conhecer as cerimônias do *scrum* em seguida.

## Sprint backlog

Durante a abordagem sobre as *sprints*, citou-se um pequeno exemplo no qual o primeiro passo para a existência de uma *sprint* é a definição de quais atividades serão feitas. Essa lista de atividades selecionadas para a *sprint* é o *sprint backlog*.

Imagine um processo com etapas. Para o *scrum*, o desenvolvimento de uma atividade contém um processo em etapas. Que tal fazer um cenário de exemplo muito minimalista?

Primeiro, haveria o *product backlog*, no qual, como visto, teria uma lista de atividades para conclusão do projeto ordenadas por prioridade.

Logo em seguida, haveria o *sprint backlog*. Por meio das atividades do *product backlog*, são escolhidas as mais prioritárias e enviadas para o *sprint backlog*. O número de atividades escolhidas depende do tempo de duração da *sprint* e do ritmo de desenvolvimento da equipe.

Durante o andamento da *sprint*, seriam desenvolvidas as atividades listadas no *sprint backlog*. Dessa forma, não há mais o interesse pelas atividades listadas no *product backlog*, e sim apenas pela *sprint*, suas atividades (*sprint backlog*) e seu objetivo.

Com o desenvolvimento e a conclusão dessas atividades e também da *sprint*, recomeçaria este processo, escolhendo-se novas atividades (as mais prioritárias) do *product backlog* e movendo-as para o *sprint backlog*, com um novo objetivo em mente para a nova *sprint*. Essa é a diferença do seu *sprint backlog* para o seu *product backlog*.

## Papéis do scrum

### Scrum master

O *scrum master* é o líder da equipe *scrum*. Responsável pela execução dos processos e a resolução de bloqueios, o *scrum master* deve representar o time *scrum*, defendendo o projeto e orientando a equipe.

A atuação diária do *scrum master* pode variar de acordo com o ambiente de trabalho. Algumas empresas utilizam o *scrum master* como um gerente de projetos, enquanto, em outros lugares, o *scrum master* atua como um *coach*.

Independentemente de qual sejam as tarefas do dia a dia do *scrum master* é essencial que ele tenha boas habilidades comunicativas e gerenciais, conheça bem o *scrum* e tenha um bom relacionamento com a sua equipe.

## *Product owner*

Traduzindo do inglês, o *product owner* pode ser entendido como o “dono do produto”. Também conhecido como PO, ele não necessariamente precisa fazer parte da empresa responsável pelo desenvolvimento do *software*, mas fará parte do time *scrum*. Inclusive, o *product owner* não precisa sequer saber programar, mas deve conhecer o projeto, os objetivos do projeto, os *stakeholders* (assunto que será abordado em seguida) e os usuários.

Com esse conhecimento sobre a solução, o *product owner* é o responsável pelo *product backlog* e suas atividades. O PO preencherá as atividades, ordenará o *product backlog* e validará o andamento do desenvolvimento do projeto. Assim, entende-se que o PO é quem aprova uma atividade entregue e quem delega novas atividades.

Em algumas empresas de desenvolvimento de *software*, cuja atuação é prestar serviço para empresas de outros ramos, é comum que o *product owner* seja um funcionário do cliente do projeto. Também pode-se encontrar times nos quais um dos próprios membros da equipe de desenvolvimento atua como *product owner*. Esse tipo de definição varia de acordo com a realidade dos projetos e das empresas. De modo geral, o importante é que o *product owner* tenha uma boa visão para gerenciamento de escopo e projeto, como o controle de riscos, de cronograma e de evolução.

## Time de desenvolvimento

O time de desenvolvimento é a equipe de desenvolvedores, *designers*, testadores e qualquer outro profissional que desenvolverá o projeto no dia a dia. O time de desenvolvimento seguirá os processos *scrum*, realizando as cerimônias (o que você verá em seguida) em prol da conclusão do projeto agilmente.

## *Stakeholders*

Os *stakeholders* são pessoas que são impactadas ou têm interesse pelo projeto. Dessa forma, pode-se entender os *stakeholders* como os principais usuários (ou, em alguns casos, mantenedores) do projeto.

## Cerimônias do *scrum*

### *Planning*

Considere novamente aquele pequeno exemplo citado no tópico sobre *sprint backlog*. Você se lembra que o primeiro passo para se ter uma *sprint* era planejar as atividades que iriam para o *sprint backlog*? **Isso é feito pelo *planning*.**

O *planning*, também chamado de “planejamento”, é uma reunião na qual o time de desenvolvimento, o *scrum master* e o *product owner*, juntos, considerarão o *product backlog*, escolherão as atividades mais prioritárias e farão sobre elas as estimativas, de modo que seja possível montar um *sprint backlog* com a quantidade de demanda certa para a duração da *sprint* e o tamanho da equipe. Ao final do *planning*, o time *scrum* terá o *sprint backlog* preenchido e estimado e um objetivo declarado para a *sprint*.

Essa cerimônia conta com um tempo máximo de acordo com a duração total da *sprint*, sendo duas horas para cada semana de *sprint*. Dessa forma, se o time planeja trabalhar com *sprints* de um mês, o *planning* terá um tempo máximo de oito horas. Na prática, os times ágeis buscam sempre ser os mais eficientes possíveis em suas reuniões, de modo que o processo de desenvolvimento seja pouco burocrático. O ideal é o time identificar o tempo necessário para um bom planejamento, a fim de que as *sprints* tenham bons resultados e as reuniões sejam eficientes.

Um dos maiores desafios do planejamento é estimar uma *user story*, principalmente para times novos, que trabalham juntos há pouco tempo. Aliás, antes de se falar sobre como estimar uma *user story*, ou uma atividade, entenda o que é necessário para se fazer uma boa estimativa.

O principal objetivo de cada *user story*, ou atividade, do *product backlog* é documentar claramente um requisito para o time de desenvolvimento. Ou seja, pretende-se conseguir, de modo simples e claro, descrever o que é preciso ser desenvolvido e entregue. Para fazer isso, existem diversas estratégias, como algumas que você viu no tópico sobre a *user story*. Porém, descrever bem uma *story* não é o único cuidado que você deve ter.

Uma *user story* é uma forma de descrever objetivos para o *software*. Isso pode significar uma integração completa com um *software* terceiro ou até mesmo um módulo completamente novo para um *software*. Nesses cenários, trata-se de uma atividade longa e que exige um alto nível de esforço e complexidade. Considerando a duração da *sprint* e o tamanho da equipe, é muito provável que se tenha uma única *user story* na *sprint* – o que seria extremamente improutivo e difícil de gerenciar. O que fazer se isso acontecer?

É muito comum que o time de desenvolvimento separe (ou, como é comumente dito, “quebre”) uma *user story* em diversas atividades diferentes. Esse processo pode ser feito dentro e fora do *planning* e, idealmente, é o *product owner* que deverá

interagir com o time de desenvolvimento para que ambos gerenciem o estado atual do desenvolvimento do projeto e da *user story*. Inclusive, é comum que não seja possível entregar uma *user story* em uma *sprint* – e, nesses momentos, vale a pena conferir se essa *user story* não tem potencial para ser um *milestone*.

As estratégias listadas para o bom preenchimento de uma *user story* também podem ser utilizadas para o preenchimento de atividades “filhas” de uma *user story*. Além disso, também existem estratégias para auxiliar no correto desmembramento de uma *user story*, como é o caso das metas SMART (específica, mensurável, atingível, relevante, temporal). Utilizando o seu buscador favorito, procure por “metas SMART” e conheça mais sobre o assunto.

Até aqui, você já tem uma lista de atividades, corretamente “quebradas”, de modo que é possível ter um fluxo de desenvolvimento contínuo, com uma boa rotação de atividades – isso é, sem que uma mesma atividade fique em desenvolvimento durante todo o *sprint*. Mas como saber se o *sprint backlog* não está com mais atividades do que as que o time consegue desenvolver em uma *sprint*? Ou menos? Para responder a essa pergunta, é **preciso estimar as suas atividades**.

## *Planning poker*



Figura 3 – Exemplificação de uma reunião com *planning poker*

Fonte: <<https://www.metodoagil.com/planning-poker/>>. Acesso em: 7 out. 2021.

A imagem apresenta três personagens do filme Star Wars sentados a uma mesa, segurando cartas com números diferentes, representando uma rodada de planning poker.

Uma das práticas mais famosas para estimar uma atividade é conhecida como *planning poker*. Inclusive, existem diversas ferramentas para auxiliar na execução do *planning poker*, como baralhos de cartas, plataformas *on-line*, extensões para ferramentas de gestão (como o Trello), entre outros.

O objetivo da técnica *planning poker* é auxiliar uma equipe a encontrar um consenso sobre a estimativa de atividades durante um planejamento. Por meio do requisito apresentado, é função do time de desenvolvimento estimar o esforço necessário para a sua conclusão. É nesse momento que a técnica *planning poker* é utilizada.

Originalmente, utiliza-se um baralho de cartas do *planning poker*, as quais devem conter os valores 1, 2, 3, 5, 7, 10 e Infinito (este último com a função de solicitar que o requisito seja separado em requisitos menores). Esses valores podem ser entendidos como números sem unidade, representando apenas uma sequência ordenada de tamanhos, ou um “dia ideal” de programação. A ideia da limitação dos valores deve-se à afirmação de que quanto maior a estimativa, menor a precisão.

Perceba que existe subjetividade no quanto vale cada número das cartas no *planning poker*. Para equipes que nunca estimaram atividades antes, uma boa prática é utilizar, no primeiro *planning*, horas (para estimar as atividades). A partir da segunda *sprint*, a equipe já pode olhar para a *sprint* anterior e agrupar as horas das atividades estimadas em pontos das cartas do *planning poker*. Com o resultado da primeira *sprint*, o time já conseguirá identificar o que é uma atividade que exige o mínimo de esforço (um 1) e o que é uma atividade que está muito complexa (um 10).

Para cada atividade a ser estimada no *planning*, e com cada pessoa do time de desenvolvimento tendo as cartas em suas mãos (talvez virtualmente), cada membro escolhe uma carta para representar a estimativa de esforço para o desenvolvimento da atividade apresentada. Assim que todos os membros do time tiverem decidido a sua carta, apresenta-se para todos o valor escolhido.

Com todas as cartas “na mesa”, caso exista um consenso no valor escolhido, registra-se a estimativa escolhida na própria atividade e inicia-se uma nova rodada (isto é, o próximo requisito é lido e o processo do *planning poker* é reiniciado). Caso não exista um consenso, é aberta uma rodada de discussão para que os membros do time de desenvolvimento apresentem as suas motivações para as escolhas e cheguem a um consenso.

De modo geral, a técnica *planning poker* é uma abordagem que busca encontrar consenso em estimativas de uma maneira ágil e fácil. A técnica impacta positivamente na velocidade em estimar requisitos, encontrando consenso rapidamente e sem a necessidade de discussões.

## Daily

A *daily*, também conhecida como *stand-up daily meeting* (reunião diária em pé), é uma reunião que ocorre todos os dias, de curta duração, e na qual todos os membros do time de desenvolvimento devem informar um breve *status* sobre as atividades que estão realizando e se têm algum bloqueio no seu fluxo de trabalho, informando se precisam de algum apoio ou orientação sobre esse bloqueio.

A *daily* deve durar entre 15 e 30 minutos e, na grande maioria dos lugares, só é realizada pelo time de desenvolvimento, acompanhado pelo *scrum master*.

Já o termo *stand-up daily meeting* surgiu por conta de que diversas empresas realizam a reunião em pé, uma vez que deve ser uma reunião objetiva e de curta duração.

Existe uma estratégia, utilizada principalmente para apoiar as equipes novas no *scrum*, para auxiliar no processo da *daily*. Nessa estratégia, cada membro do time de desenvolvimento deverá responder a três perguntas para concluir a sua parte da *daily*: “o que eu fiz ontem?”, “o que eu vou fazer hoje?” e “tenho algum bloqueio?”.

## *Sprint review*

A *sprint review* é uma reunião para apresentar os resultados da *sprint*, contando com a participação do time de desenvolvimento, do *scrum master* e do *product owner*. O time de desenvolvimento, junto ao *scrum master* apresentará ao *product owner* o resultado de cada atividade do *sprint backlog*, sendo papel do *product owner* aprovar a entrega e trazer *feedbacks* para a equipe.

Essa reunião deve ser realizada ao final da *sprint* e, nesse momento, todas as atividades já devem ter sido finalizadas.

Ao final da apresentação, o time verificará o objetivo anotado para a *sprint* e, de acordo com o decorrer da *sprint review*, o time registrará se o objetivo foi alcançado com sucesso ou não.

## Sprint retrospective

A *sprint retrospective*, também conhecida como “retro” ou “retrospectiva”, é uma reunião para o time de desenvolvimento, junto ao *scrum master*, analisar a última *sprint* realizada e debater sobre potenciais pontos de melhoria para otimizar os processos da equipe.

A retro pode ser realizada de diversas maneiras. Algumas equipes preferem realizar um bate-papo aberto, no qual todos os membros da equipe conversam, enquanto um responsável registra, em algum lugar, os pontos debatidos. Já em outras equipes, cada um dos membros da equipe tem um momento de fala para apresentar os pontos pensados e a equipe debate ponto a ponto.

Independentemente de qual seja o formato escolhido para a retro, é importante que o time tenha um bom ambiente, propício a ter liberdade na comunicação. Também é essencial registrar os pontos debatidos, levantando pontos de ação para resolver os problemas encontrados, otimizando, *sprint a sprint*, os processos da equipe.

Existem diversas ferramentas *on-line* para auxiliar no processo de realização da retro. Utilizando o seu buscador favorito, procure por “metro retro” e acesse a página “Metro Retro: Free Agile Retrospective Tool” para conhecer uma delas.

Conhecendo os artefatos, os papéis e as cerimônias do *scrum*, é possível ter uma base sólida para participar e orientar na execução do *scrum* em projetos de desenvolvimento de *software*. Assim como a metodologia é ágil como um todo, o *scrum* busca trazer praticidade e agilidade, diminuindo burocracias nos processos da equipe.

De modo geral, entende-se que o *scrum* é composto de *sprints*, sendo essas entregas incrementais, e tem-se, para cada *sprint*, um processo de desenvolvimento que começa pelo planejamento das atividades a serem realizadas (*planning*), depois há o desenvolvimento das atividades acompanhado diariamente pela *daily*, e, então, é concluída a *sprint* realizando a *sprint review*, na qual são apresentados os resultados

ao *product owner*. Por último, na *sprint retrospective*, a equipe olha para a sua última *sprint* e debate sobre potenciais pontos de melhorias. Com a realização desse processo, existe um fluxo ágil e eficiente para entregar *software* de acordo com as expectativas do seu cliente.

## Kanban

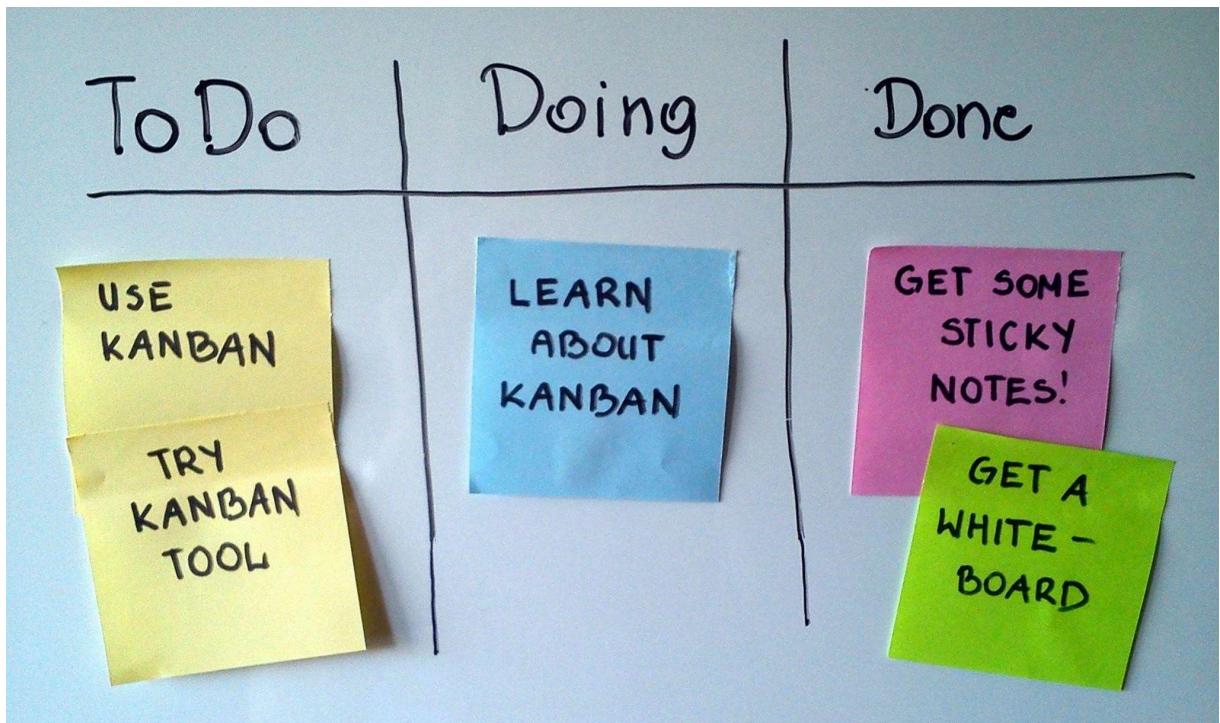


Figura 4 – Exemplificação de um quadro *kanban*

Fonte: <<https://enotas.com.br/blog/kanban/>>. Acesso em: 11 out. 2021.

A imagem contém uma tabela com três colunas: “To do”, “Doing” e “Done”, representando os estados “Para fazer”, “Fazendo” e “Feito” do kanban. Dentro de cada coluna, existem notas autoadesivas, representando atividades.

O *kanban* usa muitos dos mesmos princípios vistos no *scrum*. É uma metodologia ágil e, por isso, tem o objetivo de otimizar o processo de desenvolvimento de *software*.

Diferentemente do *scrum*, o *kanban* não trabalha com *sprints*, trabalha com um quadro para representar o fluxo de desenvolvimento. Esse quadro contém, no mínimo, três colunas: “para fazer”, “fazendo” e “feito”. Nesse cenário sem *sprints*, não consta também o *sprint backlog*. Logo, entenda que a coluna “para fazer” é igual ao *product backlog*, visto no *scrum*.

Sem as *sprints*, você pode se perguntar: “então o *kanban* não possui o *planning*, a *sprint review* e a *sprint retrospective*? ”.

A resposta curta e rápida é “sim”. Porém, entenda que existem alguns detalhes por trás.

Um projeto de *software* desenvolvido por meio de metodologias ágeis envolve entrega de produto (os “incrementos de *software*”) e alinhamento de expectativas. Se você simplesmente definir um escopo para um *software* e desenvolver item a item, movendo suas atividades do “para fazer” para o “feito”, terá, ao final, uma metodologia tradicional de trabalho. Nesse sentido, para que o *kanban* atenda às necessidades de uma metodologia ágil, é preciso implementar outros conceitos.

Muitas equipes que utilizam o *kanban* automatizam o seu processo de entregar versões novas do *software*, então todas as atividades que estão concluídas podem ser entregues ao cliente. Ainda pensando nisso, pode-se enxergar um risco em entregar tudo diretamente ao cliente. Logo, é comum que equipes que utilizem o *kanban* tenham mais do que as três colunas vistas anteriormente. É possível adicionar “em revisão”, “em homologação”, entre outras colunas, aumentando o processo de uma atividade até que ela seja considerada pronta para ser entregue.

Além disso, automatizar entregas contínuas de *software* é uma tarefa que exige um conhecimento específico. Conhecimento esse que nem sempre faz parte da equipe de desenvolvimento. Também existem muitas equipes que criam processos manuais para revisar e publicar novas versões, com as novas atividades finalizadas (esse processo é muito conhecido como “GMUD”, também chamado de “gestão de mudança”).

Para o *kanban*, também é natural que as equipes separem uma reunião periódica para revisar o seu processo de desenvolvimento, procurando otimizar os processos e a eficiência geral do time. Como pode ver, você basicamente estaria recriando a *sprint retrospective* do *scrum*, porém sem uma *sprint*, e sim com uma janela de tempo.

Sem uma *sprint*, não há mais os processos de *planning* e de *sprint review*. Esses dois processos são normalmente trocados por novas colunas no quadro de trabalho. Com isso, a automatização de tarefas e a definição de processos bem elaborados e estruturados são fatores essenciais para uma boa utilização do *kanban*, o que exige maior maturidade do time de desenvolvimento. O *product owner*, por exemplo, continua existindo no *kanban* e tem um esforço ainda mais importante, precisando organizar muito bem o *product backlog*, ou seja, a lista de atividades para fazer, e acompanhando a satisfação e as expectativas dos *stakeholders*, uma vez que não se tem mais a *sprint review* e o *planning*, cerimônias para alinhar a direção do desenvolvimento do *software*.

Por último, o *scrum master* deixa de existir no *kanban*, sendo substituído pelo *agile coach*, um profissional com bons conhecimentos de metodologia ágil, responsável por auxiliar a equipe a otimizar os seus processos e fazer boas entregas incrementais do *software*.

Como se pode perceber, o *kanban* compartilha diversos conceitos com o *scrum*. Isso acontece porque o objetivo das duas estratégias é o mesmo: entrega incremental de *software* com agilidade para redirecionamento e mudanças. A maior diferença é a não existência de *sprints*, o que impacta em outras cerimônias e papéis.

É importante saber que não existe “bala de prata” na engenharia de *software*. Não existe uma única solução que melhor atenderá a todos os cenários. Portanto, conheça as diferentes estratégias e metodologias para que você saiba aplicar o que é mais efetivo para os diferentes contextos.

A metodologia tradicional é menos custosa, no sentido de esforço de trabalho da equipe. Além disso, pelo tempo de existência, é uma abordagem mais madura, com uma vasta possibilidade de ferramentas e materiais, auxiliando na sua utilização. É

uma ótima solução para projetos pequenos, rápidos e com escopo fechado.

 Já as metodologias ágeis são ótimas opções para projetos a longo prazo, que têm alto potencial de mudança de escopo e redirecionamento. Com agilidade e flexibilidade para respostas e mudanças, a metodologia ágil atende bem a setores que exigem constante mudança. Entretanto, existem diversas técnicas diferentes, que exigem conhecimentos específicos (o que pode dificultar a curva de *onboarding* para equipes em crescimento). Além disso, é uma estratégia relativamente nova, resultando em menos materiais e ferramentas (em comparação à metodologia tradicional).