DEVOPS & AGILE CULTURE

METODOLOGIA E MELHORES PRÁTICAS DE DESENVOLVIMENTO DE SOFTWARE

PEDRO IVO CORREIA DE ARAÚJO

LISTA DE FIGURAS

SUMÁRIO

2 METODOLOGIA E MELHORES PRÁTICAS DE DESENVOLVIMENTO DE SOFTWARE

Na era moderna, a tecnologia da informação e seus respectivos *softwares*, serviços e aplicativos desempenham um papel cada vez mais predominante no dia a dia da sociedade, e, para o desenvolvimento dos mesmos, utilizamos metodologia de desenvolvimento de software, que é um conjunto de métodos coordenados para se alcançar um objetivo, através de uma dinâmica iterativa visando à qualidade e à produtividade dos projetos. Mas antes de abordar as metodologias atuais, vamos conhecer a história e sua influência nas metodologias modernas.

Após a Segunda Guerra Mundial, a japonesa **Toyota Motors** identificou a necessidade de tornar a produção mais eficiente e, para tal, desenvolveu um sistema de produção para fornecer melhor qualidade, custo e menor tempo de entrega. Visando diminuir o desperdício, utilizou-se do princípio de *jidoka* (capacidade de detectar uma anormalidade rapidamente e interromper imediatamente o trabalho, evitando desperdícios, otimizando o processo e garantindo a qualidade do produto final) e o conceito de *just-in-time* (produzir o que é necessário, no momento necessário e na quantidade necessária), através da prática *kanban* (termo de origem japonesa que significa literalmente “cartão” ou “sinalização”) que utiliza cartões de informações para controlar a produção de acordo com a necessidade. Todo esse processo industrial ficou conhecido como TPS (*Toyota Production System*) e possui grande influência nas metodologias que iremos abordar neste capítulo.

Saindo do modelo de desenvolvimento industrial, no modelo de desenvolvimento de software algo muito utilizado foi o Modelo em Cascata (*Waterfall Mode*), uma sequência de fases no desenvolvimento de software que são: análise e definição dos requisitos, design, desenvolvimento, testes e manutenção. Porém esse modelo apresenta alguns desafios como:

- **Escopo engessado no começo do projeto:** Visto que mudanças no mundo acontecem cada vez mais rápido, uma mudança na demanda poderia tornar seu produto irrelevante até o final dele.

- **Desperdício com funcionalidades desnecessárias:** Já que tudo necessita ser definido no começo, o máximo de funcionalidade será solicitada, mesmo sem a certeza da necessidade das mesmas.
- **Entregas lentas:** Apenas após um longo período de planejamento, execução e testes de grandes pedaços, é que veremos alguma entrega e que, por ventura, pode ser algo de pouco valor no momento.
- **Baixa transparência:** Manter o envolvimento de todos apenas com processos e extensas documentações não é algo motivador, e provavelmente algumas pessoas não vão querer ler tudo apenas para executar parte do trabalho, logo não vão estar cientes de todo o projeto com clareza.

Para trabalhar com o desenvolvimento de *software* de forma mais adequada, na primavera de 2000 um grupo de líderes da comunidade do *Extreme Programming* (XP – metodologia que abordaremos mais à frente) se reuniram para discutir o processo de desenvolvimento com XP; no decorrer da reunião chegaram a um consenso sobre pontos importantes referentes ao desenvolvimento de software e decidiram escrever um documento que serviria como guia aos novos processos de desenvolvimento ágil de software, o famoso **Manifesto Ágil**.

2.1 Manifesto Ágil

O Manifesto Ágil possui 4 valores e 12 princípios. Vejamos a seguir os valores, e note nas frases que mesmo havendo valores nos itens à direita, a valorização é maior nos itens da esquerda:

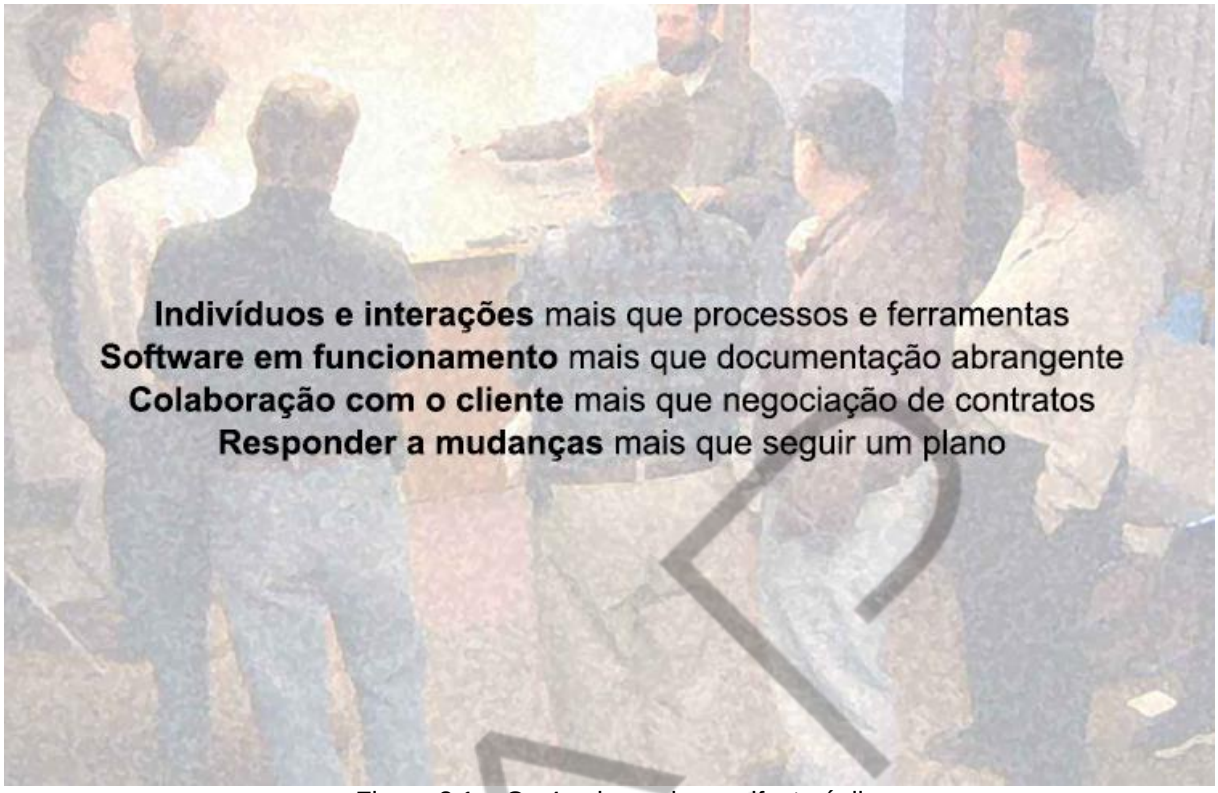

Figura 2.1 – Os 4 valores do manifesto ágil
Fonte: http://www.manifestoagil.com.br (2019)

2.1.1 Valores

- **Indivíduos e iterações mais que processos e ferramentas:** O desenvolvimento de software é uma atividade humana e que problemas de comunicação podem ser resolvidos com qualidade na interação entre os envolvidos. Os processos e ferramentas devem cumprir seu papel de forma simples e útil, mas sem “passar por cima” das pessoas.
- **Software em funcionamento mais que documentação abrangente:** O software funcionando é um indicador de sucesso é e o que o cliente espera, a documentação do mesmo deve ser algo que agregue valor e com informações necessárias.
- **Colaboração com o cliente mais que negociação de contratos:** A colaboração com o cliente é algo fundamental para o sucesso do desenvolvimento do software, o trabalho em equipe e a tomada de decisões em conjunto contribuem para alcançar um único objetivo.

- **Responder a mudanças mais que seguir um plano:** O desenvolvimento de software é algo complexo e possui alta incerteza, nessas situações devemos aprender com as situações ocorridas e adaptar o plano sempre que possível.

2.1.2 Princípios

- Nossa maior prioridade é satisfazer o cliente através da entrega contínua e adiantada de software com valor agregado
- Aceitar mudanças de requisitos, mesmo no fim do desenvolvimento. Processos ágeis se adaptam a mudanças para que o cliente possa tirar vantagens competitivas
- Entregar frequentemente software funcionando, de poucas semanas a poucos meses, dando preferência à menor escala de tempo
- Pessoas de negócio e desenvolvedores devem trabalhar diariamente em conjunto por todo o projeto
- Construir projetos em torno de indivíduos motivados, dando a eles o ambiente e o suporte necessário e confiando neles para fazer o trabalho
- O método mais eficiente e eficaz de transmitir informações para e entre uma equipe de desenvolvimento é através de conversa face a face
- Software funcionando é a medida primária de progresso
- Os processos ágeis promovem desenvolvimento sustentável. Os patrocinadores, desenvolvedores e usuários devem ser capazes de manter um ritmo constante indefinidamente
- Contínua atenção à excelência técnica e bom design aumentam a agilidade
- Simplicidade: a arte de maximizar a quantidade de trabalho não realizado é essencial

- As melhores arquiteturas, requisitos e designs emergem de times auto organizáveis
- Em intervalos regulares, a equipe reflete sobre como se tornar mais eficaz e então refina e ajusta seu comportamento de acordo

Com o Manifesto Ágil escrito, foi criada a *Agile Alliance*, uma organização sem fins lucrativos que procura disseminar o conhecimento e promover discussões sobre os métodos ágeis, como o Scrum, Kanban e XP, que carregam em suas bases os valores e princípios do Manifesto Ágil. Muitas pessoas acham que as metodologias ágeis são “ágeis” por que são rápidas e entregam funcionalidades o quanto antes, e na verdade a agilidade simboliza a capacidade de se adaptar a mudanças, a velocidade nas entregas é uma consequência que acontece quando utilizamos adequadamente as metodologias que vamos abordar a seguir.

2.2 Framework Scrum

O **Scrum** é um framework simples para gerenciar projetos complexos, através de uma dinâmica, papéis e cerimônias no decorrer de todo o ciclo. O Scrum foi concebido na década de 1990, e o termo foi inspirado em uma jogada de rúgbi em que 8 jogadores se reúnem para retirar os obstáculos à frente do jogador que correrá com a bola, utilizando-se do próprio corpo como principal armadura para livrar o companheiro do time, para que assim ele consiga avançar o máximo possível e marcar mais pontos. Essa analogia descreve a importância das equipes e seu trabalho em conjunto para resolver os mais variados problemas.

O Scrum não restringe sua aplicação apenas para a área de TI, ele pode ser aplicado em projetos diversos que possuem complexidade. A abordagem incremental e iterativa é utilizada fortemente no Scrum, onde a cada determinado espaço de tempo, uma funcionalidade do produto que faz parte de um todo é entregue. Vamos detalhar agora as características desse framework, mas, antes, é necessário internalizar os pilares do Scrum!

2.2.1 Pilares

- **Transparência:** Todos possuem o conhecimento dos requisitos, processos e do andamento do projeto
- **Inspeção:** Durante todo o tempo é inspecionado o que está sendo feito, através de reuniões diárias ou no momento de revisão
- **Adaptação:** Conforme as mudanças vão acontecendo, tanto o processo quanto o produto podem sofrer adaptações, desde que preservados os valores e práticas ágeis

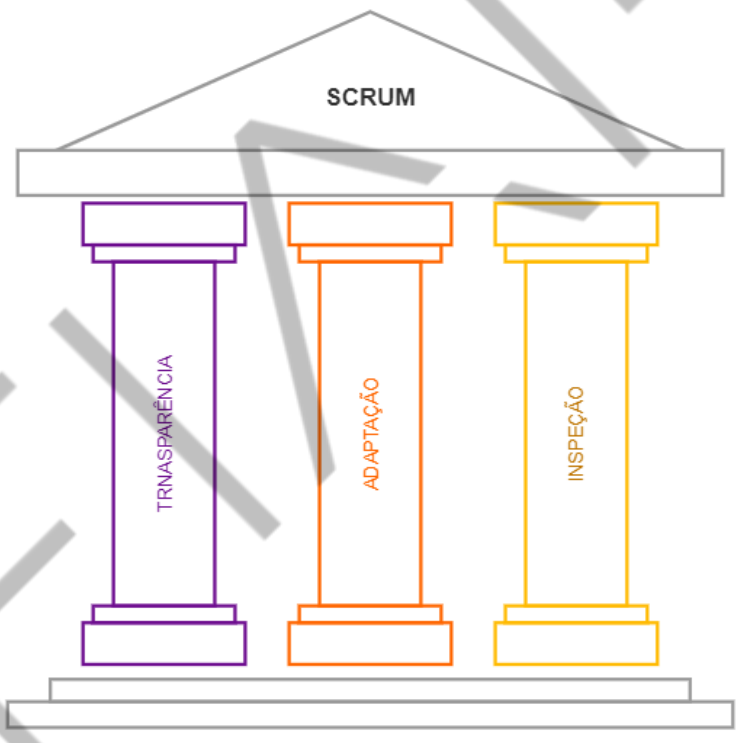

Figura 2.2 – Os pilares do SCRUM
Fonte: Google Images (2019)

2.2.2 Papéis Fundamentais

Para a realização do Scrum, são necessários 3 papéis fundamentais, sem esses papéis, não temos Scrum:

- **Product Owner (PO):** Como o nome já diz, é o dono do produto, o único responsável por decidir quais os recursos e funcionalidades o produto deve

ter e qual a ordem de prioridade que deve ser desenvolvido. Também mantém e comunica uma visão clara do que o *Time Scrum* está trabalhando no produto, sendo um ponto de intersecção entre a área de negócio e a área de desenvolvimento. É muito importante que o PO saiba o que é o produto que quer desenvolver e tenha autonomia para tal, pois uma vez que ele depende de outros para tomar decisões ou tocar o seu produto, acaba virando um "*PO Proxy*" que não sabe o que tem que fazer, e só fica perguntando ou dependendo de terceiros para desenvolver o seu produto.

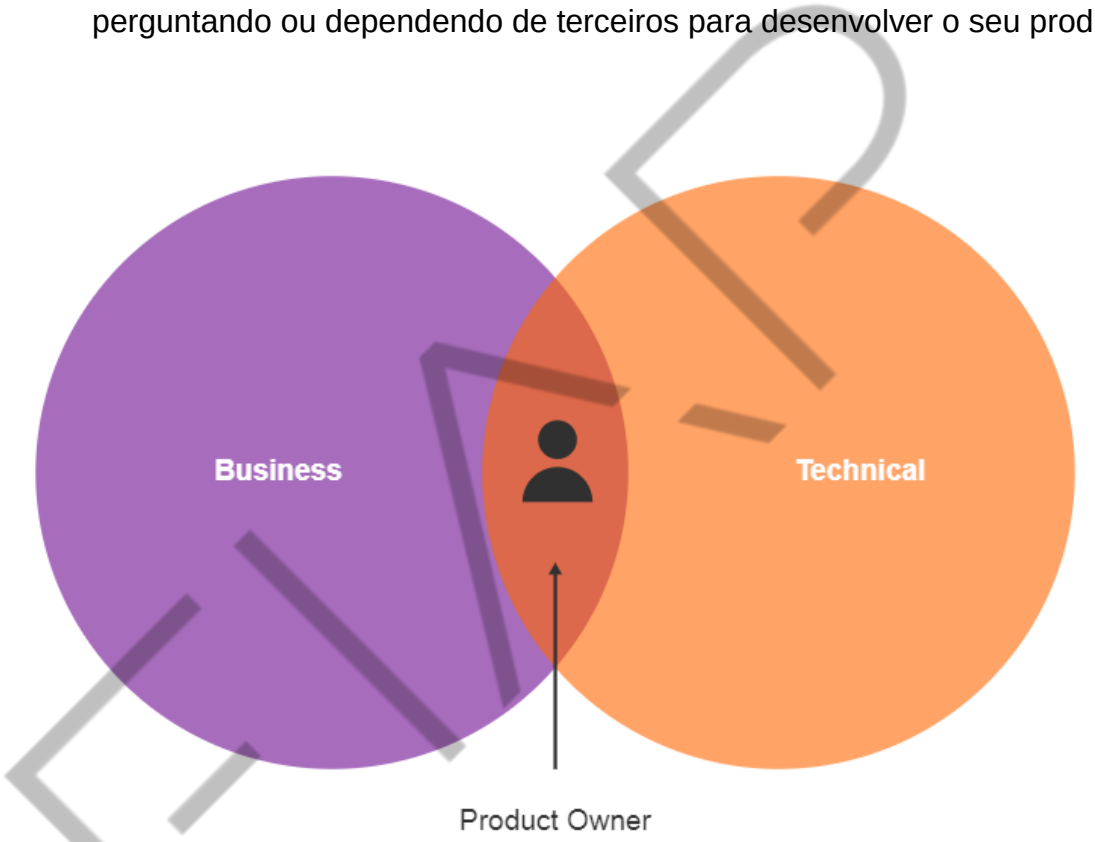

Figura 2.3 – Papéis fundamentais do Scrum - Product Owner
Fonte: Google Images (2019)

- **Scrum Master (SM):** Responsável por ajudar os envolvidos a entender a cultura ágil com todos os seus valores e princípios, além de todas as práticas do framework Scrum. Ele atua como um líder do processo e ajuda os demais a desenvolver sua própria forma de trabalho com Scrum. O Scrum Master também é um facilitador, contribuindo na resolução de conflitos e remoção de impedimentos, protegendo o time contra interferências externas que possam acontecer. Existe uma linha tênue que o Scrum Master deve se atentar e que é muito comum no mercado, a

confusão do Scrum Master com um chefe, que de fato ele não deve ser, e assim como o PO, ele faz parte do time. O outro ponto de atenção é referente a ser um facilitador e não um bloqueador, muitos Scrum Masters acabando indiretamente se tornando um bloqueador ao proteger demais os seus times, o Scrum Master deve procurar o desenvolvimento do time para que os mesmos sejam independentes.

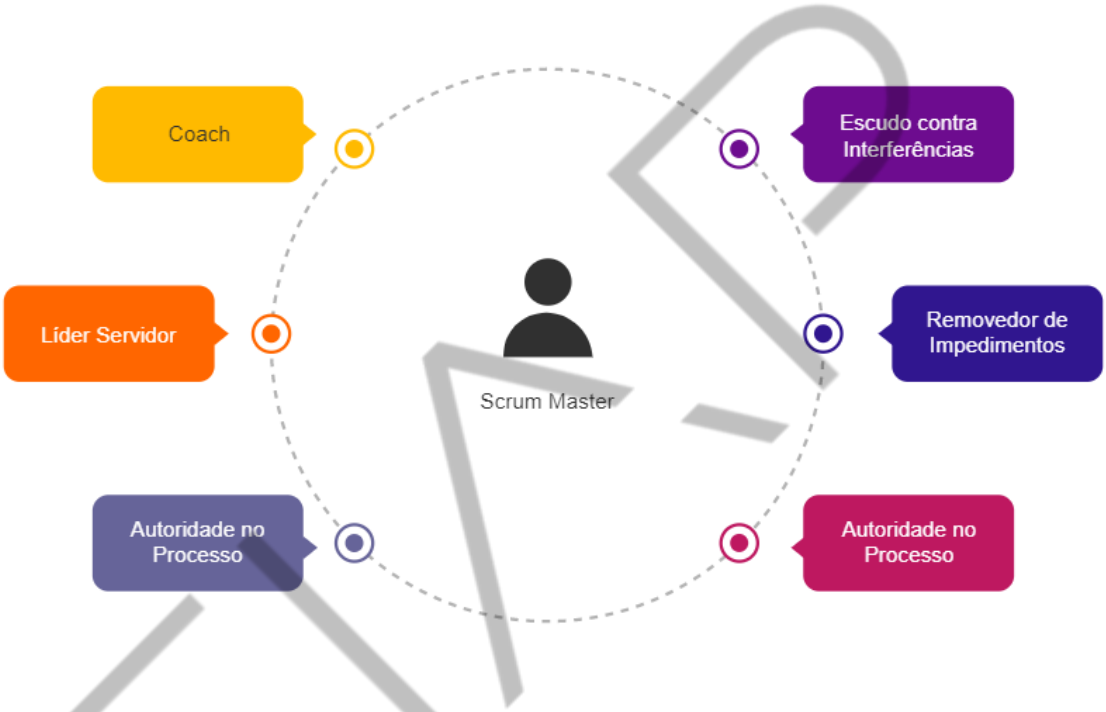

Figura 2.4 – Papéis fundamentais do Scrum – Scrum Master
Fonte: Google Images (2019)

- **Time Scrum:** Time de desenvolvimento que possui vários membros com diferentes funções, que possuem as habilidades necessárias para o desenvolvimento do software, é o Time Scrum que decide como fazer para se alcançar o objetivo do desenvolvimento. Normalmente o time Scrum possui entre 4 e 8 pessoas, em que os mesmos colaboram entre si para se tornarem um time auto-organizado e multidisciplinar. Quando são necessárias equipes maiores para o desenvolvimento de um projeto, em vez de montar uma única equipe Scrum com, por exemplo, 50 pessoas, o ideal seria montar vários times Scrum menores.

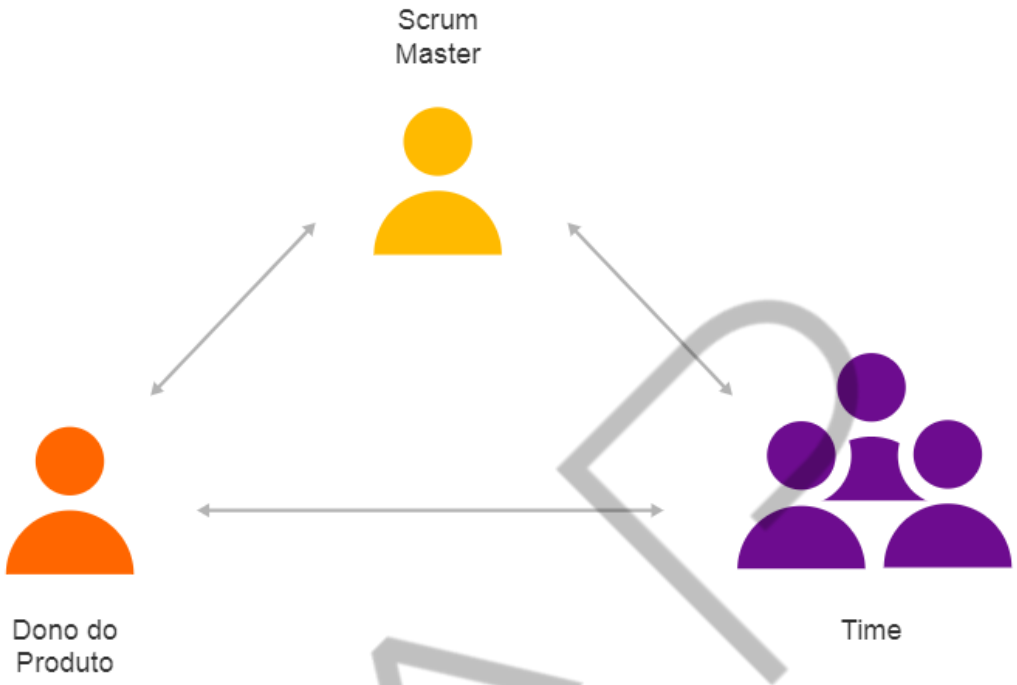

Figura 2.5 – Papéis fundamentais do Scrum
Fonte: Google Images (2019)

2.2.3 Valores

Assim como no Manifesto Ágil, o Scrum também possui seus próprios valores, que dão personalidades aos papéis:

- **Foco:** Todos estão focados em um único objetivo, em vez de cada um trabalhar em diversas partes isoladas para entregar tudo junto no final. O time deve manter o foco em pequenas partes por vez, essas partes devem estar alinhadas com a priorização do *Product Owner*, em que o mesmo estará focado nas necessidades mais prioritárias do seu produto. Para manter o foco entre ambas as partes, o *Scrum Master* facilita o meio de campo e ajuda na remoção dos impedimentos.
- **Coragem:** O time precisa ter coragem para enfrentar os desafios mais diversos, independentemente da dificuldade, e sempre fazer as coisas do modo mais correto possível. No decorrer do tempo, imprevistos acontecerão

e a coragem envolve aceitar as mudanças, errar, corrigir o erro e aprender com o mesmo o mais rápido possível.

- **Comprometimento:** Todos os papéis possuem seu comprometimento, o Time Scrum em realizar o que foi acordado, o *Scrum Master* em "girar a roda" do framework Scrum junto ao time, e o *Product Owner* em priorizar as necessidades de negócio e tocar o produto.
- **Respeito:** Os membros dos times respeitam um ao outro e se sentem confortáveis para interagir, ouvir e respeitar as opiniões, reconhecendo e entendendo as diferenças entre as diversas personalidades dentro do grupo, promovendo assim um ambiente saudável para o trabalho. O *Product Owner* respeita as decisões técnicas do Time Scrum, e em contrapartida o Time Scrum respeita as decisões de negócio vindas pelo *Product Owner*.
- **Abertura:** Todos do time estão abertos aos trabalhos e desafios que estão por vir, também envolve a abertura de oferecer e receber feedbacks, dialogar sobre situações e criar visibilidade dos problemas acontecidos, além da clareza do trabalho realizado e da performance do mesmo.

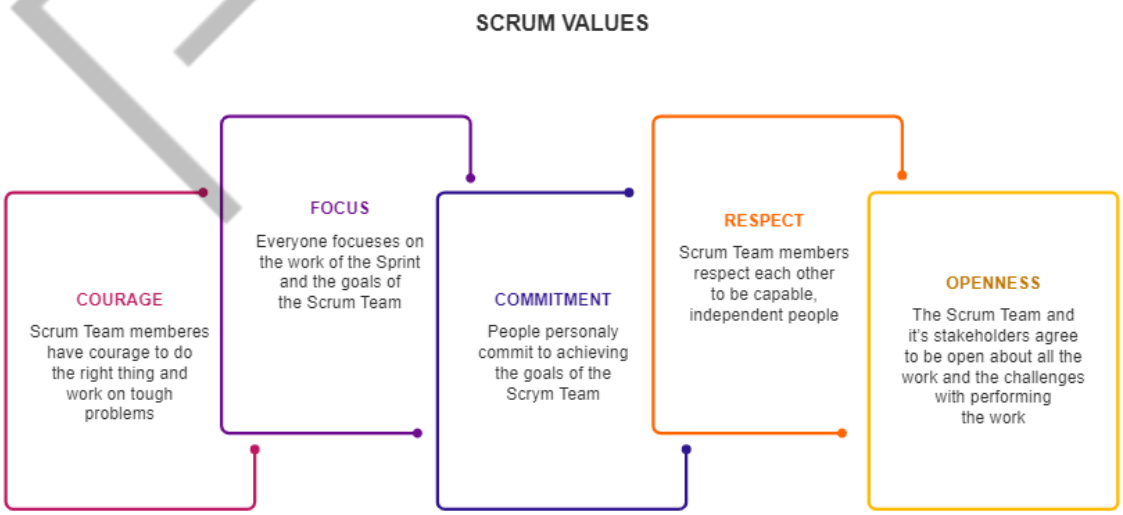

Figura 2.6 – Valores do Scrum
Fonte: https://www.scrum.org/resources/scrum-values-poster (2019)

2.2.4 Princípios

Os princípios do Scrum completam os valores e são fundamentais para o bom funcionamento do framework, eles são:

- **Controle dos processos empíricos:** As decisões no Scrum são tomadas baseadas na observação e experimentos, através da transparência, inspeção e adaptação no decorrer dos ciclos.
- **Auto-organização:** Os times são auto-organizáveis e possuem a responsabilidade compartilhada, conseguem atuar ponta a ponta no desenvolvimento do produto, entregando assim um maior valor.
- **Colaboração:** O trabalho colaborativo defende um processo de criação de valor compartilhado, com times trabalhando em conjunto para atingirem melhores resultado.
- **Priorização baseada em valor:** A priorização feita pelo *Product Owner* considera valor, risco, incerteza e dependências, resultando em entregas que fornecem o maior valor de negócio em menor tempo possível.
- ***Time-boxing*:** O conceito do *time-boxing* propõe a fixação de um determinado período de tempo para cada processo e atividade no ciclo do Scrum, garantindo assim que o time utilize da melhor maneira o seu tempo, e não desperdice tempo e energia em um trabalho no qual tenha pouco conhecimento ou não promova tanto retorno.
- **Desenvolvimento iterativo:** O processo iterativo permite que os times aprendam com o decorrer dos ciclos, promovendo refinamento a cada iteração, devido ao modelo de entrega de forma incremental.

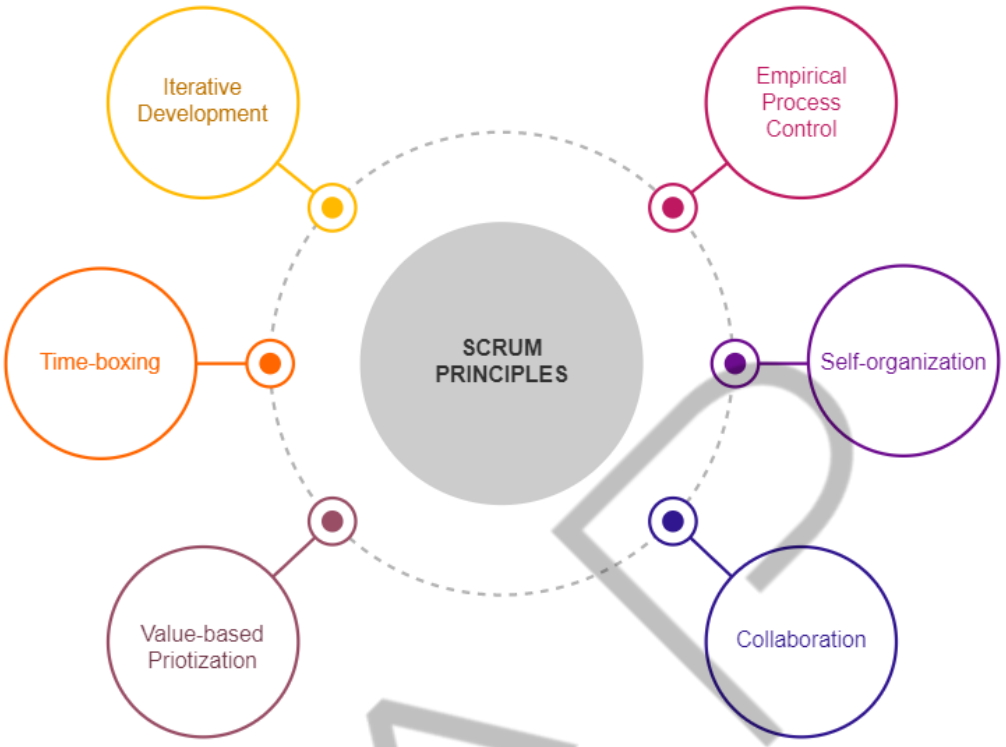

Figura 2.7 – Princípios do Scrum
Fonte: Google Images (2019)

2.2.5 Dinâmicas, cerimônias e artefatos

No Scrum tudo começa com a visão de um produto, em que o *Product Owner* descreve o que ele quer do produto e aonde quer chegar, e, dada a visão macro, o *Product Owner* lista as funcionalidades necessárias, essa lista é o artefato **Product Backlog**.

Com o *Backlog* em mãos, o *Product Owner* realiza a priorização das atividades, seguindo o princípio da priorização baseada em valor. Cada funcionalidade do *Backlog* é definida como um artefato de ***User Stories*** (história de usuário), que possui como objetivo explicar a funcionalidade de forma simples, deixando claro quem é o usuário, o que ele quer e para qual objetivo.

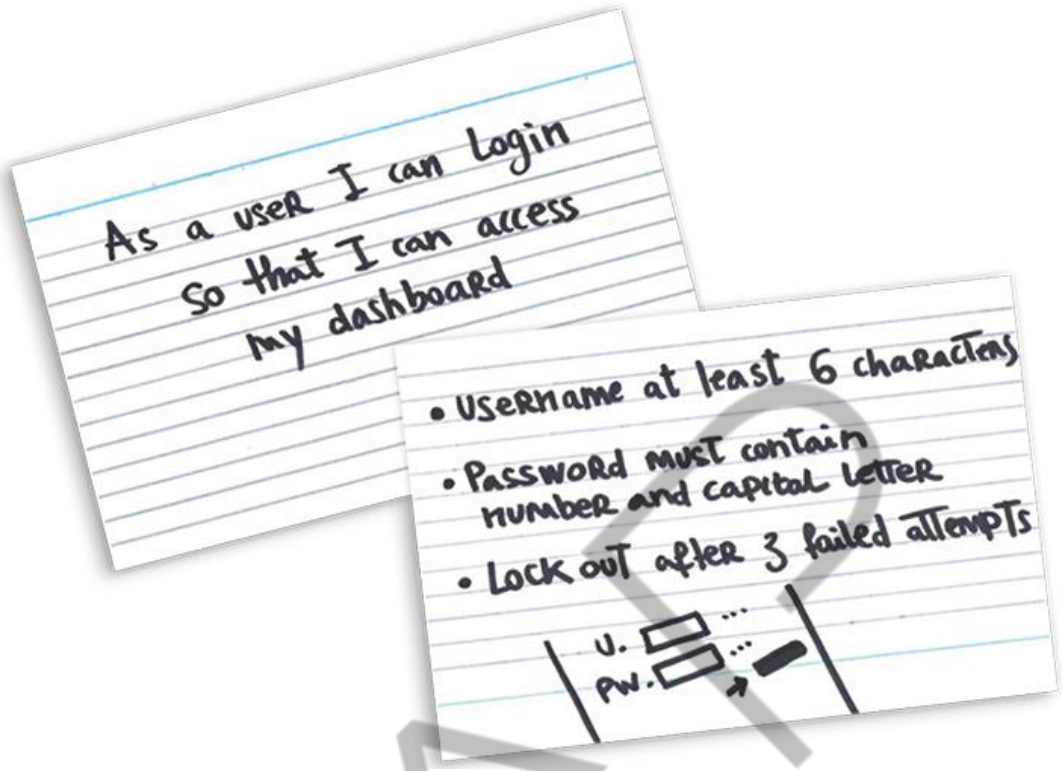

Figura 2.8 – User Story
Fonte: Google Images (2019)

Para a execução dos *User Stories* que estão no *Backlog*, no Scrum utilizamos o princípio de desenvolvimento iterativo através da **Sprint**, período de tempo limitado (*time-boxing*) de duração fixa (normalmente de 2 a 4 semanas) que cria algo de valor tangível para o cliente ou usuário do produto.

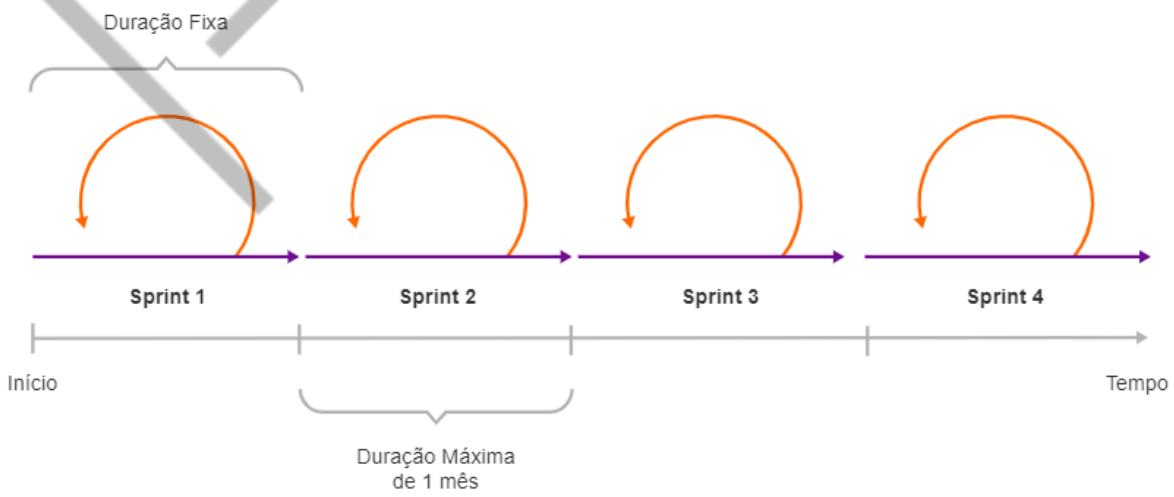

Figura 2.9 – Ciclos de Sprint
Fonte: Google Images (2019)

Antes de cada Sprint começar é realizada uma cerimônia chamada de ***Sprint Planning***, para determinar qual o objetivo da Sprint e quais *User Stories* serão desenvolvidas naquela Sprint. Durante o planejamento, o *Product Owner*, *Scrum Master* e Time Scrum precisam entrar em um consenso referente ao que cabe dentro da Sprint baseando-se na capacidade e velocidade da equipe. Caso existam dúvidas referentes aos tamanhos das atividades, utilizamos o ***Planning Poker*** para auxiliar na decisão**,** na qual, dado o escopo, os integrantes realizam votações com cartas para expor sua opinião referente ao tamanho daquela *User Story*.

Figura 2.10 – Planning Poker
Fonte: Google Images (2019)

Com o escopo do Sprint definido, o *Sprint Planning* é finalizado e, em seguida, o Time Scrum se reúne novamente para discutir sobre as atividades e ter uma visão mais detalhada do que precisa ser feito no Sprint que será iniciado. Caso o time perceba que não poderá cumprir com o esperado no Sprint, uma negociação com o *Product Owner* será realizada para ajustes do escopo. É responsabilidade do Time Scrum determinar o quanto ele é capaz de se comprometer a entregar.

Perceba que houve duas fases de planejamento, a primeira fase também é chamada de **Planning Estratégico** que define o que será feito, e a segunda fase de **Planning Tático** que define como será feito.

Com o Sprint definido, o desenvolvimento das atividades é iniciado, e todos os dias é realizada uma reunião de alinhamento rápido, essa cerimônia é chamada de **Daily Scrum**. No *Daily,* três perguntas fundamentais devem ser respondidas:

- O que eu fiz ontem que contribuiu para o atingimento da meta do Sprint?
- O que eu farei hoje para contribuir para o atingimento da meta do Sprint?
- Existe alguma circunstância que impeça a mim ou ao time atingir a meta do Sprint?

O *Daily* deve durar no máximo 15 minutos e acontecer idealmente no mesmo horário, e todos os integrantes do Time Scrum devem comunicar suas atividades baseadas nas perguntas, o *Product Owner* pode participar da *daily* opcionalmente. É recomendado que o Daily seja realizado com todos de pé, com a intenção de fazer com que a reunião seja rápida, essa prática é chamada de ***Stand-Up Meeting***. Esse acompanhamento diário promove a transparência e o alinhamento constante entre o time e a área de negócio, todos conseguem visualizar como está o andamento do Sprint com relação ao objetivo acordado.

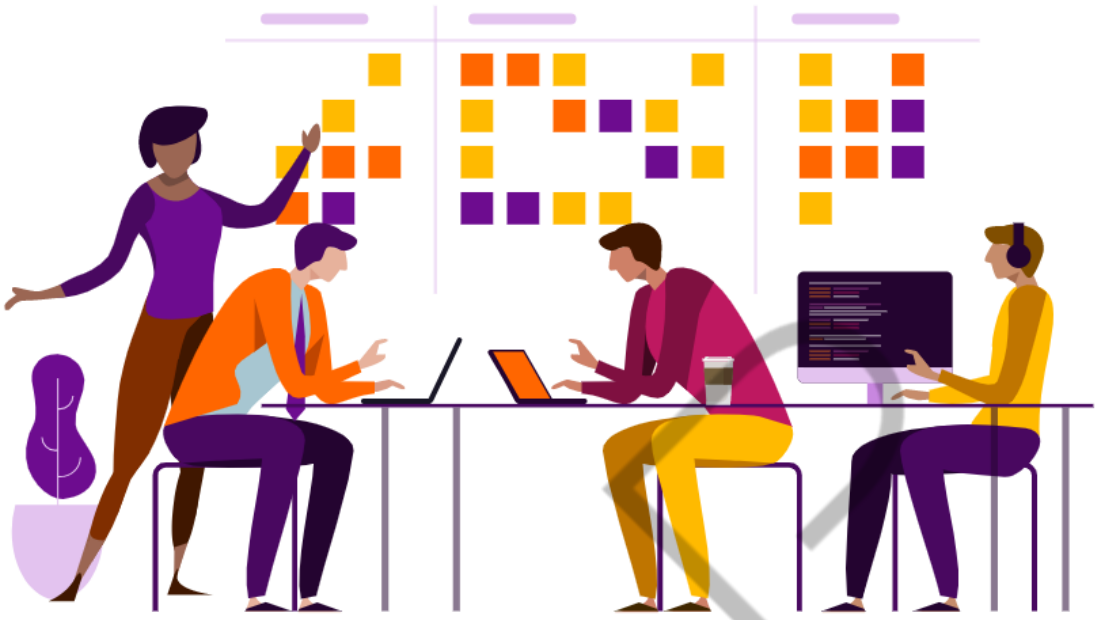

Figura 2.11 – Daily Scrum
Fonte: Google Images (2019)

Durante o período do Sprint, para saber se a atividade pode ser considerada concluída, utilizamos o conceito de ***Definition of Done*** **(DoD)**, que é um acordo formal do time que define claramente quais são os passos mínimos para a conclusão de um item potencialmente entregável.

Perto de finalizar o *Sprint*, é realizada uma cerimônia para apresentar o que foi feito, o ***Sprint Review***, com o intuito de verificar e adaptar o produto que está sendo desenvolvido. Por ser uma cerimônia mais para o final do *Sprint*, o ideal é que seja apresentado, durante o *Sprint,* o que está sendo desenvolvido ao *Product Owner* de forma mais informal, para se necessário realizar alguma adaptação se possível dentro do próprio *Sprint*.

Para finalizar, a cerimônia de ***Sprint Retrospective*** é realizada, enquanto o objetivo do *Sprint Review* é verificar necessidades de adaptações e melhorias no produto, o *Sprint Retrospective* tem como objetivo verificar necessidades de adaptações e melhoria no processo de trabalho como um todo. É importante que na reunião de retrospectiva sejam levantados os pontos positivos e a melhora (sempre focando nas causas raiz) da *Sprint* que passou.

Figura 2.12 – Papéis, atividades e artefatos do Scrum
Fonte: Google Images (2019)

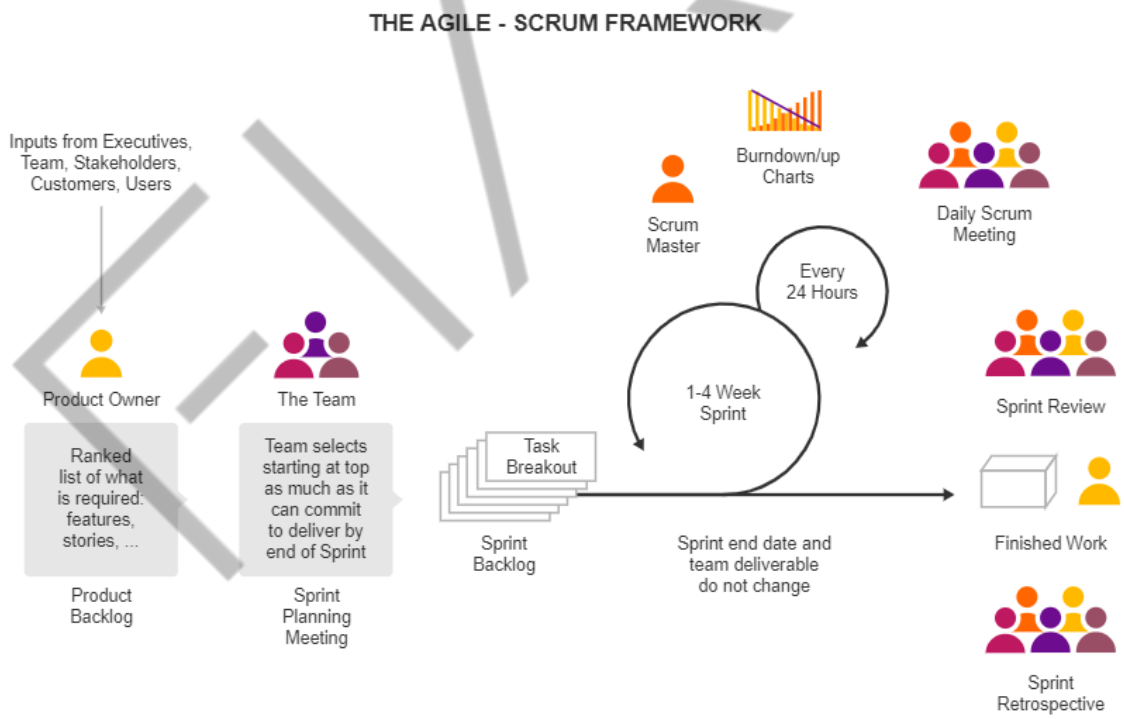

Figura 2.13 – O Framework Scrum
Fonte: Google Images (2019)

2.2.6 Caso de Uso

O framework Scrum é recomendado para os times que querem iniciar na cultura ágil e para projetos que têm uma janela de delivery constante, pois o mesmo possui uma metodologia bem estruturada ponta a ponta e promove um direcionamento do que deve ser feito, e, apesar disso, com o amadurecimento do time, o Scrum permite ser adaptado para melhor atender à necessidade. O Scrum utiliza elementos visuais para gestão das atividades, o quadro kanban, que veremos logo a seguir no método Kanban.

2.3 Método Kanban

Antes de estudarmos o método Kanban, vamos diferenciar a palavra Kanban com “K” maiúsculo e Kanban com “k” minúsculo. O **k**anban refere-se à utilização de um quadro com o fluxo de trabalho, com os respectivos cartões que representam as atividades, sua origem é japonesa e significa literalmente “cartão” ou “sinalização”, e o **K**anban refere-se ao método de desenvolvimento ágil criado em 2005, influenciado pelo Sistema Toyota de Produção.

No modelo tradicional corporativo, a transição de mudanças muitas vezes acontece de forma radical em um determinado período de tempo, porém como toda mudança, existe um período de adaptação no qual possíveis prejuízos poderão acontecer, até tal mudança se estabilizar e trazer o retorno desejável. Dependendo do nível da mudança radical proposta pela organização, a mesma não possui estrutura suficiente para suportar a mudança e pode decidir voltar atrás, ou, no pior cenário, até mesmo quebrar. Esse tipo de comportamento organizacional, em que determinada mudança do status quo dentro de um espaço de tempo limitado impactará a capacidade organizacional, é conhecido como Kaikaku. A

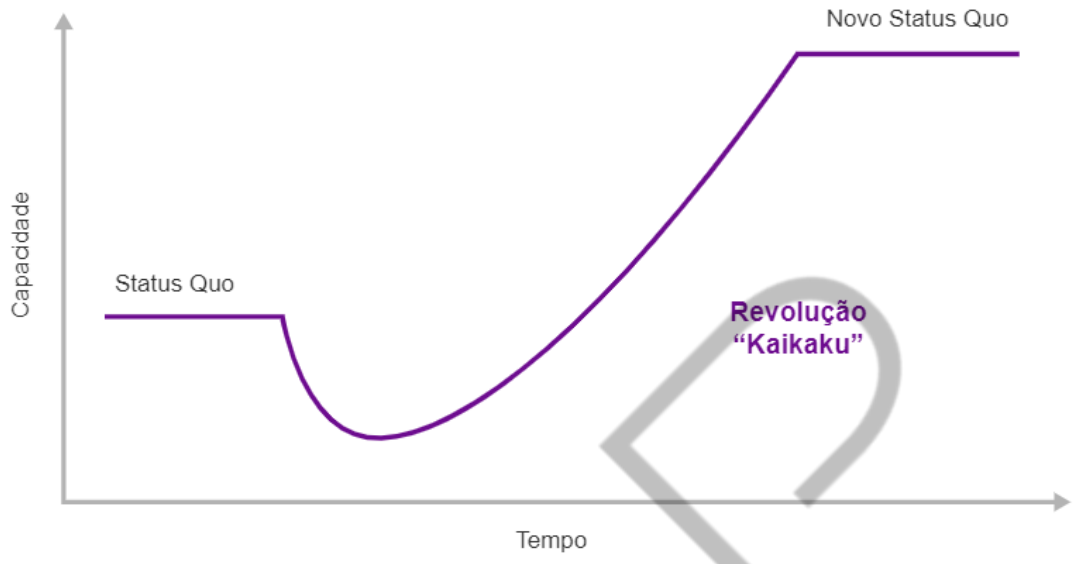

Figura 2.14"Gráfico de mudança Kaikaku" exemplifica o modelo.

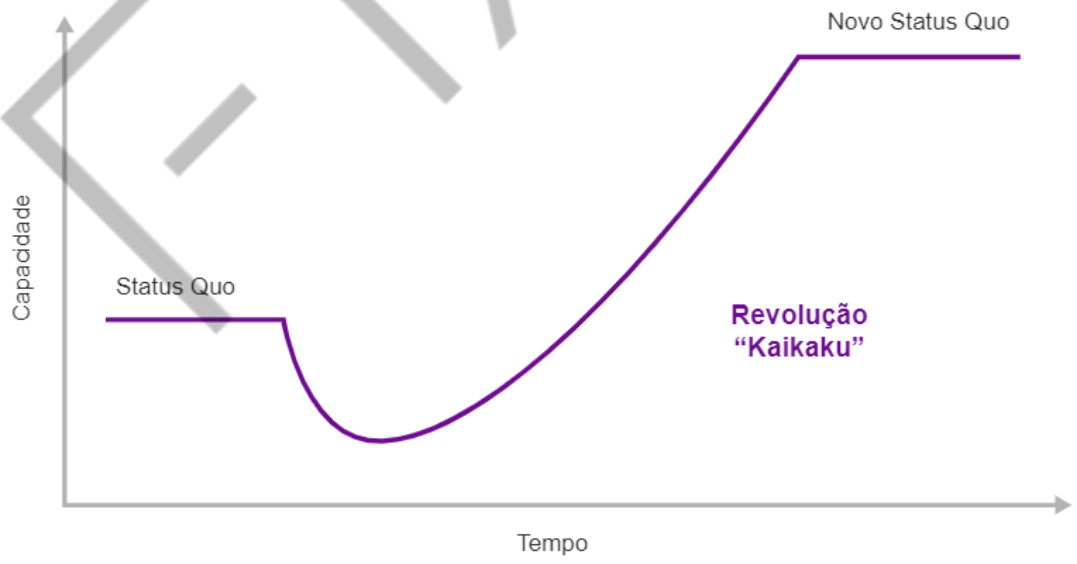

Figura 2.14 – Gráfico de mudança Kaikaku
Fonte: Google Images (2019)

Para trabalharmos mudanças de forma incremental, sem que o período de adaptação seja muito grande e diminuindo o risco da mudança, temos a abordagem de evolução Kaizen, que visa ciclos de iterações e melhorias contínuas referentes à mudança organizacional, com o objetivo de eliminar desperdícios e fazer “Hoje melhor do que ontem, amanhã melhor do que hoje!” Na Figura 2.15“Gráfico de mudança Kaizen versus Kaikaku” temos um gráfico comparativo entre as duas abordagens:

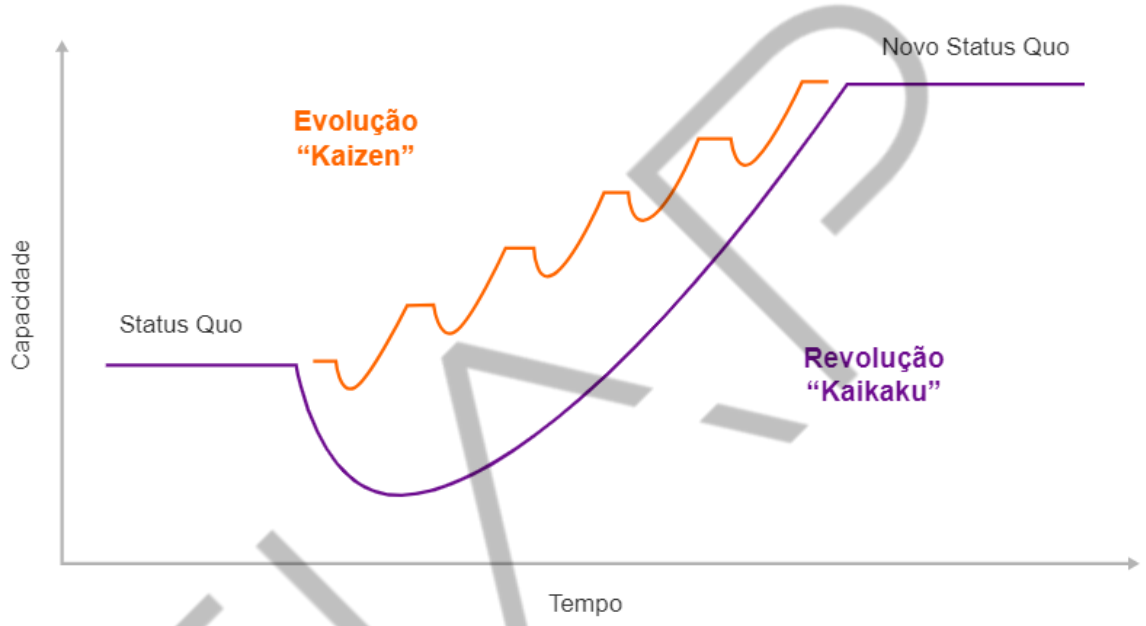

Figura 2.15 – Gráfico de mudança Kaizen versus Kaikaku
Fonte: Google Images (2019)

Quando falamos sobre metodologias de desenvolvimento ágil, o modelo de Evolução Kaizen é extremamente aderente e está intrínseco na filosofia das metodologias, e o método Kanban é um ótimo exemplo, ele é composto por um fluxo de valor no qual as etapas de trabalho são mapeadas em um quadro kanban, e as atividades andam da esquerda para a direita, e cada etapa de trabalho adiciona uma melhoria de valor naquele item, e ao chegar à direita do kanban esse item estará concluído e entregando um valor completo.

Vamos exemplificar o seguinte cenário, em que para desenvolver um *software* são necessárias as etapas de levantamento das atividades (*backlog*), *design*, desenvolvimento, testes e *deploy*; o quadro kanban ficaria assim:

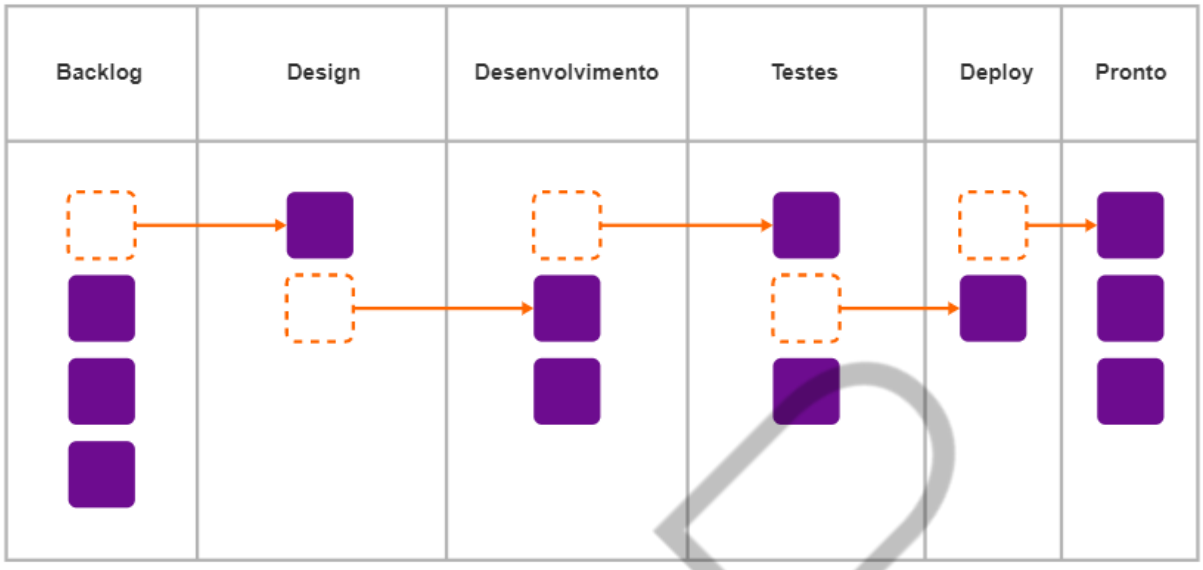

Figura 2.16 – Quadro kanban.
Fonte: Google Images (2019)

Mas para realmente conseguirmos trabalhar com o método Kanban, apenas o quadro não é o suficiente, precisamos incrementá-lo com alguns conceitos, primeiramente vamos falar sobre o sistema puxado e empurrado de trabalho.

O Sistema empurrado faz a produção ser baseada na demanda, onde dada uma quantidade de trabalho X, esse trabalho é “empurrado” para uma equipe que muitas vezes não tem a capacidade para executá-lo, exigindo 100% de ocupação do tempo para o desenvolvimento, para que um grande lote de trabalho seja entregue. Os principais efeitos de um sistema empurrado são a demora para entrega uma vez que todos estão 100% ocupados, e isso não significa 100% de produtividade, sobrecarga de trabalho, entregas em grandes lotes, e no pior dos cenários o *burnout* (exaustão prolongada e a diminuição do interesse em trabalhar) dos profissionais.

O método Kanban trabalha com o conceito de sistema puxado, em que os profissionais “puxam” o trabalho a ser feito quando existe capacidade para executá-lo, evitando sobrecargas, uma vez que a capacidade é limitada dentro de limites estabelecidos corretamente, para assim prover um equilíbrio entre a capacidade do time e a demanda que o mesmo pode executar. Para facilitar o entendimento, vamos inserir o conceito de sistema puxado no nosso quadro *kanban*, nas colunas de trabalho, vamos inserir mais duas etapas, o “fazendo” e o “pronto”, e, para cada etapa, vamos limitar a capacidade de trabalho possível em cada etapa:

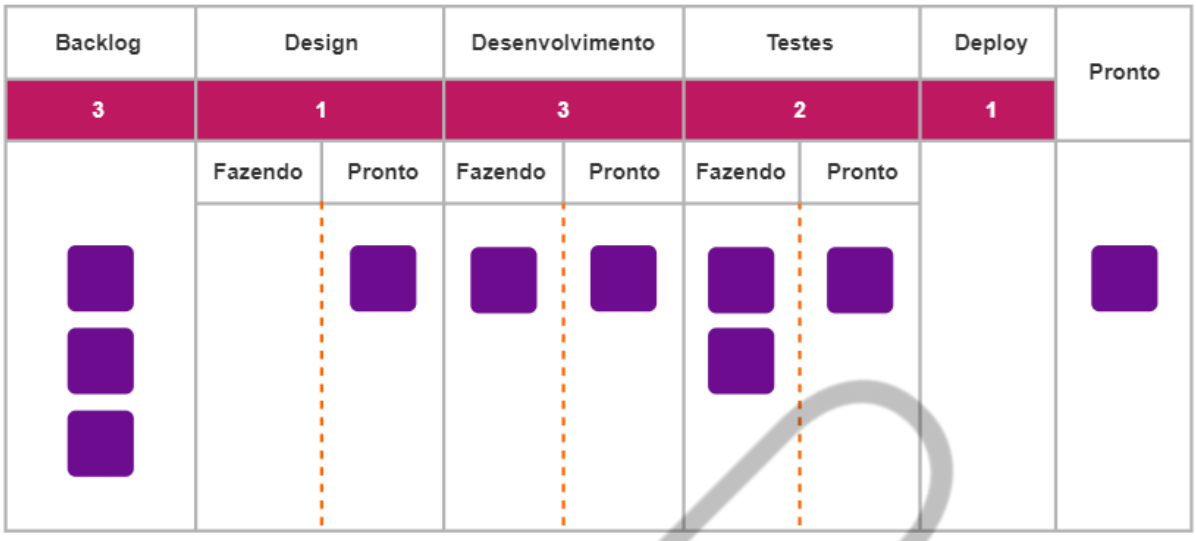

Figura 2.17 – Quadro kanban com sistema puxado.
Fonte: Google Images (2019)

Com o *kanban* configurado, vamos executar o sistema puxado, uma vez que exista trabalho no *backlog*, o profissional de *Design* "puxa" a atividade para a etapa de *Design* na coluna "Fazendo", desde que ele não esteja executando nenhuma atividade, quando ele finalizar essa atividade, moverá a atividade para a coluna de "Pronto", isso indica que o profissional de Desenvolvimento possui atividade a ser feita, e o profissional de Desenvolvimento "puxa" a atividade do "Pronto" de Design para o "Fazendo" do Desenvolvimento, e assim segue esse fluxo até a atividade chegar ao "Pronto" à direita do quadro *kanban*. Os limites em cada etapa servem para controlar o fluxo de trabalho e evitar sobrecarga, um item só pode transitar entre as etapas se houver espaço para o mesmo, consequentemente em determinado momento algum profissional poderá ficar ocioso, pois ele não poderá "puxar" trabalho, e isso não significa ficar sem trabalhar, em um sistema Kanban todos do time são estimulados a olhar para todo o fluxo e tentar contribuir com o valor da atividade mesmo fora da sua etapa, ou até mesmo usar o tempo para pensar em melhorias ou se desenvolver em algum ponto, o importante é que o profissional não fique apenas focado em sua etapa e respeite o limites do quadro kanban, para evitar a geração de estoque e sobrecarga. Esse fluxo de trabalho baseia-se na teoria das filas, que comprova que a ociosidade no fluxo da fila aumenta a vazão da mesma, e como os itens no quadro kanban são puxados apenas quando há capacidade disponível, o desperdício e a sobrecarga são evitados. Para estimular o fluxo do Kanban e respeitar os limites, uma frase é muito conhecida no mundo ágil que é: "**Pare de começar e comece a terminar"**.

Transitar de um sistema tradicional empurrado, que muitas vezes é predominante no mercado, para um sistema puxado é desafiador, porém traz muitas vantagens e ganhos para a organização. Assim como no Manifesto Ágil e no Scrum, o método Kanban também possui seus princípios e práticas que veremos a seguir:

2.3.1 Princípios

- Comece com o que você tem hoje
- Concorde em propor mudanças incrementais e evolucionárias
- Respeite o processo atual, papéis, responsabilidade e títulos
- Encoraje atos de liderança em todos os níveis

Seguir esses princípios ajudará a colocar em prática o método Kanban, pois identificando o que você tem hoje e respeitando os processos atuais, papéis e responsabilidades, não modificará inicialmente o status quo já existente, porém através de mudanças incrementais e evolucionárias, e com o encorajamento de todos, atuaremos na mentalidade do time para conseguir pequenos passos da melhoria contínua, porém de forma consistente.

2.3.2 Práticas

Para o sucesso da implementação do método Kanban, temos 6 práticas, sem elas, podemos dizer que ainda não estamos fazendo completamente o Kanban. As práticas são:

- **Visualizar o fluxo de trabalho:** Ao utilizar o quadro *kanban* é criado um modelo visual do fluxo de trabalho, para que seja possível identificar de forma clara o trabalho que está sendo feito pela equipe, e com o trabalho visível é possível identificar problemas e bloqueios para tomar ações, aumentar a comunicação, transparência e visibilidade de todo o processo
- **Limitar o trabalho em progresso:** Quando falamos de limite de trabalho em progresso, uma sigla é muito utilizada no Kanban, o WIP (Work in

Progress), e ao limitar o WIP o ritmo da equipe se torna equilibrado evitando que a mesma se comprometa com muito trabalho de uma só vez. Se o trabalho não for limitado, é muito provável que o time ainda esteja em um sistema empurrado

- **Gerenciar e medir o fluxo:** Com a visualização e o limite de trabalho estabelecidos, podemos gerenciar o fluxo de trabalho e, a cada ciclo completado, podemos coletar métricas e obter indicadores de problemas para otimizar ainda mais o trabalho
- **Tornar as políticas de processo explícitas:** O fluxo de trabalho deve ser claro para que todos tenham o entendimento, para isso, é fundamental tornar todas as políticas do processo de trabalho explicitas, pois não é possível melhorar algo que os envolvidos não entendem
- **Implementar ciclos de feedback:** Para evoluir o processo de trabalho são necessários ciclos de *feedback*, através de cerimônias específicas (ex.: retrospectivas) para retroalimentar os sistemas e permitir que o time se adapte as mudanças. Também é possível identificar se a funcionalidade entregue está com a qualidade esperada ao final do ciclo
- **Melhore colaborativamente:** Apesar de implementar todas as práticas anteriores, a melhoria contínua ainda não está acontecendo, o método Kanban continua incompleto. O modelo Kanban sugere a utilização do Kaizen para implementar mudanças colaborativamente de forma contínua, incremental e evolutiva

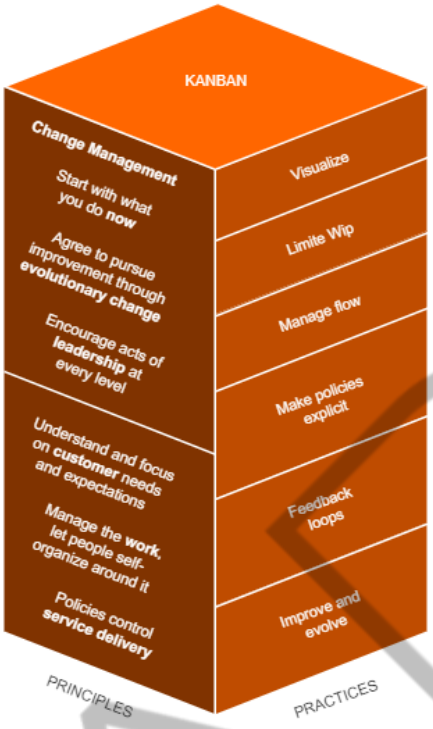

Figura 2.18 – Princípios e práticas Kanban.
Fonte: Google Images (2019)

2.3.3 Elementos do sistema

- **Quadro kanban:** Um dos elementos principais para visualizar de forma simples o fluxo e andamento do trabalho, porém o fato de você ter um kanban não significa que você está usando o método Kanban se você não implementar os princípios e práticas

Figura 2.19 – Exemplo quadro kanban
Fonte: Google Images (2019)

- **Colunas do kanban:** As fases do trabalho no quadro *kanban* são representadas por coluna, e quanto mais para a direta for a coluna, mais valor e tempo foram investidos. Na comunidade ágil discute-se se um item de trabalho pode ou não voltar colunas; vamos supor que uma atividade está em teste, e quando um erro é descoberto são necessários ajustes de desenvolvimento, um lado da comunidade defende que o item pode voltar para desenvolvimento para representar melhor a situação atual do trabalho, outro lado diz que não, que deve-se manter o item na coluna que está, pois mesmo com um pouco de desenvolvimento na fase de testes, a fase principal do desenvolvimento já passou, além de poder trazer complicações de WIP para a coluna anterior se a mesma estiver na capacidade máxima. Vale a pena experimentar e ver qual o melhor modelo para a situação do fluxo do seu trabalho
- **Raias:** Além das colunas verticais que indicam o estágio do processo, também podemos inserir raias horizontais, utilizadas para destacar visualmente um diferente tipo de atividade, como, por exemplo, algo

urgente, normalmente a raia de algo urgente é chamada de *fast-lane* ou *expedite*

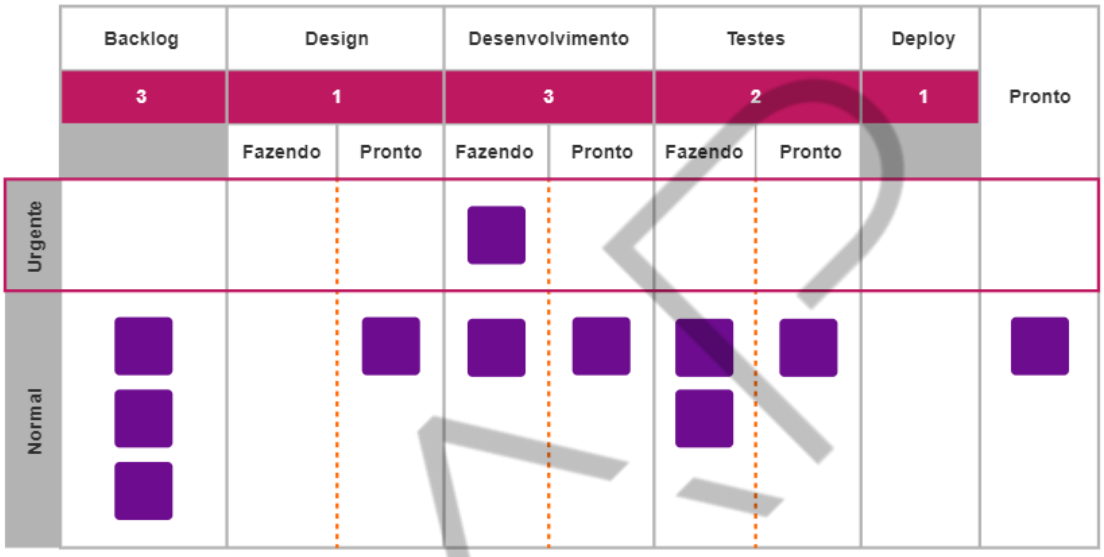

Figura 2.20 – Quadro kanban com raia
Fonte: Google Images (2019)

- **Cartões:** A representação das atividades de trabalho se faz por meio de cartões, que na maioria das vezes são *post-its* em quadros físicos. Nos cartões são anotadas as atividades, pessoas que estão atuando, sinalização de impedimentos, entre outros
- **Limite de WIP:** O limite do trabalho em progresso é um elemento fundamental para organização do fluxo, o mundo ideal seria que o limite do WIP fosse igual em todas as etapas do processo. Imagine o seu fluxo de trabalho como um tubo de água e o limite do WIP a espessura do mesmo; se o tubo for uniforme do começo ao fim, a fluidez da água é constante, porém, se esse tubo tiver espessuras diferentes entre o começo e o fim, a água pode passar de forma mais rápida ou mais devagar dependendo do momento. O mesmo se aplica a trabalhos a serem realizados dentro do quadro, porém mesmo que as limitações de WIP não sejam iguais em todas

as etapas, dependendo da quantidade de profissionais ou tempo necessário para a atividade, o fluxo pode se tornar constante, o ideal é observar o fluxo e ir ajustando o WIP da melhor forma

- **Políticas explícitas:** As regras e fluxos do trabalho devem estar visíveis para evitar dúvidas sobre o processo de trabalho, essas políticas podem estar no quadro *kanban*, em um painel do time, entre outros; o importante é expor e respeitar as mesmas. Alguns exemplos de políticas são: Só podemos trabalhar com 2 itens urgentes por vez, os limites das colunas devem ser respeitados, somente o responsável pelos testes pode mover a atividade para pronto

2.3.4 Vantagens sobre o Scrum

Se abordamos o tema concorrência entre Metodologias Ágeis, há uma discussão muito grande entre Scrum versus Kanban, o que acaba sendo uma comparação injusta, pois o Scrum é um framework muito mais prescritivo (possui regras, normas, diz como fazer) e o Kanban é um método mais flexível. Porém quando aplicamos o Kanban no mundo de desenvolvimento de software, algumas vantagens aparecem referente ao SCRUM:

- **Múltiplos focos:** Se um time necessita dar manutenção em um produto legado e também precisa desenvolver funcionalidades novas no mesmo, no Scrum é exigido um time para o desenvolvimento e outro para a manutenção, visto que o Scrum possui um *time box* e quantidade de trabalho predefinidos em determinado período de tempo, e conciliar as duas atividades impactaria diretamente nos acordos da entrega. Já no Kanban, é possível realizar ambas as atividades com o mesmo time já que o trabalho é mais *just-in-time*, além de não possuir um *time box* definido
- **Entregas constantes:** Tanto no Scrum quanto no Kanban as entregas são constantes, porém o Kanban permite ciclos menores de entregas tornando-as mais constantes. A utilização de *Sprints* e *time box* dificulta um pouco a diminuição do ciclo da entrega

- **Mais evolutivo:** Quando aplicamos o Scrum, algumas mudanças iniciais no processo atual serão necessárias, além de solicitar que o time Scrum seja pequeno, o que pode ocasionar alguma reestruturação dos profissionais, já no Kanban é enfatizado começar com o que você tem, até mesmo equipe maiores, e trabalha as mudanças de forma incremental e evolutiva

2.3.5 Caso de Uso

O método Kanban é recomendado para times que já conhecem a cultura ágil e precisam trabalhar em escopos de trabalho variados (manutenção, novas funcionalidades, operação). Muitos associam o Kanban a uma evolução do Scrum. Apesar de ser mais leve e menos prescritivo, o Kanban tem maiores chances de sucesso quando completado com as práticas do Scrum, como as cerimônias, papéis e responsabilidades, desenvolvimento incremental do produto, trabalho em equipe, priorização de trabalho de acordo com maior valor para o cliente ou produto, entre outros. A junção do framework Scrum e do método Kanban está sendo denominada no mercado como ***Scrumban***, e seja qual for a metodologia de desenvolvimento ágil que você for adotar, escolha aquela que mais se adapte ao seu projeto, produto, pessoas e organização.

Figura 2.21 – Scrumban, o melhor do Scrum e do Kanban
Fonte: Google Images (2019)

2.4 *Extreme Programming*

Extreme Programming ou XP é uma metodologia de desenvolvimento de software leve e não prescritivo, criado em 1997, que procura levar ao extremo as boas práticas de engenharia de software, com foco em agilidade e qualidade, e assim como o Scrum e Kanban, se apoia em valores e práticas.

2.4.1 Valores

- **Comunicação:** O XP é organizado em práticas que não podem ocorrer sem a comunicação, como, por exemplo, a programação em pares, em que os desenvolvedores estão juntos programando em um único computador, trocando informações e se comunicando constantemente
- **Simplicidade:** Muitas vezes ao iniciarmos o desenvolvimento de uma demanda, acabamos por fazer algo mais complexo do que o necessário, o XP preza por fazer o mais simples possível no agora e, caso seja necessário, algo mais complexo no amanhã
- **Feedback:** O XP procura alcançar o feedback da forma mais rápida possível, através de testes automatizados, conseguimos respostas imediatas se aquilo que foi implementado ou alterado está funcionando
- **Coragem:** Para trabalhar com o XP, coragem é necessária para dar e receber *feedback*, fazer o que precisa ser feito, descartar o código ruim e protótipos criados que não devem virar produtos, aprender com os erros e acreditar na capacidade de reagir a mudanças
- **Respeito:** Os membros da equipe precisam respeitar uns aos outros e seus respectivos pontos de vista, saber ouvir e falar contribuirá com a comunicação e com o sucesso do projeto

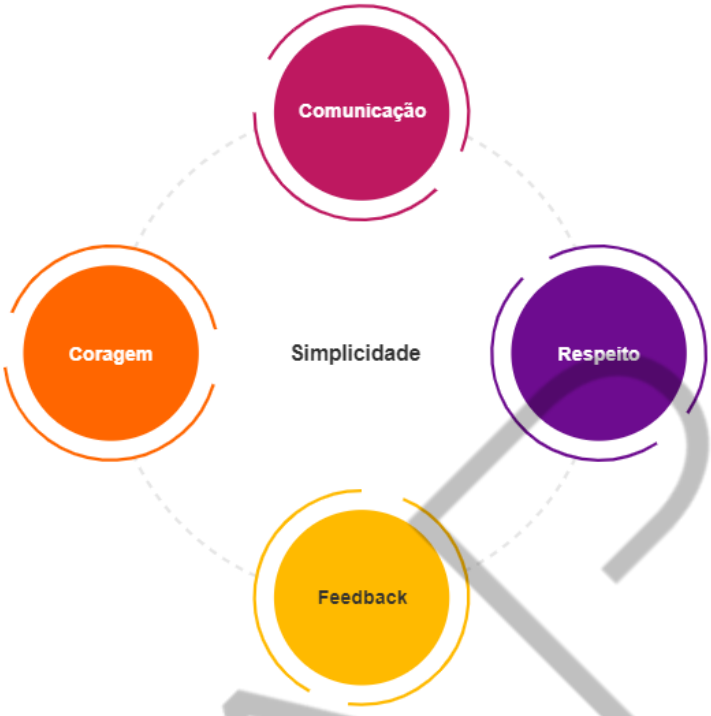

Figura 2.22 – Valores do *Extreme Programming (XP)*
Fonte: Google Images (2019)

2.4.2 Práticas

Para aplicar o XP, existe um conjunto de boas práticas que o método sugere:

- **Propriedade Coletiva (*Collective Ownership*):** O código-fonte não tem dono, todos da equipe têm permissão para modifica-lo, contribuindo para disseminar o conhecimento de diversas partes do sistema
- **Padronização de Código (*Coding Standards*):** O time de desenvolvimento define padrões e regras de programação que todos devem seguir, para que assim o código mantenha uma padronização independentemente da quantidade de pessoas que adicionou ou alterou códigos

- **Versões Pequenas (*Small Releases*):** O projeto é liberado em versões funcionais pequenas, auxiliando no processo de aceitação do cliente

- **Planejando o jogo (*Planning Game*):** O desenvolvimento das iterações entre cliente e desenvolvedores é realizado para priorizar as funcionalidades através de uma reunião, chamada de Jogo do Planejamento, e segue a mesma ideia do Planning do Scrum, porém acontece semanalmente
- **Testes de aceitação (*Customer Tests*):** O cliente em conjunto com o time cria testes de aceitação referente a um requisito do sistema
- **Programação Pareada (*Pair Programming*):** A programação é realizada em dupla e em um único computador, em que a dupla deve ser composta se possível por um profissional iniciante e outro mais experiente. Assim o código é sempre revisto por duas pessoas, além de disseminar o conhecimento entre o mais experiente e o iniciante

Figura 2.23 – Programação pareada
Fonte: Google Images (2019)

- **Desenvolvimento orientado a testes (*Test Driven Development*):** É uma abordagem complexa devido ao fato de ser contrária ao padrão comum seguido nos times de desenvolvimento. Se costumeiramente se cria o código da funcionalidade para depois desenvolver os testes unitários, em

TDD (e portanto no XP) é o contrário, primeiro cria-se os testes unitários para depois criar o código da funcionalidade que fará esses testes passarem, obrigando assim que a qualidade do código seja mantida

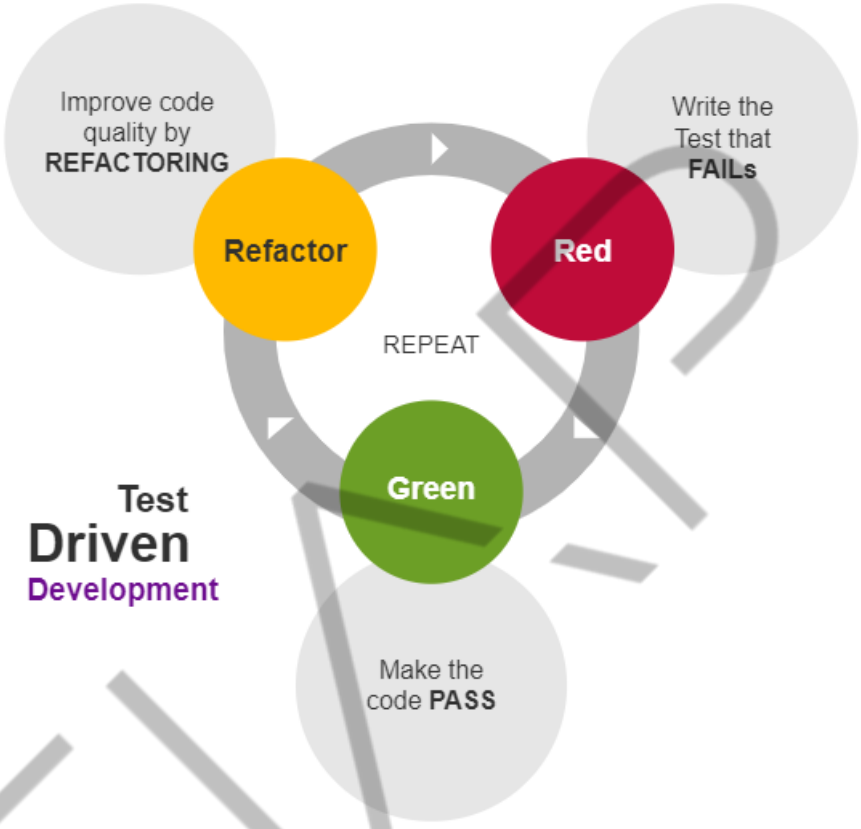

Figura 2.24 – *Test Driven Development*
Fonte: Google Images (2019)

- **Integração contínua (*Continuous Integration*):** Sempre integrar a funcionalidade que está sendo desenvolvida com a funcionalidade atual, para evitar o descobrimento de problemas apenas no final do ciclo
- **Design Simples (*Simple Design*):** Manter o código simples é um dos princípios do XP, o que não quer dizer código fácil, a simplicidade do código contribui para o entendimento do mesmo e com o foco do que foi solicitado na funcionalidade
- **Refatoração (*Refactoring*):** Processo de melhoria contínua de programação, com o objetivo de melhorar a clareza do código e permitir maior reaproveitamento do mesmo

- **Metáforas (*Metaphor*):** Facilitar a comunicação entre o cliente e os membros do time através de metáforas, nem sempre o cliente entenderá o mundo sistêmico então devemos traduzir as necessidades que ele espera para dento do projeto
- **Semana de 40 horas (*Sustainable Pace*):** O XP acredita que o trabalho com qualidade deve ser feito dentro de 40 horas semanais, 8 horas por dia e sempre evitando horas extras para que não ocorra sobrecarga dos profissionais. O ambiente de trabalho deve possuir condições favoráveis e o clima do time deve ser motivado

2.4.3 Caso de Uso

Recomendado para times que necessitam disseminar o conhecimento, o *Extreme Programming* foca muito mais em processos e práticas em volta do software do que nos processos de trabalho e pessoas, e para compensar isso, é possível aplicar o XP com outras metodologias de desenvolvimento que o completam.

O *Extreme Programming* propõe uma série de práticas que visam o melhor para o desenvolvimento de software, porém é comum não conseguir aplicar todas elas, o ideal é escolher as práticas mais viáveis dentro do possível do escopo de trabalho e aplicá-las, principalmente as práticas relacionadas aos testes de aceitação e integração para melhorar ainda mais a qualidade do produto.

2.5 Atuação Profissional

A utilização em grande escala das variadas metodologias de desenvolvimento ágeis no mercado de trabalho atual introduziu o conceito de times multidisciplinares e com isso criou a necessidade de novos papéis e responsabilidade a serem desempenhados na organização, como, por exemplo, o *Scrum Master* e *Product Owner* que o framework Scrum implementa. Vamos abordar agora cada um desses papéis e suas respectivas funções de forma resumida.

Figura 2.25 – Atuação profissional no mercado de trabalho
Fonte: Google Images (2019)

O time de desenvolvimento normalmente é composto por:

- **Desenvolvedor *back-end*:** Profissional que trabalha na parte de "trás" da aplicação, na maioria das vezes responsável pela codificação de grande pedaço das regras de negócio. Geralmente esse profissional atua pouco com a parte visual, e domina linguagens de programação (ex: java, c#, go, php, python, ruby, etc.) que dão suporte ao seu trabalho
- **Desenvolvedor *front-end*:** Ao contrário do *back-end*, o desenvolvedor *front-end* trabalha na parte da "frente" da aplicação, desenvolvendo as telas

que vão interagir com os usuários se importando com a experiência do mesmo. Esse profissional domina geralmente linguagens como JavaScript, Node, CSS e HTML, entre outras, para trabalhar

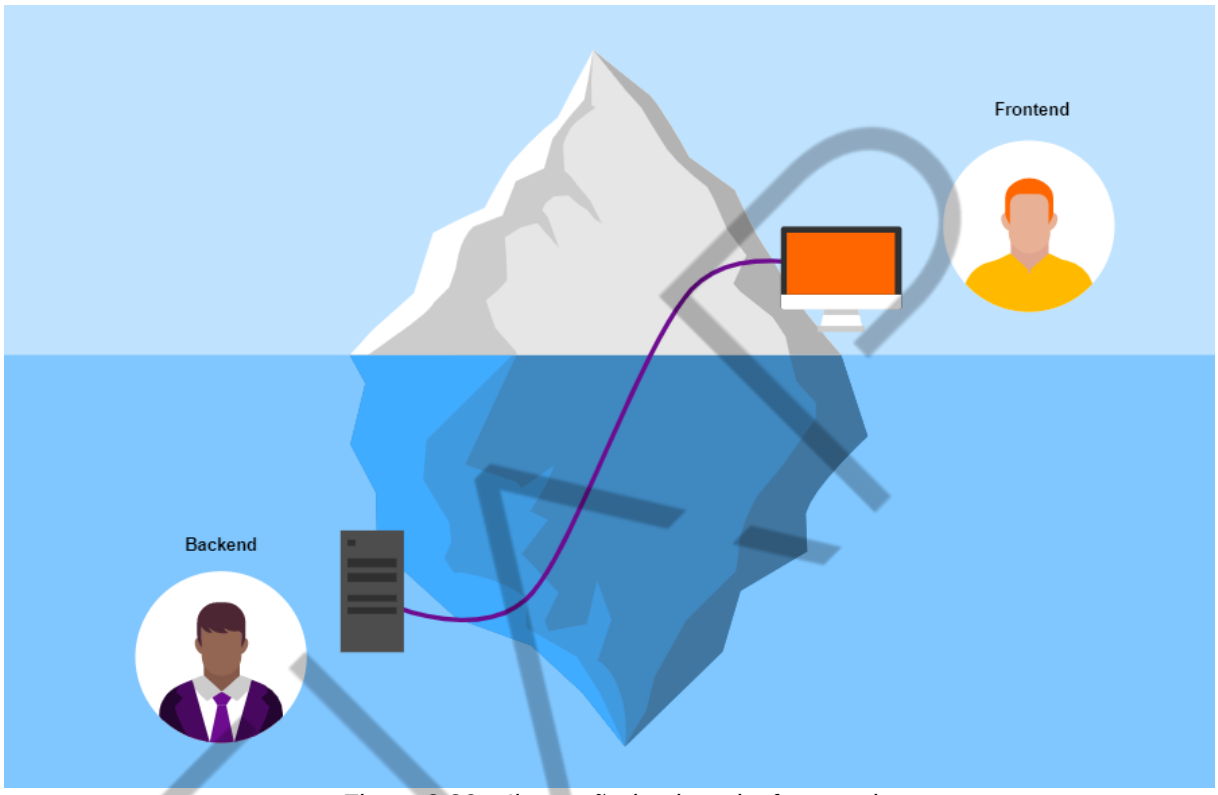

Figura 2.26 – Ilustração *back-end* e *front-end*
Fonte: Google Images (2019)

- **Desenvolvedor *full stack*:** Um profissional muito difícil de se encontrar no mercado, o desenvolvedor *full stack* é aquele que desempenha tanto a função de *back-end* como *front-end*. A dificuldade em se tornar *full stack* deve-se à grande quantidade de tecnologias que o desenvolvedor deve dominar tanto no âmbito *back* quanto *front*

Figura 2.27 – Ilustração desenvolvedor full stack
Fonte: Google Images (2019)

- **Analista de Qualidade (Quality Assurance - QA):** Profissional responsável pela realização dos testes do software, é importante enfatizar que nas metodologias ágeis a qualidade é responsabilidade de todos, porém o QA faz o papel de guardião. O QA normalmente domina técnicas de teste unitário, aceitação, integração, carga, entre outros. Muitos QA também sabem desenvolver e acabam contribuindo com o desenvolvimento do software quando possível

- ***Scrum Master / Agile Master*:** O Scrum Master é responsável pela cultura ágil dentro do time e por disseminar todas as práticas do framework Scrum. Ele atua como um líder do processo Scrum e um facilitador, contribuindo na resolução de conflitos e remoção de impedimentos. O Agile Master exerce as mesmas atividades do Scrum Master, com a diferença que também domina demais metodologias de desenvolvimento ágil como Kanban ou XP

Junto ao time de desenvolvimento, também temos:

- ***Product Owner* (Dono do Produto):** Responsável por decidir quais os recursos e funcionalidades o produto deve ter, e qual a ordem de prioridade

que devem ser desenvolvidos. Também mantém e comunica uma visão clara do que o Time Scrum está trabalhando no produto, sendo um ponto de intersecção entre a área de negócio e a área de desenvolvimento

Os profissionais a seguir normalmente são compartilhados entre times, mas também podem fazer parte de um único time de desenvolvimento:

- ***Designer de User Interface* ou *User Experience*:** O designer pode atuar tanto com foco na experiência do usuário, aplicando a melhor usabilidade e interação que o software terá, quanto na interface do software, construindo o design que represente a identidade visual da empresa e melhores padrões de design que o desenvolver front-end irá codificar
- **Analista de Segurança:** Com o objetivo de olhar para a segurança na organização, o Analista de Segurança é fundamental para promover a Cultura do DevSecOps e trazer o quanto antes a preocupação com segurança para o início do processo de trabalho, identificando, protegendo, detectando, respondendo e recuperando os pontos necessários sobre segurança

Figura 2.28 – Ilustração analista de segurança
Fonte: Google Images (2019)

- **SysAdmin (Administrador de Sistemas):** A função de SysAdmin é algo bem abrangente, mas geralmente são responsáveis por instalar, suportar e manter os servidores e sistema da organização junto ao time de desenvolvimento. Também é um ponto-chave para a disseminação da cultura DevOps dentro da empresa

- **Analista de Dados (AD)/ Administrador de Banco de Dados (DBA):** Quando o assunto é banco de dados, existem dois perfis, o de DBA responsável por gerenciar, instalar, configurar, atualizar e monitorar um banco de dados, e o AD responsável por coletar, compilar, analisar e interpretar os dados do banco. O DBA acaba atuando mais no nível de hardware e software, e o AD no nível de dados e negócio
- **Especialista:** O papel de especialista tem como objetivo ser o ponto de referência por atuação contribuindo com a formação dos profissionais e resolução de demandas complexas ou que envolvam diversas áreas da organização. Para exemplificar, é possível ter um Especialista *Back-End* e outro Especialista *Front-End*, e os mesmos darem o suporte para os demais desenvolvedores *back-end* e *front-end*, isso também serve para as outras funções, Especialista de Segurança, Qualidade, entre outras

Na parte de Gestão da empresa, geralmente temos:

- **Coordenador e Gerente:** Responsáveis por coordenar e gerenciar as atividades das equipes de TI, avaliar e identificar soluções tecnológicas para otimizar os processos, planejar os projetos de implantação de sistemas e acompanhar as necessidades do negócio e dos clientes. Com relação as pessoas, realizar o acompanhamento e evolução dos profissionais, manter a motivação, realizar feedbacks constantes e mentorias. Os papéis de Coordenador e Gerente possuem atividades similares, a principal ideia é que o Coordenador consiga escalar o trabalho do Gerente, se um Gerente possuir 6 times, ele poderia ter 2 Coordenadores responsáveis por 3 times cada um, e os Coordenadores realizarem o *report* desses times para o Gerente. Dependendo da organização, o Gerente possui algumas funções

a mais que o Coordenador pode não ter acesso, como planejamento financeiro, aprovação de contratos, *report* para o *board* executivo, entre outros

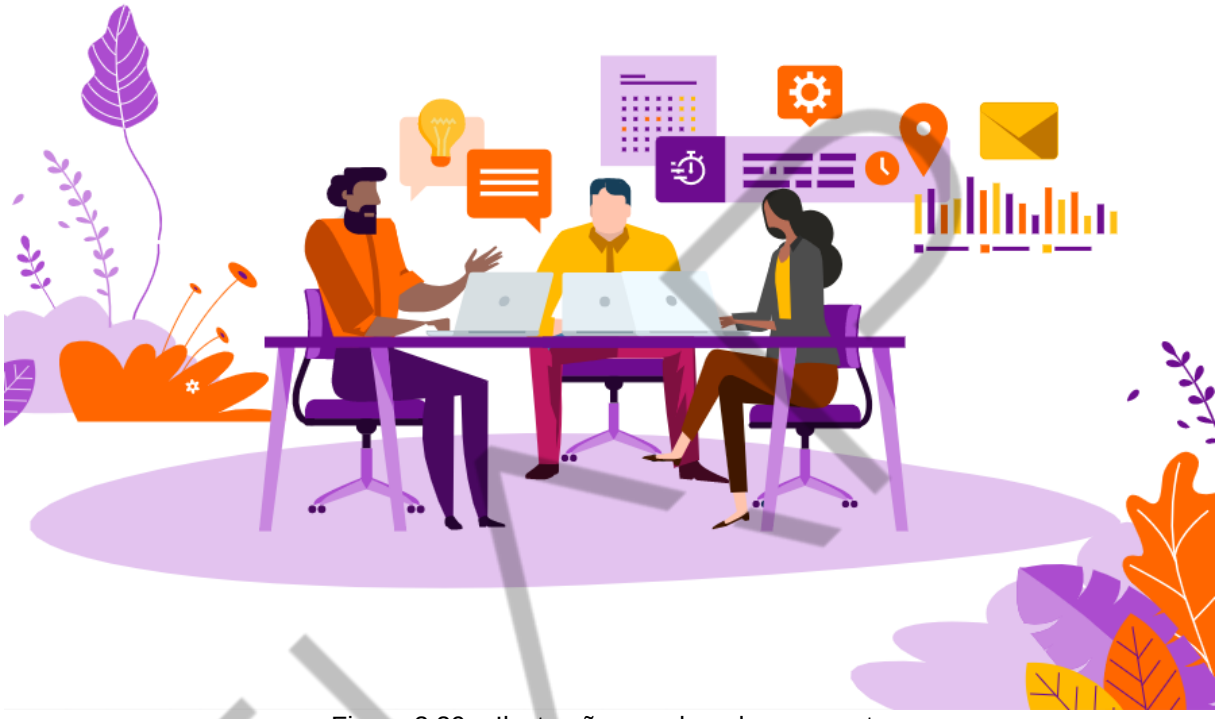

Figura 2.29 – Ilustração coordenador e gerente
Fonte: Google Images (2019)

REFERÊNCIAS

ANDERSON, David J. **The Principles & General Practices The Kanban Method**. 2010. Disponível em: <http://www.djaa.com/principles-general-practices-kanban-method>. Acesso em: 18 mar. 2019.

TOYOTA. **Systema Toyota de Produção**. Disponível em: <https://www.toyota.com.br/mundo-toyota/toyota-production-system>. Acesso em: 11 mar. 2019.

SCRUM ORG. **What is Scrum?**. Disponível em: <https://www.scrum.org/resources/what-is-scrum>. Acesso em: 15 mar. 2019.

WELLS, Dom. **Extreme Programming: A gentle introduction**. 2013. Disponível em: <http://www.extremeprogramming.org/>. Acesso em: 21 mar. 2019.