

ENTENDENDO E APRENDENDO
O FRONT-END COM PROJETOS

REALIZANDO AS PRIMEIRAS ENTREGAS
TÉCNICAS DA IMERSÃO

HENRIQUE POYATOS



10

LISTA DE FIGURAS

Figura 1 – Pilares do Scrum	6
Figura 2 – Valores do Scrum.....	8
Figura 3 – Produzir com qualidade.....	13
Figura 4 – Resumo do <i>framework</i>	14
Figura 5 – O framework Scrum	14
Figura 6 – Documento Visão	16
Figura 7 – Sequência de Fibonacci.....	27
Figura 8 – <i>Planning Poker</i> Fonte: Elaborado pelo autor, adaptado por FIAP (2019)	28
Figura 9 – Hierarquia do <i>release</i>	31
Figura 10 – Sequência de Planejamento de releases	32
Figura 11 – Modelo de Tuckman.....	35

LISTA DE TABELAS

Tabela 1 – Priorização com matriz GUT	25
Tabela 2 – Exemplo de <i>Planning Poker</i>	29
Tabela 3 – Exemplo de <i>Planning Poker</i> (2)	29
Tabela 4 – Exemplo de <i>Planning Poker</i> (3)	30

EMANIP

LISTA DE QUADROS

Quadro 1 – Interpretação de valores do Scrum.....	9
Quadro 2 – Exemplo de um modelo aplicado.....	17
Quadro 3 – Exemplo de um <i>Product Backlog</i>	19
Quadro 4 – Priorizando as histórias	21
Quadro 5 – Técnica MoSCoW.....	22
Quadro 6 – Matriz GUT	23
Quadro 7 – Pesos da matriz GUT	24
Quadro 8 – Exemplo de um <i>Product Backlog</i> priorizado.....	26
Quadro 9 – Exemplo de Plano de <i>Releases</i> orientado ao escopo	33

SUMÁRIO

1 REALIZANDO AS PRIMEIRAS ENTREGAS TÉCNICAS DA IMERSÃO	6
1.1 Introdução	6
1.2 Pilares do SCRUM	6
1.3 Valores do SCRUM	7
1.4 Papéis no SCRUM	10
1.4.1 <i>Product Owner</i>	10
1.4.2 Scrum Master	11
1.4.3 Time	11
2 O FRAMEWORK SCRUM.....	13
2.1 Iniciando o projeto	16
2.2 Criando o <i>Product Backlog</i>	18
2.2.1 Priorizando as histórias	22
2.2.2 Técnica MoSCoW	22
2.2.3 Matriz GUT	23
2.2.4 Importância subjetiva.....	25
2.3 Estimativas	27
2.4 Planejamento de <i>releases</i>	30
2.5 Cálculo da velocidade	34
REFERÊNCIAS.....	39

1 REALIZANDO AS PRIMEIRAS ENTREGAS TÉCNICAS DA IMERSÃO

1.1 Introdução

Um *software* (como aquele que você fará) é composto por uma série de características almejadas pelas partes interessadas. Tais desejos são requisitados à equipe de desenvolvimento e, por serem de naturezas tão diferentes, precisam ser anotados de uma maneira organizada.

Neste capítulo, vamos aprender o bê-á-bá do Scrum e descobrir como as empresas dão início aos seus projetos por meio deste framework.

1.2 Pilares do SCRUM

No Scrum Guide®, estão definidos os três pilares básicos do SCRUM que sustentam o framework e trazem os verdadeiros ganhos aos projetos. Eles estão ilustrados na figura “Pilares do Scrum”:

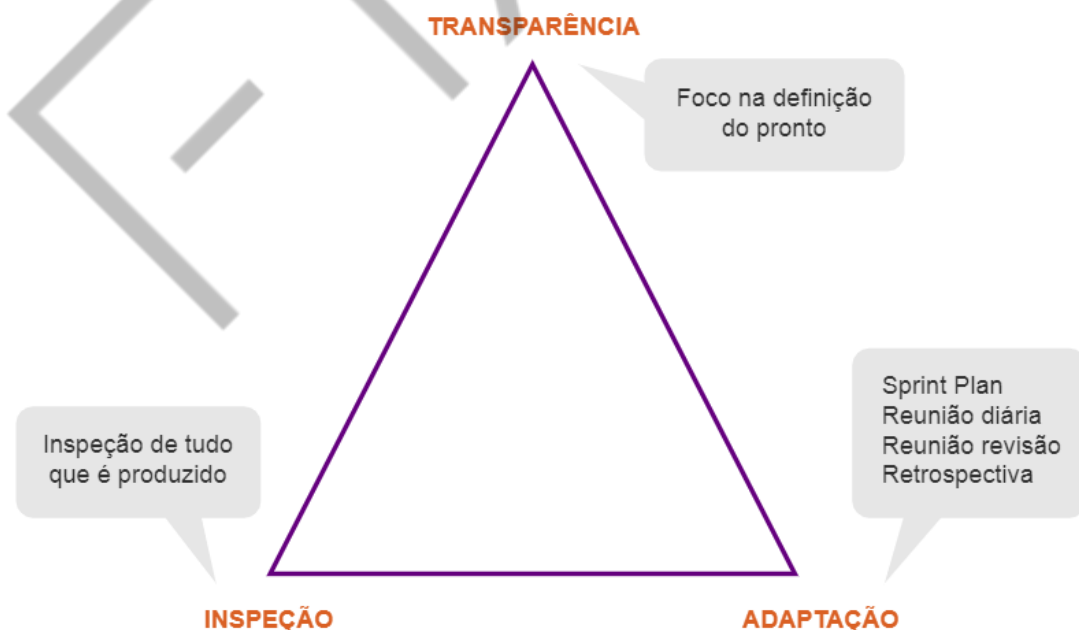


Figura 1 – Pilares do Scrum
Fonte: Scrum Guide, adaptado por FIAP (2017)

A transparência refere-se a deixar claro para todos qual é a situação real das atividades, alinhando expectativas e realizando feedback constante com o cliente, permitindo a verificação da evolução do projeto.

A inspeção é um preceito de qualidade que deve ser observado sempre. Tudo que é produzido precisa ser inspecionado para garantir que está aderente ao que foi solicitado. Ágil não significa ser somente rápido, mas também fazer e entregar com qualidade.

O pilar de inspeção no Scrum é exercitado em cada um dos 5 eventos ou cerimônias: antes do Sprint se inspeciona o Product Backlog e o Sprint Backlog no Planejamento; durante o Sprint se inspeciona o andamento do trabalho nas diárias; ao fim do Sprint se inspeciona na Review o incremento sendo entregue e na Retrospectiva se inspeciona a dinâmica interacional e de processos do time. O próprio Sprint é o 5º evento, no qual se passam todas estas oportunidades de inspeção.

Aqui aplica-se o conceito de um faz e outro revisa para melhorar o resultado final.

O processo de adaptação é obtido por meio das diversas reuniões que ocorrem durante o ciclo Scrum, em que todos avaliam o desempenho atual, criticam, fazem adequações imediatas e/ou melhorias para o próximo ciclo.

1.3 Valores do SCRUM

Na sua versão 2020 o Scrum Guide® incluiu o conceito de valores importantes para serem compreendidos por todo o time. O objetivo é de tornar claras algumas definições que conflitam com a visão tradicional dos projetos e trazem à tona algumas quebras de paradigmas.



Figura 2 – Valores do Scrum
Fonte: Scrum Guide, 2016, adaptado por FIAP (2017)

Vamos agora analisar, no Quadro “Interpretação de valores do Scrum”, cada um desses valores e sua interpretação correta em relação ao que o Scrum preconiza:

Valores	Visão tradicional	Visão scrum
Compromisso	Comprometer-se só porque o seu chefe mandou.	Comprometimento com o time e a meta do <i>Sprint</i> .
Foco	Manter o cliente satisfeito.	Entregar o combinado no prazo.
Abertura	Falar bem do que você faz	Destacar o sucesso de todos e alertar para os impedimentos
Respeito	Você como herói para resolver os problemas	Auxiliar o time a fazer o melhor e não julgar os outros
Coragem	Culpar a tudo e a todos pelos problemas.	Aceitar opiniões dos outros, críticas e assumir seus erros.

Quadro 1 – Interpretação de valores do Scrum
Fonte: Adaptado de West (2019)

Segundo Dave West (2016), algumas ações podem ser tomadas para auxiliar os envolvidos a cultivarem esses valores e romperem com os preconceitos estabelecidos para melhorar o ambiente e o desempenho do time:

- Coloque os valores em uma parede e peça para cada membro da equipe escrever como eles estão sendo aplicados em seu dia a dia no trabalho.
- Adicione um “momento dos valores” em sua retrospectiva. Essa experiência dará a todos uma oportunidade para inspecionar e adaptar os valores em sua equipe.
- Considere introduzir um prêmio ‘valores’ para estimular a prática e o reconhecimento de todos os envolvidos.

1.4 Papéis no SCRUM

No framework SCRUM, os papéis são muito bem definidos e reduzidos, minimizando conflitos e aumentando o grau de comunicação entre todos os envolvidos. São três os papéis descritos pelo SCRUM:

1.4.1 *Product Owner*

O *Product Owner* é o representante do cliente no projeto e deve ter autonomia e autoridade para a tomada de decisões e validações necessárias no projeto.

- Define as funcionalidades do produto, mantendo o *Product Backlog* atualizado;
- Decide datas de lançamento e conteúdo dos *releases*;
- Responsável pela rentabilidade (ROI);
- Prioriza funcionalidades de acordo com o valor;
- Ajusta funcionalidades e prioridades;
- Apresenta ao time os requisitos necessários para a entrega do produto;
- Aceita ou rejeita o incremento de software entregue ao fim do Sprint
- Atua como facilitador quando há mais clientes envolvidos;
- Garante que especialistas estejam disponíveis para o time;
- Desenvolve e comunica a visão de produto;
- Cria e define itens do *Product Backlog*;
- Garante que o *Product Backlog* é transparente, visível e compressível para todos do time.

1.4.2 Scrum Master

O *Scrum Master* é o papel que trabalha para garantir eficiência de trabalho do time, ou seja, atua como facilitador, removendo impedimentos, organizando o Scrum e auxiliando a organização, time e cliente a trabalharem juntos; é um dos principais elos dos times com o *Product Owner*, protegendo o time para que esse não perca o foco na construção do produto.

- Pode representar a gerência para o projeto (não é incomum que gerência de projeto e Scrum Master sejam papéis separados);
- Responsável pela aplicação dos valores e práticas do Scrum;
- Remove os impedimentos;
- Garante a plena função e produtividade da equipe;
- Garante a colaboração entre os envolvidos;
- Escudo para interferências externas;
- Responsável por garantir que o produto atenda às necessidades definidas pelo *Product Owner*;
- Auxilia o *Product Owner* com o *Product Backlog*;
- Facilita as reuniões.

1.4.3 Time

O time é auto-organizado e autogerenciado e sua responsabilidade é a execução dos trabalhos acordados nos planejamentos com o *Product Owner* e o *Scrum Master*. Não existe diferença de título entre testador e arquiteto. Todos compartilham o mesmo título de *Scrum Team Member*.

As equipes SCRUM devem ser pequenas o suficiente para permanecerem ágeis e grandes o suficiente para conseguirem desenvolver um trabalho significativo ao fim do Sprint. Possuem tipicamente 10 (dez) membros ou menos, incluindo *Team Members*, *Scrum Master* e *Product Owner*, sendo a palavra-chave para o sucesso o COMPROMETIMENTO de todos.

- O trabalho entregue ao cliente é gerado por um “Time Scrum” dedicado;
- Times Scrum são auto-organizados, multidisciplinares e comprometidos;
- Responsáveis por entregar o que foi acordado como meta do *Sprint*;
- O próprio time estipula como as tarefas devem ser divididas para gerar o resultado esperado;
- O próprio time define quem irá executar as tarefas e em que ordem elas serão realizadas.

Agora que você entendeu o papel de cada um, vamos detalhar as principais responsabilidades de cada um no projeto SCRUM, segundo descrito no Scrum Guide®:

2 O FRAMEWORK SCRUM

Como já vimos, o *framework* define papéis claros com times reduzidos, descreve um conjunto de processos básicos que estabelecem o que deve ser feito (o “como” fica a critério de cada time), um conjunto mínimo de artefatos que devem ser produzidos durante o ciclo de execução (cabendo ao time definir outros que julgue necessários), faz uso de ferramentas ágeis (outras que acredite precisar para facilitar a execução e o acompanhamento dos trabalhos) e parte da premissa que os envolvidos já se desvencilharam de paradigmas tradicionais e incorporaram conceitos como:

O cliente é uma parte presente e colaborativa no projeto, tornando “o projeto de todos” e não só do time, usando o conceito de líder servidor e produzindo com qualidade.



Figura 3 – Produzir com qualidade
Fonte: Banco de Imagens Shutterstock (2017)

Esse conjunto de definições são detalhadas na Figura “Resumo do *framework*”.

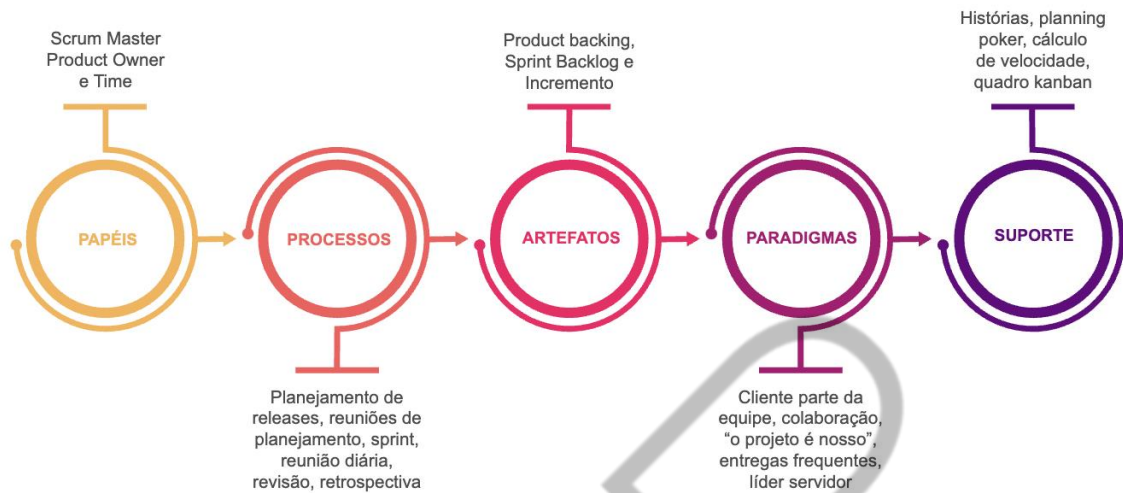


Figura 4 – Resumo do *framework*
 Fonte: Elaborado pelo autor, adaptado por FIAP (2017)

Vamos detalhar cada uma das etapas do *framework* adiante. Primeiramente, vamos alinhar as terminologias para facilitar o entendimento.

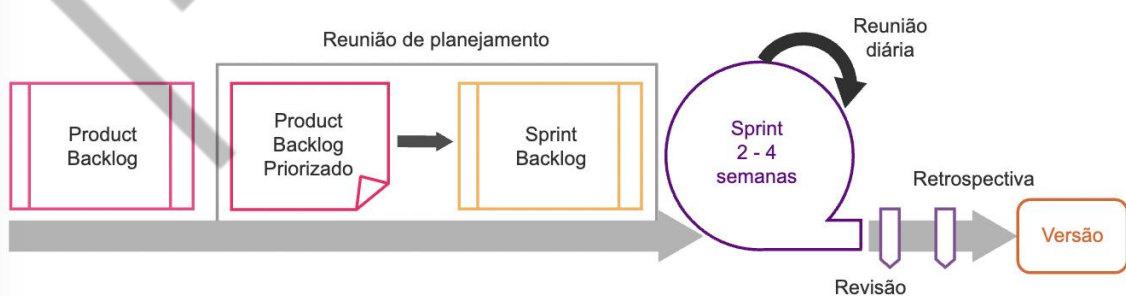


Figura 5 – O framework Scrum
 Fonte: Scrum Guide®, adaptado por FIAP (2019)

- **Product Backlog:** o Product Backlog é uma lista ordenada de necessidades de aprimoramento de produto, elaborada pelo Product Owner a partir dos requisitos a partir dos requisitos iniciais fornecidos pelo cliente que descreve tudo que deverá ser feito no projeto, sendo a referência para a execução do trabalho. As tarefas que o compõem devem ser organizadas em ordem de importância.
- **Reunião de planejamento:** reunião realizada no início de cada *Sprint*, com o objetivo de definir o que será feito no *Sprint*. Todos os membros do projeto participam.
- **Sprint Backlog:** lista detalhada de atividades que precisam ser executadas durante o Sprint para gerar o produto esperado. Deve ser elaborada e estimada pelo time do projeto.
- **Sprint:** tem duração predeterminada de 1 a 4 semanas, não pode ser estendido nem reduzido. É no *Sprint* que o produto é elaborado e entregue com qualidade.
- **Reunião diária:** reunião breve, muitas vezes, realizada em pé e no local de trabalho, com duração máxima de 15 minutos. O objetivo dessa reunião é acompanhar as atividades que estão sendo realizadas e melhorar a comunicação do time.
- **Reunião de revisão:** reunião em que o time apresenta o resultado do trabalho ao *Product Owner* e seus convidados (clientes, gestores e outros stakeholders). O objetivo é fazer uma avaliação e definir se o incremento de *software* produzido ao longo do *Sprint* será aceito. Eventuais apontamentos de erros e melhorias devem ser inseridos no *Product Backlog*.
- **Reunião de retrospectiva:** reunião na qual são discutidos os pontos positivos e negativos da execução de cada *Sprint*, com o objetivo de melhorar o desempenho dos próximos *Sprints*.
- **Versão:** nem todo *Sprint* gera um produto final a ser utilizado pelo cliente. Às vezes são necessários mais *Sprints* para produzir uma versão fechada para utilização.

2.1 Iniciando o projeto

Um projeto SCRUM começa a partir de um documento apresentado pelo *Product Owner* denominado Visão do Produto. Esse documento contém todas as necessidades que devem ser atendidas pelo projeto, juntando as informações fornecidas pelos principais clientes, usuários, patrocinadores, entre outros, de modo que reflita como o produto deve ser ao final do projeto, conforme Figura “Documento Visão”.

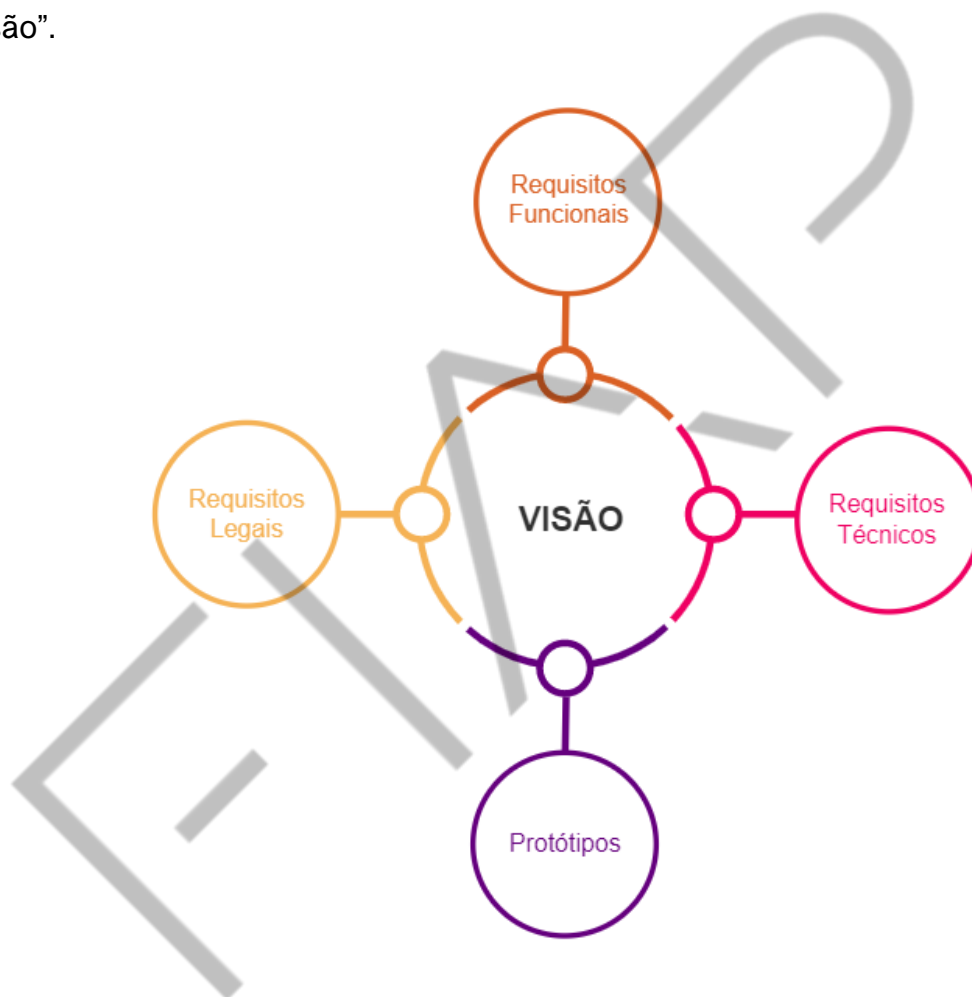


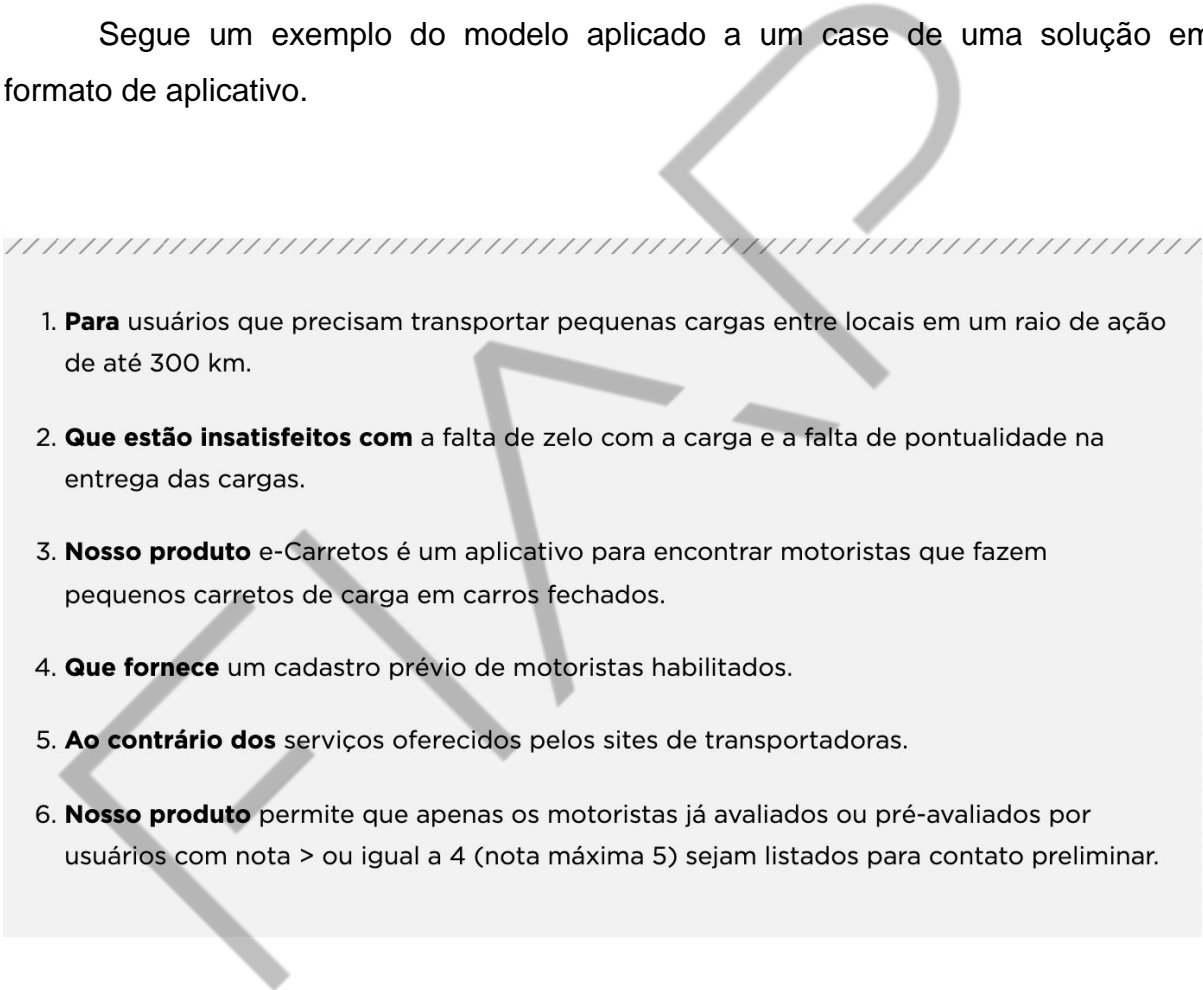
Figura 6 – Documento Visão
Fonte: Elaborado pelo autor, adaptado por FIAP (2019)

Existe um modelo de declaração de visão que se chama Elevator *pitch*, modelo de estrutura padronizada proposto por Moore (2001), como se segue:

1. **Para** (clientes-alvo).
2. **Que estão insatisfeitos com** (a atual alternativa de mercado).

3. **Nosso produto** (nome do produto) **é uma** (proposta do nosso produto).
4. **Que fornece** (capacidade-chave de resolução de problemas).
5. **Ao contrário de** (a melhor alternativa do produto).
6. **Nosso produto** (proposta de valor, nosso diferencial).

Segue um exemplo do modelo aplicado a um case de uma solução em formato de aplicativo.

- 
1. **Para** usuários que precisam transportar pequenas cargas entre locais em um raio de ação de até 300 km.
 2. **Que estão insatisfeitos com** a falta de zelo com a carga e a falta de pontualidade na entrega das cargas.
 3. **Nosso produto** e-Carretos é um aplicativo para encontrar motoristas que fazem pequenos carretos de carga em carros fechados.
 4. **Que fornece** um cadastro prévio de motoristas habilitados.
 5. **Ao contrário dos** serviços oferecidos pelos sites de transportadoras.
 6. **Nosso produto** permite que apenas os motoristas já avaliados ou pré-avaliados por usuários com nota > ou igual a 4 (nota máxima 5) sejam listados para contato preliminar.

Quadro 2 – Exemplo de um modelo aplicado
Fonte: Elaborado pelo autor (2021)

Além da versão de Moore, temos também a versão Harvard-MIT, que é semelhante, porém mais generalista.

Acesse: <https://ocw.mit.edu/courses/health-sciences-and-technology/hst-921-information-technology-in-the-health-care-system-of-the-future-spring-2009/lecture-notes/MITHST_921S09_lec07_tu_pch.pdf>.

Segue a estrutura abaixo:

1. Problema
2. Solução
3. Quem é você e porque você deve resolver o problema
4. Proposta de valor
5. Call-to-action

2.2 Criando o *Product Backlog*

Na definição mais simples, o *Product Backlog* é uma lista de todas as coisas que precisam ser feitas dentro do projeto, elaborada a partir da Visão do Produto. Ele não substitui os artefatos de especificação de requisitos tradicionais, pois apenas reflete o conteúdo macro do que será executado.

O proprietário do *Product Backlog* é o *Product Owner*, mas o *Scrum Master* e o Time podem contribuir para se ter uma ampla e completa lista de tarefas para garantir o atendimento às necessidades do usuário.

Como o *Product Backlog* não é um documento detalhado, ele pode ser completado por outros documentos que trazem maior riqueza de detalhes e vão permitir melhor entendimento das funcionalidades. Alguns exemplos de artefatos adicionais que podemos ter são:

- Memorial descritivo.
- Um resumo das várias funções de usuário.
- Descrições de fluxo de trabalho.
- Diretrizes de interface de usuário.
- Storyboards.
- Requisitos técnicos.
- *Bugs* identificados previamente.
- Melhorias.
- Casos de uso ou protótipos de interface de usuário.

O *Product Backlog* não é um documento estático e vai existir durante todo o ciclo de vida do projeto, devendo ser continuamente atualizado pelo *Product Owner* para refletir tudo o que acontece com os requisitos durante o projeto.

À medida que novos itens são descobertos, eles são descritos e adicionados à lista. Os já existentes podem ser alterados ou eliminados, conforme o projeto evolui, portanto, o *Product Backlog* nunca está completo.

IMPORTANTE: O dono e responsável pelo *Product Backlog* é o *Product Owner*.

Segue um exemplo de *Product Backlog*:

#	Descrição
1	Como atendente, posso consultar os veículos disponíveis para locação.
2	Como atendente, posso fazer locação de um veículo por km livre ou controlada.
3	Como atendente, preciso aprovar o crédito do cliente antes de fazer a locação do veículo.
4	Como atendente, posso cadastrar os clientes que vão fazer locação de veículos.
5	Como atendente, posso fazer o pagamento da locação com cartão de crédito, validando no sistema da operadora.
6	Como atendente, posso fazer o pagamento da locação com cartão de crédito, validando no sistema bancário.
7	Como atendente, posso imprimir um comprovante da locação de veículos.
8	Como atendente, posso receber a devolução do veículo locado na data prevista.

Quadro 3 – Exemplo de um *Product Backlog*
Fonte: Elaborado pelo autor (2021)

Cartões de histórias não fazem parte do *framework* SCRUM, mas é a técnica mais utilizada para descrever o *Product Backlog*. Uma história representa uma pequena parte da funcionalidade a ser construída, ou seja, não é uma completa especificação de requisitos, mas deve ser suficiente para que o time compreenda o que deve ser feito e possa estimar sua complexidade.

Tipicamente, uma história é descrita seguindo o formato:

“COMO UM <<papel>>, EU GOSTARIA DE <<funcionalidade>> PARA <<benefício>>

Exemplo:

EU, COMO **vendedor**, GOSTARIA DE **ver a lista de pedidos comissionados**
PARA **saber quanto irei receber de comissão**.

Repare que cada história é escrita da perspectiva de um usuário do sistema a ser desenvolvido (no exemplo, um vendedor) que precisa realizar uma ação (ver a lista de pedidos comissionados) com um propósito em mente (saber quanto irá receber de comissão). Então, a montagem do *Product Backlog* comumente é uma lista de histórias representando cada item que precisa ser realizado no projeto.

Vamos agora dar um exemplo da construção de um *Product Backlog* para um projeto de um Sistema de Compra de Automóveis pela Internet a partir da Visão de Produto descrita abaixo.

Visão do Produto:

- Uma empresa tem apenas uma loja de vendas de carros e todo o processo é manual, sem nenhuma automação.
- A empresa deseja desenvolver um sistema para realizar a venda de automóveis pela Internet.
- O usuário deverá se cadastrar no site, procurar e detalhar os automóveis disponíveis, escolher o automóvel que deseja comprar e efetuar a compra por meio de Cartão de Crédito.

- O usuário também pode cadastrar o seu veículo para venda que alimenta o site para novos negócios.

Para criar as histórias, vamos identificar os papéis e as funções que cada um espera que o sistema possua.

Neste cenário, temos um site em que você pode fazer a compra de um automóvel e também pode publicar seu veículo para a venda. Tudo feito pelo próprio usuário. Então, temos:

#	História
1	Eu, como usuário, posso me cadastrar no site para fazer compra e venda de veículos.
2	Eu, como usuário, posso realizar buscas para encontrar o tipo de veículo que desejo.
3	Eu, como usuário, posso ver os detalhes do veículo à venda.
4	Eu, como usuário, posso fazer a compra de um veículo com cartão de crédito.
5	Eu, como usuário, posso cadastrar meu veículo para a venda no site.

Quadro 4 – Priorizando as histórias
Fonte: Elaborado pelo autor (2019)

Observe que não precisamos entrar no detalhe para definir a história, mas ela é informação suficiente para ser entendida, mas não sabemos ainda os detalhes que serão adicionados depois no *Sprint Planning*.

2.2.1 Priorizando as histórias

Após listar todas as histórias, o *Product Owner* deve priorizá-las em ordem de importância com o objetivo de criar uma sequência que gere valor para o negócio e faça sentido como um *Sprint* ou uma versão.

Existem diversas técnicas de priorização, mas vamos analisar algumas delas:

2.2.2 Técnica MoSCoW

Esta é uma técnica muito utilizada para priorização de escopo, priorização de requisitos e para classificação de mudanças. O objetivo desta técnica é classificar as funcionalidades com base no seu valor de negócio. O funcionamento da técnica é explicado no Quadro “Técnica MoSCoW”:

#	Significado	-	O requisito é:
M	Must have	Deve ter	Essencial para o negócio.
O	-	-	
S	Should have	Deveria ter	Importante, mas não essencial.
C	Could have	Poderia ter	Seria bom ter, mas não é importante.
O	-	-	
W	Won't have	Não precisa agora	Não gera valor para o negócio agora.

Quadro 5 – Técnica MoSCoW
Fonte: Elaborado pelo autor (2021)

Ao classificar os requisitos com MUST HAVE, por exemplo, você tem certeza que precisam ser feitos primeiro e devem estar no primeiro *Sprint* e compor a primeira versão. Esses terão prioridade em relação ao SHOULD HAVE e assim sucessivamente.

2.2.3 Matriz GUT

A matriz GUT é muito utilizada para a priorização de tomada de decisões e para a tomada de ações corretivas e leva em consideração a gravidade da falta de um requisito, a urgência para se resolver o problema e a tendência, caso o problema não se resolva, conforme Quadro “Matriz GUT”.

Importância = G x U x T		
G	Gravidade	Impacto do problema sobre coisas, pessoas, resultados etc.
U	Urgência	Relação com tempo disponível ou necessário para resolver o problema.
T	Tendência	Avaliação da tendência de crescimento, redução ou desaparecimento do problema.

Quadro 6 – Matriz GUT
Fonte: Elaborado pelo autor (2019)

Para cada uma dessas variáveis, são atribuídos pesos de acordo com critérios preestabelecidos que vão direcionar a categorização, conforme Quadro “Pesos da matriz GUT”.

Pontos	Gravidade	Urgência	Tendência
5	Os prejuízos ou dificuldades são extremamente graves	É necessária uma ação imediata.	Se nada for feito, o agravamento será imediato.
4	Muito grave	Com alguma urgência.	Vai piorar em curto prazo.
3	Grave	O mais cedo possível.	Vai piorar em médio prazo.
2	Pouco grave	Pode esperar um pouco.	Vai piorar em longo prazo.
1	Sem gravidade	Não tem pressa.	Não vai piorar ou pode até melhorar.

Quadro 7 – Pesos da matriz GUT
Fonte: Elaborado pelo autor (2019)

Após isso, multiplicam-se os pesos de cada variável para se obter um número que, quanto maior, maior a prioridade, conforme mostrado na Tabela “Priorização com matriz GUT”.

#	Descrição	Prioridade
18	Como supervisor da agência de locação, posso cadastrar, alterar, consultar e excluir os veículos para locação.	1
7	Como atendente, posso consultar os veículos disponíveis para locação.	2
10	Como atendente, posso cadastrar os clientes que vão fazer locação de veículos.	3
8	Como atendente, posso fazer locação de um veículo por km livre ou controlada.	4
9	Como atendente, preciso aprovar o crédito do cliente antes de fazer a locação do veículo.	5
20	Como atendente, posso fazer reservas de locações de veículos para clientes cadastrados.	5
14	Como atendente, posso receber a devolução do veículo locado na data prevista.	6
15	Como atendente, posso receber a devolução do veículo locado com diferenças do previsto.	7
19	Como supervisor da agência de locação, posso consultar as locações feitas no dia.	8

Tabela 1 – Priorização com matriz GUT
Fonte: Elaborado pelo autor (2019)

2.2.4 Importância subjetiva

Nesta técnica, o *Product Owner* avalia onde a empresa se encontra e define o que é mais importante a ser feito a fim de atender às necessidades do negócio.

Por exemplo, imagine que uma locadora de automóveis não possui nenhum sistema informatizado. Quais seriam as primeiras funcionalidades que deveriam ser entregues? Provavelmente, as funções de locação e cadastro de veículos e de

clientes, pois a locadora já poderia começar a utilizar esta parte, enquanto é desenvolvido o processo de devolução de uma locação, como no quadro:

#	Descrição	Prioridade
18	Como supervisor da agência de locação, posso cadastrar, alterar, consultar e excluir os veículos para locação.	1
7	Como atendente, posso consultar os veículos disponíveis para locação.	2
10	Como atendente, posso cadastrar os clientes que vão fazer locação de veículos.	3
8	Como atendente, posso fazer locação de um veículo por km livre ou controlada.	4
9	Como atendente, preciso aprovar o crédito do cliente antes de fazer a locação do veículo.	5
20	Como atendente, posso fazer reservas de locações de veículos para clientes cadastrados.	5
14	Como atendente, posso receber a devolução do veículo locado na data prevista.	6
15	Como atendente, posso receber a devolução do veículo locado com diferenças do previsto.	7
19	Como supervisor da agência de locação, posso consultar as locações feitas no dia.	8

Quadro 8 – Exemplo de um *Product Backlog* priorizado
Fonte: Elaborado pelo autor (2019)

2.3 Estimativas

Após a priorização do *Product Backlog*, as histórias precisam ter seu tamanho estimado para que possa ser avaliado quantas histórias cabem num *Sprint*. Caso contrário, elas precisam ser quebradas em histórias menores para melhorar a compreensão do time.

O time é o responsável pelas estimativas, mas a presença do *Product Owner* é essencial para detalhar ou esclarecer cada história.

A técnica de estimativa mais utilizada em projetos *Scrum* é o *Planning Poker*. Esta técnica consiste em cartas com numeração seguindo a tabela de Fibonacci e os membros do time avaliam um tamanho para cada um dos itens do *Product Backlog*.

IMPORTANTE: o tamanho de uma história não é estimado em dias ou em horas, normalmente relaciona-se à complexidade de cada história e também serve de comparação entre elas. Essa numeração atribuída a cada item é chamada de pontos de complexidade.

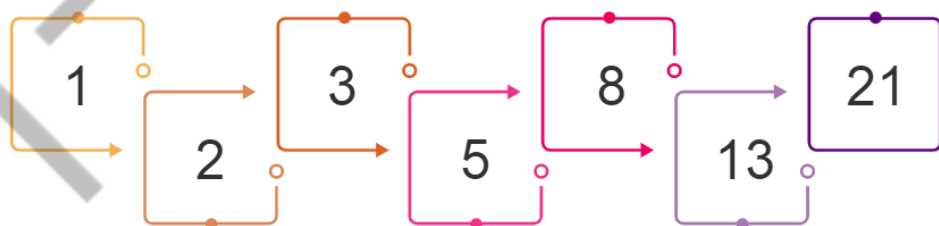


Figura 7 – Sequência de Fibonacci
Fonte: Elaborado pelo autor (2021)

O passo a passo para se jogar o *Planning Poker* é o seguinte:

- Cada membro do time tem a capacidade de votar em quaisquer dos valores secretamente uma única vez. Isso pode ser feito por meio de uma

ferramenta ou com o uso de um baralho com as cartas numeradas na sequência de Fibonacci;

- Escolhe-se uma história do *Product Backlog*;
- O *Product Owner* detalha o que espera dessa história, mas não participa da estimativa;
- O time compreende e tira suas dúvidas;
- Os membros do time votam secretamente nos valores que acreditam representar melhor o tamanho da tarefa. O voto de cada membro do time é revelado simultaneamente. Se usado um baralho, cada membro do time apresenta a carta que representa seu voto ao mesmo tempo;
- O time compara e discute suas estimativas;
- Em caso de divergências, a história é rediscutida e uma nova rodada de votação acontece;
- O ciclo se repete até que todos tenham jogado a mesma carta ou quando o time entra em consenso, ou seja, quando há concordância sobre o tamanho.



Figura 8 – *Planning Poker*

Fonte: Elaborado pelo autor, adaptado por FIAP (2019)

Para exemplificar, vamos tomar uma história do *Product Backlog* da Figura “*Planning Poker*”:

História: como atendente, eu gostaria de consultar os veículos disponíveis para a locação.

Esclarecimento do *Product Owner*: ao entrar no site, deve ser apresentada uma lista de todos os veículos disponíveis para a locação e o detalhe de cada veículo deve ser mostrado com a sua tarifa diária.

O time faz a 1ª jogada de cartas:

NOMES	RODADA 1	RODADA 2	RODADA 3
Huguinho	3		
Zezinho	2		
Luizinho	5		

Tabela 2 – Exemplo de *Planning Poker*
Fonte: Elaborado pelo autor (2019)

Como houve divergência, novos detalhes da história são fornecidos e o time debate o tamanho que cada um jogou. Nova rodada é feita.

NOMES	RODADA 1	RODADA 2	RODADA 3
Huguinho	3	3	
Zezinho	2	3	
Luizinho	5	5	

Tabela 3 – Exemplo de *Planning Poker* (2)
Fonte: Elaborado pelo autor (2019)

Como ainda há uma divergência, novas explicações são realizadas. Não pode haver “vencido pela maioria”, deve haver um consenso natural.

NOMES	RODADA 1	RODADA 2	RODADA 3
Huguinho	3	3	3
Zezinho	2	3	3
Luizinho	5	5	3

Tabela 4 – Exemplo de *Planning Poker* (3)
Fonte: Elaborado pelo autor (2019)

O final só é alcançado quando todos chegam ao mesmo tamanho. No início, pode haver uma demora natural, mas à medida que são feitas jogadas, o consenso ocorre de forma natural.

Com a evolução das estimativas, os tamanhos das histórias passam a ser comparados entre si e podem sofrer ajustes normais durante todo o ciclo do projeto.

DICA: O objetivo da estimativa é obter o comprometimento de todos com o prazo que será utilizado nos *Sprints*.

Itens mais prioritários do Product Backlog podem ser estimados para proporcionar ao Product Owner uma visão geral de tamanho e esforço requerido para a conclusão de certas versões ou funcionalidades. Porém, isso normalmente é feito por partes e somente sob solicitação.

2.4 Planejamento de *releases*

O Planejamento de *Release* é um plano de muito alto nível para determinar o tempo total do projeto, o ciclo de versões que serão geradas e definir quais funcionalidades serão construídas, sua prioridade e quando serão feitas. Um *release* pode ter um ou mais *Sprints* para que possa atingir sua meta, conforme Figura “Hierarquia do *release*”.

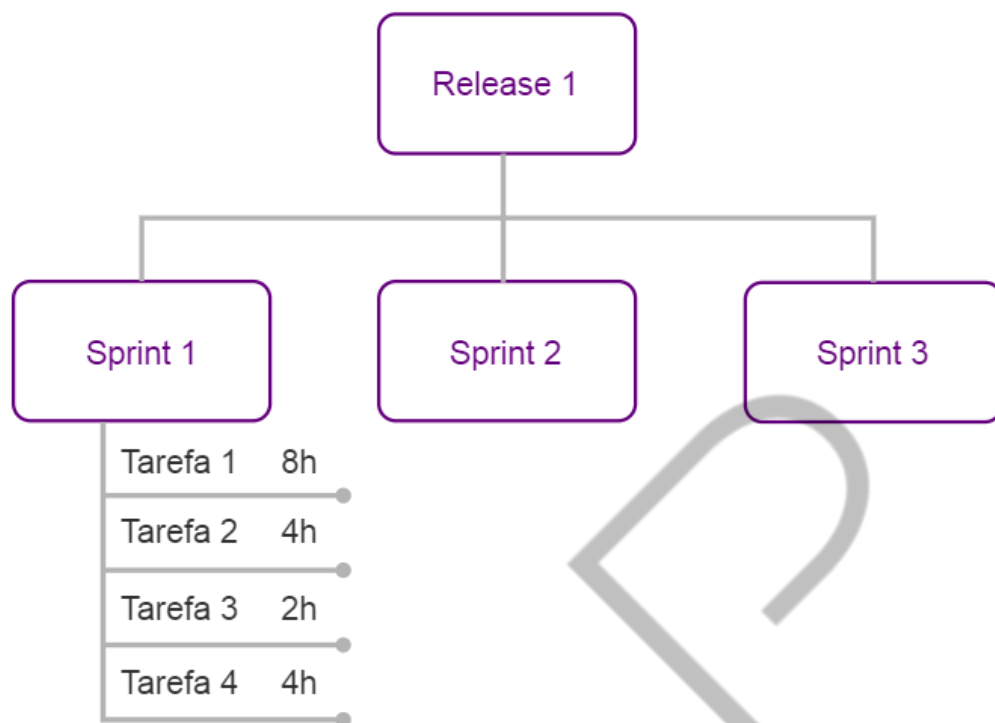


Figura 9 – Hierarquia do *release*
Fonte: Elaborado pelo autor (2019)

O seu objetivo principal é proporcionar o entendimento claro do projeto, sua estratégia de entrega e, com isso, a melhoria da comunicação entre todos os envolvidos.

Segundo o ScrumGuide® 2016:

O plano estabelece a meta do *release*, as maiores prioridades do *Product Backlog*, os principais riscos e as características gerais e funcionalidades que estarão contidas no *release*. Ele estabelece também uma data de entrega e custos prováveis que devem se manter. A organização pode, então, inspecionar o progresso e fazer mudanças neste plano do *release* a cada *Sprint*. O Planejamento do *Release* é inteiramente opcional. Se um Time de *Scrum* iniciar o trabalho sem essa reunião, a ausência de seus artefatos aparecerá como um impedimento que deverá ser resolvido.

A primeira atividade do planejamento de um *release* é determinar sua META a ser definida pelo *Product Owner* com a ajuda do *Scrum Master*. Seu objetivo deve estar alinhado com as necessidades do negócio. A meta serve para guiar o time na realização das atividades e no cumprimento dos *Sprints* planejados.

Para se criar um Plano de *Releases*, devemos seguir os seguintes passos, conforme a Figura “Sequência de Planejamento de *Releases*”:

- Determinar a meta (objetivo) do *release*.
- Ter o *Product Backlog* priorizado e estimado.
- Definir o tamanho do *Sprint* (de 1 a 4 semanas).
- Estimar a velocidade do time.
- Estabelecer quantos *Sprints* e quais as histórias são necessários para cumprir a meta do *release*.

IMPORTANTE: velocidade é o número de pontos que um time é capaz de realizar durante um *Sprint* de tamanho determinado. Veremos como calcular adiante.

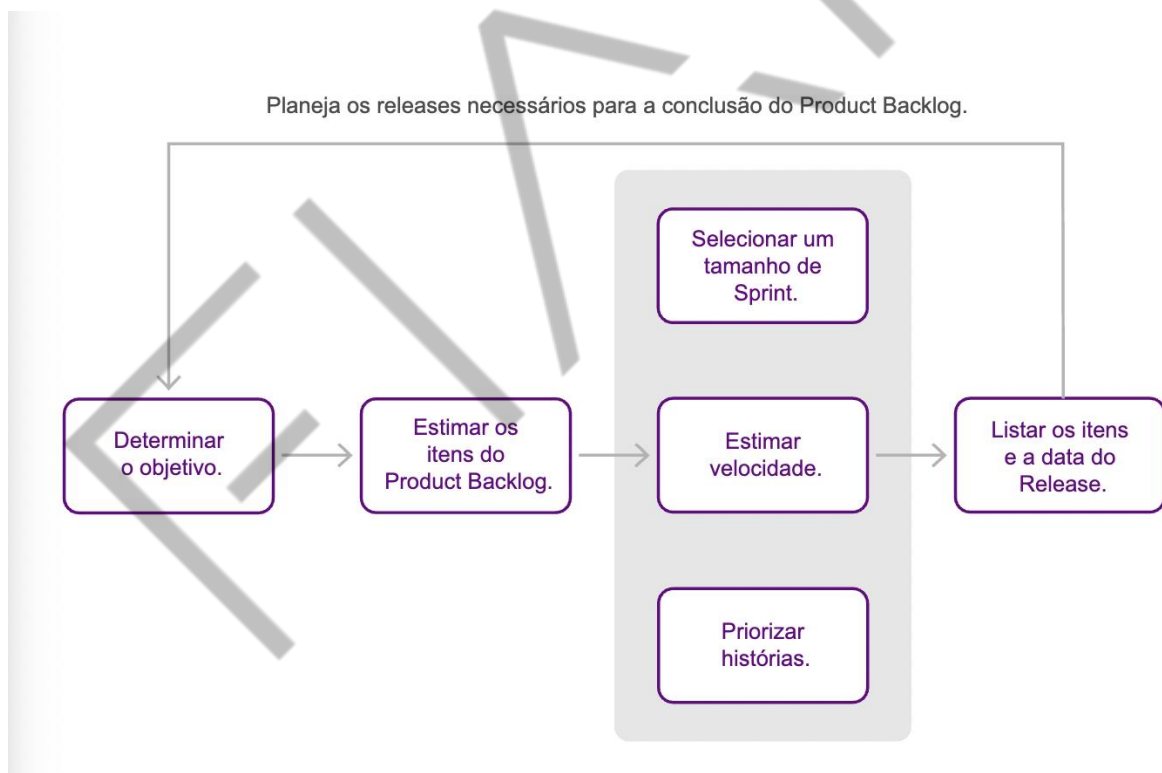


Figura 10 – Sequência de Planejamento de releases
Fonte: Scrum Guide®, adaptado por FIAP (2019)

Como o *Product Backlog*, o Plano de *Releases* não é um plano estático. Ele pode ser alterado durante todo o ciclo do projeto à medida que a equipe for adquirindo conhecimento maior. Por isso, o Plano de *Releases* deve ser revisto e atualizado em intervalos regulares, por exemplo, após cada *Sprint*.

Podemos orientar o planejamento dos *releases* de duas formas:

1. Orientado ao escopo

A forma mais utilizada é a orientada ao escopo, na qual o *Product Backlog* priorizado e estimado é dividido de acordo com a meta do *release*. Lista as histórias necessárias para atingir a sua meta e define quantos *sprints* são precisos para cumprir o *release*.

Essa definição do número de *Sprints* é feita considerando o tamanho das histórias (soma) e a velocidade do time. Essa abordagem é exemplificada na Figura “Hierarquia do *release*”.

PLANEJAMENTO DE RELEASES					
R	#	Descrição	Prior.	Tam	Pts.
	18	Como supervisor da agência de locação, posso cadastrar, alterar, consultar e excluir os veículos para locação.	2	5	
	7	Como atendente, posso consultar os veículos disponíveis para locação.	3	8	
	10	Como atendente, posso cadastrar os clientes que vão fazer locação de veículos.	4	5	
	8	Como atendente, posso fazer locação de um veículo por km livre ou controlada.	5	13	
	9	Como atendente, preciso aprovar o crédito do cliente antes de fazer a locação do veículo.	5	8	
	20	Como atendente, posso fazer reservas de locações de veículos para clientes cadastrados.	5	0	39

Quadro 9 – Exemplo de Plano de *Releases* orientado ao escopo

Fonte: Elaborado pelo autor (2019)

No exemplo do Quadro “Exemplo de Plano de *Releases* orientado ao escopo”, temos um *release* com tamanho de 39 pontos. O tamanho do *release* é a somatória de pontos de todas as histórias para atingir a meta. Considerando hipoteticamente que a velocidade do time é de 13 pontos por *Sprint* de 2 semanas, serão necessários 3 *Sprints* para entregar esse *release*.

2. Orientado ao prazo

Na forma orientada ao prazo, o *Product Owner* determina o tempo em que deseja que um *release* seja entregue e dividem-se as histórias em *Sprints* para se enquadrarem no prazo definido.

Vamos supor que o *Product Owner* deseja um *release* a cada 4 semanas. Levando em conta que a velocidade do time é de 10 pontos por *Sprint* de 2 semanas, vamos entregar 20 pontos por *release*.

IMPORTANTE: Como determinar a velocidade é abordado no item Sequência de Planejamento de *releases*.

2.5 Cálculo da velocidade

Como já mencionado, para estabelecer quantos pontos de histórias o time é capaz de realizar durante um *Sprint*, precisamos fazer o cálculo da velocidade do time, ou seja, saber se as histórias descritas e priorizadas no *Product Backlog* pelo *Product Owner* cabem dentro do *timebox* do *Sprint* que está sendo planejado.

Algumas considerações...

Antes de entrar no cálculo da velocidade, vamos analisar alguns fatores importantes relacionados ao time e ao ambiente que devemos levar em conta nos cenários de projeto:

- Quando começa a usar *Scrum*, há um ciclo natural de aprendizado do time e de todos os envolvidos;
- Não espere que um *Scrum team* vá começar a trabalhar com o mais alto desempenho possível desde o início;

- A estrutura e o conhecimento do time melhoram à medida que executam os *Sprints*.

Segundo Tuckman, todo time passa pelas seguintes etapas: formação, conflito, acordo, desempenho e dispersão, o que está ilustrado na Figura “Modelo de Tuckman” em que cada time tem tempos diferentes para atingir a fase de desempenho.

Ao iniciar o projeto, um time é montado e entra na fase de formação. Em seguida, no dia a dia do trabalho, o time se ajusta, descobrindo as forças e fraquezas de cada membro e estabelecendo padrões e limites até atingir a fase de acordo. Somente após estas três fases, o time passa para a fase de desempenho. Ao final do projeto, entra a fase de dispersão do time e o ciclo recomeça.



Figura 11 – Modelo de Tuckman
Fonte: Elaborado pelo autor (2021)

Em estudos realizados, um time *Scrum* demora cerca de 3 a 4 *Sprints* para entrar em ritmo de desempenho e produzir o seu melhor.

Agora, sim! Calculando a velocidade.

Existem diversas formas de calcular a velocidade de um time *Scrum*. Vamos apresentar uma muito utilizada no mercado e fácil de utilizar.

Neste método, o primeiro passo é definir qual é o percentual de dedicação de cada membro do time *Scrum* durante o *Sprint*, por exemplo: 100% dedicado, 50% dedicado. Esse nível é chamado de fator de foco. Podemos dizer que seria o percentual de produtividade do time, dado pela fórmula:

Fator de Foco = Número de pontos produzidos / Capacidade de produção do time

Em que:

Número de pontos produzidos = quantos pontos foram produzidos no *Sprint*.

Capacidade de produção do time = Somatória do número de dias de cada membro do time durante o *Sprint*.

IMPORTANTE:

Quando NÃO se tem histórico ou algum *Sprint* executado pelo time, o fator de foco inicial pode variar entre 50% e 70% para um novo time SCRUM. O *Scrum Master* deve avaliar o melhor fator para aplicar de acordo com as características do time. Sugerido é usar o fator de foco inicial em 60%.

Exemplo:

Vamos considerar um time com três recursos, sendo dois 100% e um 50% dedicado ao projeto. Em um *Sprint* de 2 semanas, foram produzidos 18 pontos.

Qual é o fator de foco?

EQUIPE	RECURSOS	ALOCACÃO	
3 pessoas	Huguinho	100%	100% / 10 dias = 10 dias/homem (d/h)
	Zezinho	100%	100% / 10 dias = 10 dias/homem (d/h)
	Luizinho	50%	50% / 10 dias = 5 dias/homem (d/h)

Somando-se a capacidade de produção dos recursos do projeto, temos:

10 + 10 + 5 = 25 d/h de capacidade de produção

Aplicando a fórmula:

Fator de Foco = número de pontos produzidos / capacidade produção time

Temos:

$$\text{Fator de foco} = 18 / 25 = 0,72 \Rightarrow \text{fator de foco} = 72\%$$

Para calcular a velocidade, usamos a fórmula:

$$\text{Velocidade} = \text{Capacidade de produção do time} \times \text{Fator de foco}$$

Dessa forma, vamos calcular a velocidade no cenário abaixo, considerando o fator de foco calculado anteriormente (72%):

EQUIPE	RECURSOS	ALOCÇÃO	
3 pessoas	Huguinho	100%	100% / 10 dias = 10 dias/homem (d/h)
	Zezinho	100%	100% / 10 dias = 10 dias/homem (d/h)
	Luizinho	100%	100% / 10 dias = 10 dias/homem (d/h)

$$\text{Velocidade} = \text{Capacidade de produção do time} \times \text{Fator de foco}$$

$$\text{Velocidade} = 30 \text{ d/h} \times 72\% = 22 \text{ pontos por Sprint.}$$

Sendo assim, nenhum *Sprint* planejado pode ter mais do que 22 pontos.

Outro exemplo:

Um time novo com três pessoas alocadas 100% do tempo para a realização de um *Sprint* de 2 semanas (10 dias), com fator de foco igual a 60%.

EQUIPE	RECURSOS	ALOCÇÃO	
3 pessoas	Huguinho	100%	100% / 10 dias = 10 dias/homem (d/h)
	Zezinho	100%	100% / 10 dias = 10 dias/homem (d/h)
	Luizinho	100%	100% / 10 dias = 10 dias/homem (d/h)

A capacidade produtiva do time é calculada da seguinte forma:

Capacidade produtiva = Número de dias do *Sprint* x Número recursos dedicados

Portanto, temos:

Capacidade produtiva = 10 dias x 3 recursos = 30 h/d

Velocidade = 30 h/d x 60% / 100

Velocidade = 18 pontos por *Sprint*.

Resumo das fórmulas de cálculo da velocidade.

Velocidade = Capacidade de produção do time x Fator de foco

Capacidade produção = Número de dias *Sprint* x Número recursos

Fator de Foco = Número de pontos produzidos / Capacidade de produção do time

O cálculo da velocidade deve ser feito ao final de cada *Sprint*. Esse recálculo é necessário para que o time refine seu planejamento e melhore a capacidade produtiva do time. Neste recálculo, devem ser considerados os pontos que foram construídos.

IMPORTANTE:

Mais detalhes para escalar o SCRUM, consulte o *Nexus Guide* no site

www.scrum.org.

REFERÊNCIAS

BEZERRA, E. Princípios de análise e projeto de sistemas com UML. 2. ed. São Paulo: Campus, 2011.

BOOCH, G.; RUMBAUGH, J.; JACBSON, I. **UML – Guia do Usuário**. 2. ed. São Paulo: Campus, 2006.

GUEDES, G. T. A. **UML 2 – Uma abordagem prática**. 2. ed. São Paulo: Novatec, 2011.

IEEE 610.12-1990, IEEE. **Standard Glossary of Software Engineering Terminology**. IEEE, 1990.

IIBA. **Um Guia para o Corpo de Conhecimento de Análise de Negócios, Guia BABOK, versão 2.0**. Traduzido pelo IIBA Capítulo São Paulo, Canadá. International Institute of Business Analysis, 2011.

KOSCIANSKI, A.; SOARES, M. S. **Qualidade de software**. 2. ed. São Paulo: Novatec, 2007.

LARMAN, C. **Utilizando UML e padrões**. 3. ed. Porto Alegre, RS: Bookman, 2007.

OAKLAND, J. S. **Gerenciamento da Qualidade Total**. São Paulo: Nobel, 1998.

PFLEEGER, S. L. **Engenharia de Software – Teoria e Prática**. 2. ed. São Paulo: Prentice Hall, 2004, pp. 111-156.

PRESSMAN, R. S. **Engenharia de Software**. São Paulo: Makron Books, 2011.

SOMMERVILLE, I. **Engenharia de Software**. 9. ed. Tradução de Maurício de Andrade São Paulo: Pearson, 2011.