Basicamente a definição de pronto é o documento que define o que é considerado, para todos envolvidos no projeto, o que “pronto” (done) significa.

Quais são os requisitos para arrastar um card para a coluna DONE do [Kanban](https://www.luiztools.com.br/post/kanban-como-gerenciar-pipelines-de-desenvolvimento-de-software/)? Se um desenvolvedor diz que algo está pronto, o que isso significa? O que o Product Owner espera ver, em termos gerais, durante a Sprint Review?

Essas são algumas perguntas que podem nortear a construção deste documento, que geralmente e um checklist de coisas a serem realizadas/verificada, antes de uma tarefa ser dada como pronta e, como manda o pilar da transparência do Scrum, deve ser visível e conhecido por todos do time. Dica: coloque ao lado do Kanban.

**Documento colaborativo**

Um primeiro ponto a se considerar é que a criação da definição de pronto deve ser realizada de maneira colaborativa, ou seja, por todos os membros do Time Scrum. Claro, alguns membros possuem mais direitos, e outros mais deveres, como o Time de Desenvolvimento, por exemplo, que será o principal encarregado de implantar os itens da definição de pronto na sua rotina de desenvolvimento. Já o Scrum Master deverá garantir e auxiliar o time na execução do que prega este documento, afinal, ele faz parte do processo Scrum e deve ser respeitado. E por fim, o Product Owner espera que todos os itens dados como prontos e entregues estejam dentro do padrão de qualidade acordado na definição de pronto.

Achou complicado? Não deveria, é mais simples do que parece.

Um bom começo é, durante a primeira [Sprint Planning](https://www.luiztools.com.br/post/dicas-para-um-sprint-planning-de-sucesso/), o time definir a v1 da sua definição de pronto. Pense em uma folha A4 e comece com coisas simples como dizer que “todo item dado como pronto deve ter passado em testes unitários” e depois se aprofunde em itens mais “avançados” como testes de regressão, teste em pares, etc e até mesmo itens difíceis dependendo da disponibilidade do Time Scrum como “aprovar com o Product Owner”. Sim, é bem complicado deste último item ser factível pois geralmente o Product Owner não é (mas deveria ser) tão acessível quanto gostaríamos.

Como tudo no Scrum (pilares da inspeção e adaptação, lembra?), itere e melhore sua definição de pronto a cada Sprint. Pegue o que deu errado na Sprint Review (ela nunca sai 100% como foi planejado pelo Time de Desenvolvimento), aborde na [Sprint Retrospective](https://www.luiztools.com.br/post/como-criar-retrospectivas-de-sprint-que-funcionam/) e aplique de maneira aperfeiçoada na próxima Sprint Planning. Comece simples e avance rapidamente. Lembre-se que a função deste artefato é garantir a qualidade, mas lembre-se também de se manter ágil. A dose certa de um e de outro é você que vai descobrir.

E por último, eu sugiro fazer na Sprint Planning pois dependendo dos itens colocados na sua definição de pronto, o tempo para que cada entrega fique pronta pode mudar drasticamente. Tenha a definição de pronto pronta antes de jogar [Planning Poker](https://www.luiztools.com.br/post/planning-poker-como-estimar-tempo-de-desenvolvimento-de-software/), por exemplo.

**Contrato moral**

A definição de pronto é algo com o qual o time se compromete a cumprir para garantir a qualidade das entregas. Sendo assim, é um contrato moral. Moral porque estamos falando de pessoas e processos, não há um elemento de software envolvido, lhe cobrando diariamente que cumpra os requisitos do documento, embora o time possa optar por usar algum, como um dos itens da definição, mas nada a substitui.

Sendo um contrato moral e ao mesmo tempo algo colaborativo, o time terá de achar o checklist que agrade a todos, incluindo aqui o Product Owner, que é quem tem a palavra final sobre os itens do Backlog que o Time de Desenvolvimento está trabalhando. Jamais crie uma definição em que não há unanimidade dentro do Time pois caso contrário ela será sabotada, mais cedo ou mais tarde. Caso o time seja inexperiente, como Scrum Master “force” algumas regras pedindo um voto de confiança, explique que esse documento poderá ser alterado na próxima Sprint caso o Time não se adapte.

Mas principalmente: crie um documento que seja útil para garantir a qualidade das entregas que seja executável, exequível. É muito fácil cair na tentação de adicionar dezenas e dezenas de itens de qualidade que jamais serão empregados no projeto como “validar pessoalmente com o cliente final” seja porque não é prático, seja porque realmente e inviável (ex: cliente do outro lado do mundo). Se o Time optar por ferramentas, escolha o menor conjunto delas possível, caso contrário o tempo de desenvolvimento poderá ser enormemente afetado ou a definição enormemente sabotada.

E se você acha que seu Time não dará atenção a um contrato moral, que tipo de time você montou para executar o projeto?

[](https://www.luiztools.com.br/livro-agile)

**Exemplos e Contra-exemplos**

A seguir alguns exemplos de itens que já implementei em Times Scrum que eu liderei como Scrum Master e que sei que funcionam. Obviamente, alguns deles não funcionam dentro de realidades fora das que eu vivenciei, uma vez que a maioria das empresas que trabalhei desenvolviam software para si mesmas (cliente interno). Também obviamente não pegue todos e coloque na sua definição. Use-os como ideias, não tenho intenção alguma definir o que significa “pronto” para o SEU time de desenvolvimento.

Lembre-se que a definição de pronto deve ser clara e não permitir desculpas como “está pronto, só falta testar”…

Toda tarefa de software, para ser considerada pronta, deve…

**Ter sido atualizado com o controle de versão e permanecer compilando;**

Isto é o mínimo que se espera de algo dado como pronto. Em Times que usem versionamento de código (algum não usa?) o desenvolvedor, após concluir a codificação da tarefa, pegar a última versão do servidor, rodar seus testes (ver abaixo), fazer a fusão (merge) do que for necessário e garantir que, antes de enviar seu código amalgamado ao servidor, que tudo continua compilando. Ponto.

*Contra-exemplo: cuidado com o envio de software inacabado (mesmo que algumas features individuais já estejam) para produção. Versionadores de código (como TFS, SVN, Git, etc) possuem recursos como branches e forks que permitem aos desenvolvedores manterem sempre uma versão de produção 100% operacional e livre de bugs enquanto trabalham em outra versão mais instável. Colocar na definição de pronto que o requisito de software desenvolvido esteja rodando em produção é muito perigoso.*

**Ter passado nos testes unitários com sucesso;**

Se você ainda não usa [Testes Unitários](https://www.luiztools.com.br/post/tdd-como-criar-unit-tests-em-node-js-com-tape/) em sua equipe de desenvolvimento, deveria. A ideia aqui, basicamente, é que, se você testar cada uma das micro-partes que envolvem o seu software (unidades) isoladamente, a probabilidade de que o todo funcione é muito maior, ao mesmo tempo em que lhe obriga a manter um baixo acoplamento do seu software para que ele possa ser testado em micro-pedaços.

*Contra-exemplo: colocar*[*TDD*](https://www.luiztools.com.br/post/tdd-como-criar-unit-tests-em-node-js-com-tape/)*na definição de pronto sem que o time tenha experiência com a metodologia. TDD é bem complicado e difícil de ser seguido à risca, mais do que o próprio Scrum.*

**Ter passado por testes de uso de outro colega da equipe;**

Desenvolvedores tendem a ser os piores testadores do mundo quando o assunto é testar suas próprias criações. Certamente temos um gene defeituoso que faz com que auto-sabotemos os testes visando não encontrar bugs. Geralmente outro colega da equipe (chamado cross-testing ou [peer-review](https://pt.wikipedia.org/wiki/Revis%C3%A3o_por_pares)), que não está preocupado com o trabalho que vai dar corrigir qualquer erro encontrado, é um testador melhor que o desenvolvedor original. E se isso não for suficiente para garantir a qualidade, em casos extremos mude para “teste de uso de quem vai usar a feature”, chamando para o teste de uso o próprio usuário da feature.

Ah, esse teste de uso deve ser em ambiente de homologação. Nada de chamar o colega ou o usuário final pra usar na tua máquina. Isso evita aquela famigerada frase: “na minha máquina funciona!”.

*Contra-exemplo: teste de uso automatizado. Isso faz com que você perca a experiência da usabilidade do software. Podem existir testes automatizados, mas eles não devem substituir por completo os testes manuais quando o recurso envolver interface gráfica.*

**O código encontra-se dentro dos padrões da empresa;**

É comum, embora não deveria, desenvolvermos uma v1 de qualquer funcionalidade de maneira meio…porca. O que não é “comum” é que essa v1 seja a versão “pronta” daquele requisito. Refatoração é a chave aqui. Garanta com esse item que o código passe por uma avaliação estrutural para ver se está 100% de acordo com as normas de desenvolvimento da empresa ([ex. do Google](https://google.github.io/styleguide/javaguide.html)). Obviamente para que isso funcione, estas normas também devem estar visíveis e serem de conhecimento geral do Time de Desenvolvimento.

*Contra-exemplo: empresas que possuam normas de desenvolvimento muito particulares tendem a fracassar ao usar este item na definição de pronto. O que seria algo muito particular? Na minha opinião algo que foge aos padrões de codificações oficiais da linguagem/framework utilizados. Quando escrevo código em*[*Java*](https://www.luiztools.com.br/livro-java-i)*, uso os*[*padrões da Oracle*](http://www.oracle.com/technetwork/java/codeconvtoc-136057.html)*(Camel Case para métodos, por exemplo). Já quando escrevo código em C#, uso os*[*padrões da Microsoft*](https://msdn.microsoft.com/en-us/library/ff926074.aspx)*(Pascal Case para métodos, por exemplo). Isso facilita a vida para todos, inclusive novos funcionários e geralmente a resistência por adotar essa prática é culpa do ego dos veteranos da empresa.*

**Softwares de apoio e documentação atualizados;**

Um último item na definição de pronto deve dar cabo de tarefas burocráticas e pouco interessantes, mas igualmente necessárias, como essas. Todo Time de Desenvolvimento usa algum software para controlar seu progresso ([Trello](https://www.trello.com/), por exemplo). Mas para garantir que esse software mantenha sua utilidade, ele deve permanecer atualizado com o andamento do projeto e aqui o trabalho do Scrum Master deve ser bem forte, porque o time sempre se “esquece” de atualizar. Cobrar atualizações diárias, sempre antes da Daily Scrum, geralmente resolvem este problema, mesmo antes das tarefas serem dadas como pronto.

O mesmo vale para a documentação do projeto. Eu particularmente sempre gostei de Wikis internos por serem um formato de documentação viva, colaborativa e fácil de usar e manter. Obviamente tem empresas que preferem o bom e velho Word. Encontre o que funciona para você, mas garanta que um mínimo de documentação esteja sempre atualizada e visível, como diagramas da arquitetura, do banco de dados, de implantação, etc.

*Contra-exemplo: tão nocivo quanto não ter documentação é tê-la em demasia, então tome cuidado com este item, caso contrário o time pode não se comprometer de verdade, gerando um comprometimento flácido.*

**Conclusão**

Estas foram algumas ideias de como criar ou incrementar a sua definição de pronto. A combinação de uma definição de pronto clara, com objetivos e escopos definidos, tendem a garantir entregas consistentes e de qualidade, levando o projeto ao sucesso.

Associar a sua definição de pronto a acrônimos como [DEEP, INVEST e SMART](https://www.luiztools.com.br/post/backlog-deep-historias-invest-e-tarefas-smart/) também pode ajudar bastante!

<https://www.luiztools.com.br/post/como-criar-uma-definicao-de-pronto-que-funciona/>