ENGENHARIA DE REQUISITOS

AULA 6

Prof.ª Rosemari Pavan Rattmann

CONVERSA INICIAL

A Engenharia de Requisitos nos ensina que o sucesso de um projeto de *software* depende de vários fatores, como requisitos bem entendidos e de acordo com as necessidades do cliente, do tempo de entrega e dos custos envolvidos. Estudaremos, nesta abordagem, novas maneiras de pensar em um processo de desenvolvimento de *software* que contribua para que o cliente seja mais competitivo, o que significa, hoje, entregas rápidas e com qualidade.

Veremos que, independentemente da metodologia utilizada, sempre haverá vantagens e desvantagens, a preocupação com a qualidade do produto a ser entregue, e a satisfação do cliente.

Nesse contexto de agilidade no desenvolvimento de produtos, estudaremos como o manifesto ágil impactou os processos de produção de *software*, e como ele propõe que as ações das equipes sejam ágeis e, ao mesmo tempo, focadas no que agrega valor ao projeto e ao cliente.

Estudaremos algumas abordagens ágeis, e como a elicitação de requisitos tradicional pode ser pensada na forma de requisitos ágeis que possam se ajustar e nortear esses novos métodos de desenvolvimento do produto. Veremos, especialmente, como trabalhar com a metodologia *Scrum*, focando na fase de elicitação de requisitos ágeis.

Após analisarmos algumas técnicas ágeis para a coleta de requisitos, discutiremos a importância, nas metodologias ágeis, das revisões constantes das listas de tarefas (*backlog* do produto) e, consequentemente, dos requisitos dinâmicos que evoluem ao longo do desenvolvimento do projeto.

Para acompanhar e gerenciar o trabalho das equipes ágeis, mostraremos exemplos de ferramentas que colaboram para a otimização do trabalho e controle de tarefas.

Trataremos do produto, o qual será entregue por versões incrementais a cada ciclo, e que a priorização das tarefas deverá ser constantemente avaliada, pois a participação do cliente em todo o processo passa a ser de grande relevância.

Completando o estudo da abordagem ágil quanto à elicitação de requisitos, estudaremos as histórias de usuários, e como é necessário o incentivo da colaboração e comunicação entre os envolvidos na equipe.

TEMA 1 – REQUISITOS ÁGEIS

1.1 Processos de desenvolvimento

O *software*, hoje, faz parte de quase todas as negociações de empresas. A rapidez na produção de um novo sistema que utilize *software* é algo sempre desejado, e muito mais, em um ambiente competitivo para muitas empresas. O novo sistema precisa ser útil para ampliar as negociações e possibilitar oportunidades; desse modo, o desenvolvimento rápido, com qualidade, passa a ser requisito.

Por outro lado, quando falamos em qualidade de especificação de requisitos, precisamos pensar que não é possível definir requisitos de *software* estáveis, pois o ambiente das empresas é dinâmico e competitivo, e tais requisitos mudam porque os clientes não conseguem prever certas funcionalidades, interações com outros sistemas e operações. Torna-se muito mais complexo lidar com o desenvolvimento de um *software* sem todos os requisitos disponíveis, mantendo qualidade e pouco retrabalho.

Assim, os processos de desenvolvimento que efetuam a especificação de requisitos, e depois o projeto, a construção e os testes, não são considerados processos rápidos. À medida que os requisitos mudam, ou novos são descobertos, todo o processo precisa ser revisto. Consequentemente, um processo convencional, em cascata, ou baseado em especificação, demora, e o *software* final é entregue após o prazo inicial contratado (considerando, também, os replanejamentos de datas por mudanças de requisitos). Todo processo de desenvolvimento de *software* possui um conjunto de atividades que precisam ser realizadas para o desenvolvimento do produto, como:

- **Especificação**: funcionalidades e regras de negócio, ou seja, o que deve ser feito.
- **Projeto** e **implementação**: produção do *software* de acordo com as especificações, utilizando modelos de projeto, protótipos e, após aprovação do cliente, é construído.
- **Validação**: todos os testes necessários para garantir que o produto final funcione e atende as especificações.
- **Evolução**: especificação de novas versões ao longo do tempo.

Embora os métodos ágeis minimizem a documentação e o tempo de especificação de requisitos, ainda assim, precisam incluir essas atividades durante o processo de desenvolvimento, mesmo que simplifiquem.

Há tipos de sistemas para os quais é indicado o processo de desenvolvimento com análise mais completa; no entanto, para outros casos, em que os clientes são mais dinâmicos, há a necessidade de desenvolvimento mais rápido e de processos que possam lidar com requisitos que mudam mais frequentemente (Sommerville, 2018). Esse tipo de desenvolvimento rápido de *software* utiliza a abordagem ágil.

Esses métodos ágeis são criados para produzir *software* útil de maneira rápida, intercalando processos de especificação, projeto e implementação. O documento de especificação de requisitos é resumido com as principais características do sistema. A cada período curto de tempo, versões do sistema são entregues aos *stakeholders*, novos requisitos são propostos para as próximas versões, e essas mudanças são esperadas e implementadas.

Para apoiar esses processos, são utilizados muitas ferramentas, testes automatizados, apoio à integração de partes do sistema e interfaces de usuário. Os métodos ágeis produzem desenvolvimento incremental, partes menores, várias versões, disponibilização rápida para obter *feedback* rápido dos requisitos que mudam. A documentação é reduzida, mas existe, e pode ser menos formal. A Figura 1 exibe as principais diferenças entre as metodologias tradicional e ágil para o desenvolvimento de *software*.

Figura 1 – Comparativo entre os processos de desenvolvimento de *software* tradicional e ágil

Fonte: Rattmann/Canvas.

Existem desvantagens na utilização dos métodos ágeis, como em toda metodologia. Assim como os métodos tradicionais de desenvolvimento de *software* são mais direcionados para empresas maiores, com *softwares* mais complexos ou com requisitos mais estáveis, os métodos ágeis são voltados a projetos pequenos, ou a processos inovadores em empresas com outras necessidades. Listamos algumas desvantagens:

- Nem sempre o cliente tem o tempo necessário para acompanhar o desenvolvimento do projeto.
- Nem sempre a equipe disponível tem a formação adequada e o ritmo necessário.
- Podem existir conflitos entre os donos do produto.
- Devido aos prazos, as soluções podem ser comprometidas pela pressa no desenvolvimento.

Também, segundo Foggetti (2014), há ainda desafios a serem superados pelas metodologias ágeis:

- Realizar com mais qualidade a análise de riscos, sem deixar de ser prático.
- Poder ser útil para grandes empresas e grandes equipes.
- Aumentar o foco nas pessoas e os processos mais versáteis.
- Melhorar a qualidade da documentação, sem perder muito tempo, focando na implementação, mas sem perder conhecimento.
- Enfrentar o medo de mudanças.
- Garantir o fluxo contínuo de entregas.

Podemos concluir, portanto, que, independentemente da metodologia utilizada, sempre haverá pontos comuns que precisam melhorar, e a preocupação com a qualidade do produto a ser entregue sempre será a meta principal.

1.2 Manifesto ágil

A filosofia por trás dos métodos ágeis está refletida no *Manifesto ágil* (Manifesto..., 2022), que diz:

> Estamos descobrindo maneiras melhores de desenvolver *software*, fazendo-o nós mesmos e ajudando outros a fazerem o mesmo. Através deste trabalho, passamos a valorizar:
> Indivíduos e interações mais que processos e ferramentas
> *Software* em funcionamento mais que documentação abrangente
> Colaboração com o cliente mais que negociação de contratos
> Responder a mudanças mais que seguir um plano
> Ou seja, mesmo havendo valor nos itens à direita, valorizamos mais os itens à esquerda.

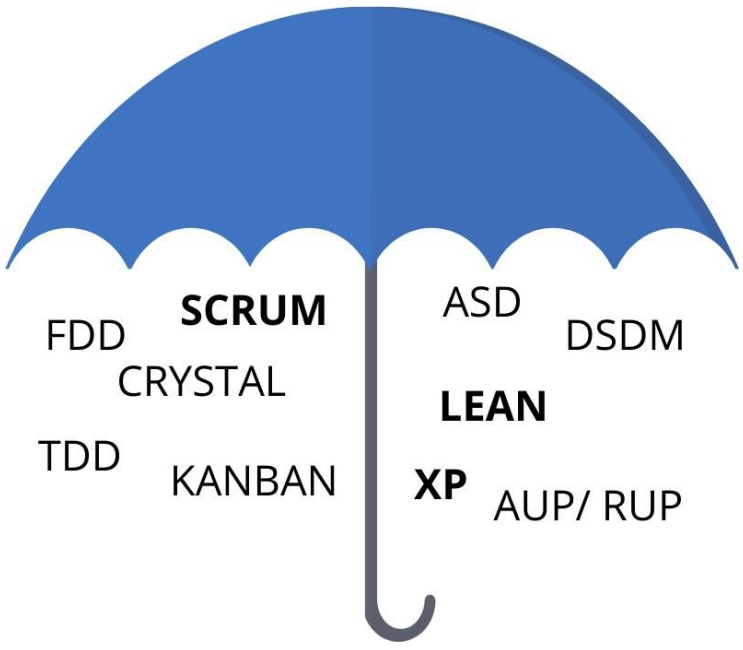
Os 12 princípios do *Manifesto ágil* (2022) são:

1. Nossa maior prioridade é satisfazer o cliente por meio da entrega contínua adiantada de *software* com valor agregado.
2. Mudanças nos requisitos são bem-vindas, mesmo tardiamente no desenvolvimento; processos ágeis tiram vantagem das mudanças, visando à vantagem competitiva para o cliente.
3. Entregar frequentemente *software* funcionando, de poucas semanas a poucos meses, com preferência pela menor escala de tempo.
4. Pessoas de negócio e desenvolvedores devem trabalhar diariamente em conjunto por todo o projeto.
5. Construa projetos em torno de indivíduos motivados. Dê a eles o ambiente e o suporte necessários, e confie neles para fazer o trabalho.

6. O método mais eficiente e eficaz de transmitir informações para e entre uma equipe de desenvolvimento é pela conversa face a face.
7. *Software* funcionando é a medida primária de progresso.
8. Os processos ágeis promovem desenvolvimento sustentável. Os patrocinadores, desenvolvedores e usuários devem ser capazes de manter um ritmo constante indefinidamente.
9. Contínua atenção à excelência técnica e bom design aumentam a agilidade.
10. Simplicidade: “a arte de maximizar a quantidade de trabalho não realizado” é essencial.
11. As melhores arquiteturas, requisitos e designs emergem de equipes auto-organizáveis.
12. Em intervalos regulares, a equipe reflete sobre como se tornar mais eficaz e então refina e ajusta seu comportamento de acordo com o necessário.

Com base nesses princípios, os métodos ágeis existentes oferecem um conjunto de atividades que podem ser adotadas durante o processo de desenvolvimento de *software*. A Figura 2 representa o guarda-chuvas ágil e a existência de diversas ferramentas e *frameworks* para utilização no processo de desenvolvimento de *software*.

Figura 2 – Ferramentas e *frameworks* sob o guarda-chuva ágil



Fonte: Rattmann/Canvas.

1.3 Fundamentos da abordagem ágil

O desenvolvimento incremental é baseado na interação com os usuários do *software*, implementando uma versão inicial, entregando-a, para obter *feedback*, evoluindo para uma nova versão incorporando outros requisitos ou alterações, e entregando a nova versão até alcançar o sistema esperado. As várias etapas de especificação, projeto e validação são intercaladas com avaliações rápidas de seu resultado.

Segundo Sommerville (2018), esse tipo de desenvolvimento é feito para solucionar os problemas – não temos ainda uma solução completa com antecedência, mas evoluímos para uma solução em uma série de passos, retrocedendo quando houver um erro e, continuamente, incorporando novas funcionalidades.

A metodologia ágil sempre apresenta as seguintes fases:

- Visão: identificação do escopo do projeto;
- Especulação: definição das datas de entrega;
- Exploração: desenvolvimento do projeto;
- Adaptação: revisão e avaliação dos erros, para não os repetir no próximo ciclo.

Para o trabalho em equipe, nas abordagens ágeis, há papéis bem definidos, como o proprietário do produto (PO, ou *product owner*) que sempre acompanha o desenvolvimento para esclarecer o que deve ser feito – é ele quem estipula as datas de entregas; o cliente, que participa muito do desenvolvimento e paga pelo serviço; e, a equipe de desenvolvimento.

Na engenharia de requisitos, o processo de especificação de requisitos deve conter as necessidades dos *stakeholders* baseadas em acordos e contratos. Além da elicitação e análise dos requisitos identificados, ocorrem validações, até a produção do documento de requisitos.

Nos métodos ágeis, a especificação de requisitos não é uma atividade separada: ela faz parte do desenvolvimento do sistema. Para cada incremento (nova versão), os requisitos são descobertos, informalmente, um pouco antes da implementação. Isso significa que eles dependem das prioridades dos *stakeholders* para serem especificados (Massari, 2014).

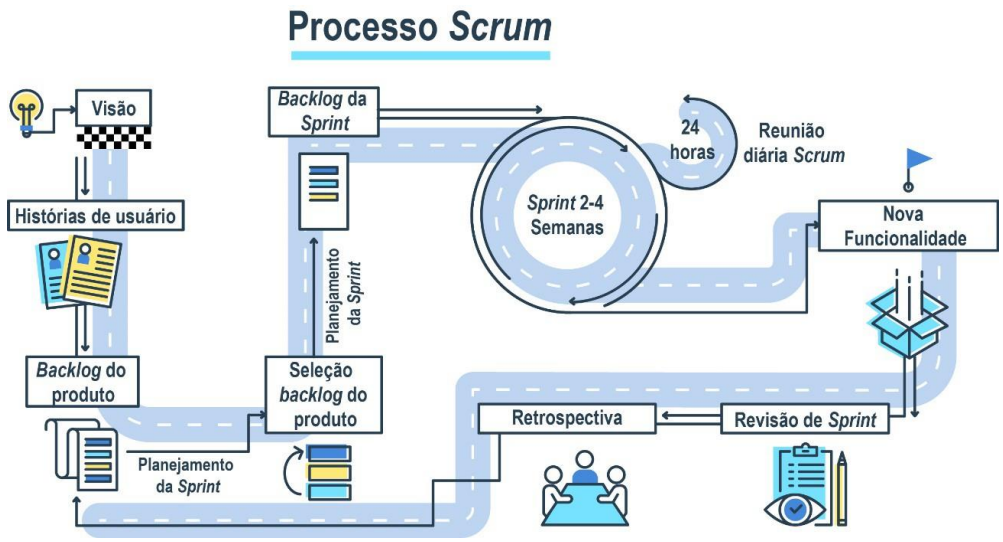
Os métodos ágeis mais conhecidos (Ferreira, 2020) são a *extreme programming* ou XP (“programação extrema”), o *Scrum*, *crystal*,

desenvolvimento adaptativo, *dynamic systems development method* (DSDM, ou "método de desenvolvimento de sistemas dinâmicos"), *Feature-Driven Development* ("entregas por funcionalidade"). Cada um deles tem suas particularidades e técnicas. O mais conhecido e utilizado, por ser bastante versátil, é o *Scrum*.

O ***Scrum*** é uma metodologia ágil muito utilizada quando os requisitos mudam rapidamente; ele não estabelece uma técnica específica para o desenvolvimento, apenas regras e práticas de gestão. As práticas são muito conhecidas (observe também a Figura 3):

- ***Backlog* do produto**: contém a lista de tudo o que deve ser realizado; são as necessidades atendidas pelo que será criado;
- **Reuniões diárias** (*daily scrum*): são reuniões que acontecem todos os dias, com toda a equipe em pé, para ser rápida e eficiente e na qual, três perguntas são respondidas:
  - ✓ O que foi feito ontem?
  - ✓ O que será feito hoje?
  - ✓ Há algum obstáculo para realizar as atividades?
- ***Sprint*** é o tempo que a equipe tem para executar o que foi definido no *backlog* do produto. Geralmente, dura até quatro semanas.

Figura 3 – *Scrum*: abordagem ágil para desenvolvimento de *software*



Crédito: Elaine333/Shutterstock.

Ao longo do nosso estudo, e depois de termos conversado bastante sobreos métodos ágeis, como especificar requisitos com essa abordagem? Alguns aspectos continuam sendo os mesmos, como ouvir as necessidades do cliente e

de todos os envolvidos. No entanto, a partir desse momento, precisamos lembrar de que o cliente precisa estar presente durante todo o desenvolvimento do projeto, para avaliar as partes entregues e opinar sobre possíveis mudanças e redirecionamentos do que está sendo implementado, pois, mesmo que a equipe de desenvolvedores saiba o *backlog*, somente o cliente tem real conhecimento do que precisará que o *software* faça em seu dia a dia.

Os requisitos não são definidos antes, em um único momento; eles podem ser repensados, e outros, novos, podem surgir ao longo do projeto. Por isso, o *feedback* do cliente indica novas possibilidades e respostas que precisarão ser incorporadas às próximas entregas.

TEMA 2 – CARACTERÍSTICAS E REFINAMENTO DE REQUISITOS

Como já estudamos, o escopo de um projeto indica as necessidades que precisarão ser satisfeitas e que derivam de uma coleta de requisitos. Na abordagem ágil, há algumas ferramentas e técnicas ágeis para essa coleta de forma colaborativa entre cliente, partes interessadas e equipe de desenvolvimento do projeto. As principais ferramentas são:

- ***Wireframes***: rascunhos do protótipo do *software* ou de partes dele, como interfaces gráficas a fim de apresentar o entendimento do que deve ser feito (Figura 4):

Figura 4 – Exemplo de *wireframe*: protótipo

Fonte: Rattmann/Canvas.

- **Personas**: personagens criadas para representar usuários e identificar características importantes e necessidades específicas. Tentar criar personagens inesperados pode ajudar a avaliar novas funcionalidades que faltaram. A Figura 5 mostra um exemplo de persona:

Figura 5 – Exemplo de descrição de personas

Créditos: zigzag design/Shutterstock.

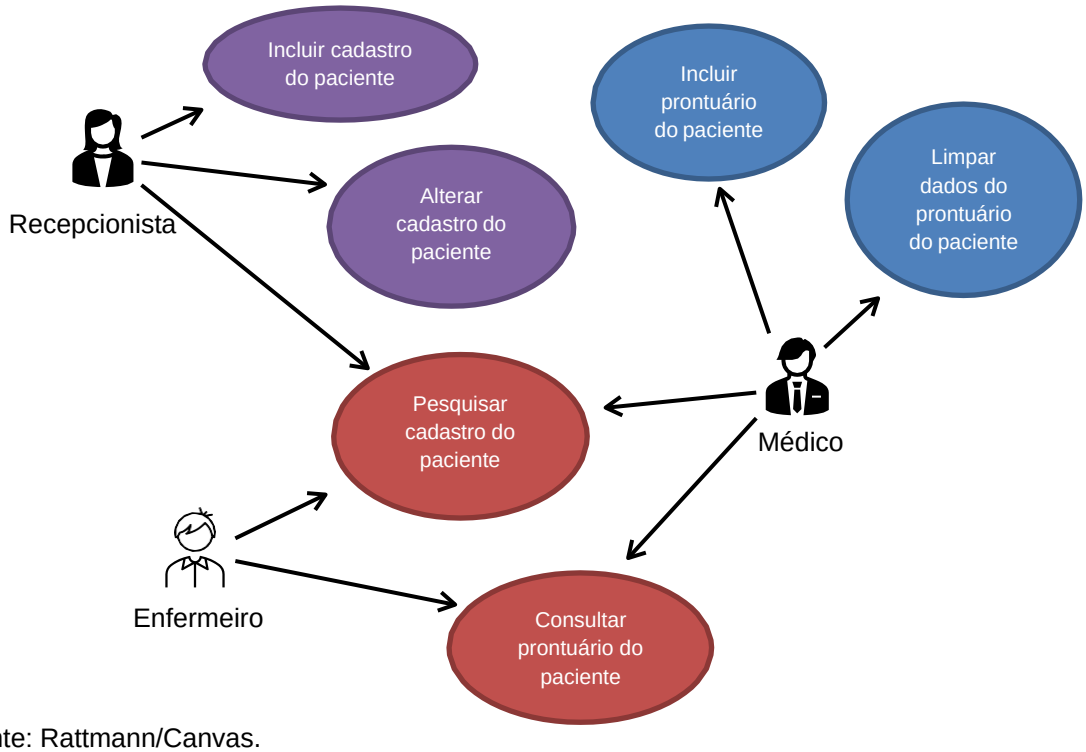
- **Jogos ágeis**: são artefatos lúdicos que ajudam na colaboração entre pessoas para a coleta de requisitos e planejamento do projeto. Uma das técnicas mais usadas é o *scrum poker*, na qual cada participante indica quanto aquela determinada história do usuário representa para si. A equipe pode estimar um conjunto de tarefas rapidamente. Na Figura 6 é representado um encontro para jogar com cartões que possuem requisitos, desejos e necessidades. Essa técnica ajuda a equipe a interagir e a entender a integração das funcionalidades que serão criadas no *software*.

Figura 6 – Planejamento do projeto por meio de jogos

Crédito: diPersika/Shutterstock.

- **Modelagem ágil**: similar aos *wireframes*, utiliza-se da representação gráfica para alinhar o entendimento e o que deve ser feito, como diagramas de caso de uso, modelo de dados e protótipos. É também útil para identificar riscos e problemas, e suas possíveis soluções. Veja o exemplo mostrado na Figura 7:

Figura 7 – Representação gráfica com diagrama de caso de uso



Fonte: Rattmann/Canvas.

- **Histórias de usuário**: segundo Foggetti (2014), essa técnica é muito eficaz para engajar as partes interessadas e para a coleta de requisitos, pois utiliza oficinas para escrita das histórias de usuário. As partes interessadas são reunidas e escrevem seus requisitos em formato pré-definido, representando o ponto de vista de cada um sobre o que o *software* precisa fazer.
- **Agrupamento de histórias de usuário (épicos)**: histórias de usuário muito pequenas são ruins e, muito grandes, dificultam as estimativas; por isso, quando uma história é grande, ou há várias que tratam de um mesmo tema, estas podem ser agrupadas, e recebem o nome de *épicos*. Em breve, estudaremos mais detalhadamente essa técnica de coleta de requisitos ágil. Todos os épicos representam o produto completo, equivalendo a um módulo.

Quando há um conjunto de histórias do usuário e identificamos funções que o produto deve ter, como vários requisitos funcionais e não funcionais, temos uma *feature*, que traduzimos para o termo “funcionalidade”. Se pensarmos na implementação de cada história de usuário, ao final, quando todas estiverem concluídas, a funcionalidade respectiva também estará.

A funcionalidade (*feature*) representa um conjunto de histórias do usuário; trata-se da função do produto, contendo diversos requisitos funcionais e não funcionais e, somente quando todas as histórias estiverem prontas, a funcionalidade estará concluída.

Essa é uma das boas práticas para monitoramento e refinamento do *backlog* do produto (lista de tarefas a serem realizadas). Algumas perguntas-chave podem ser geradas para cada requisito analisado, conforme Sommerville (2018):

- O requisito ainda possui o valor do negócio identificado?
- O requisito ainda é relevante para o produto?
- O requisito ainda é prioriário perante os requisitos seguintes?
- O requisito é épico e precisa ser decomposto em requisitos com maiores detalhes?
- São necessários novos requisitos para que o produto atenda ao objetivo de negócio, conforme a contratação?

As respostas para cada pergunta devem ser analisadas, e podem

acarretar novas prioridades, exclusões e inclusões de requisitos e mudanças no escopo do projeto.

Há uma necessidade contínua de revisar o escopo, pois, em uma abordagem ágil, sempre trabalhamos de forma incremental e iterativa, ou seja, como novos requisitos podem ser criados e alterados, o escopo pode estar sendo alterado de maneira indesejada, ou não contratada.

Independentemente da metodologia de priorização escolhida, o bom senso e as boas práticas devem indicar quais requisitos agregam maior valor ao cliente, e estes devem ter prioridade maior.

O que podemos concluir é que a adoção de metodologia de gerenciamento do projeto híbrida (com os conceitos tradicionais e o ágil) pode facilitar a priorização de requisitos (Ferreira, 2020). Ou seja, o gerente de projetos (metodologia tradicional) estabelece uma sequência para as atividades, facilitando, para o PO, o planejamento das *Sprints* (tarefas e sua sequência).

TEMA 3 – CANVAS E *STORYBOARD*

Grandes empresas têm utilizado uma ferramenta chamada *Project Model Canvas* (PMC) (Vidal, 2017) para planejar seus projetos de forma ágil e com máximo envolvimento dos *stakeholders*. Trata-se de um modelo de planejamento ágil de projetos que foi idealizada pelo brasileiro José Finocchio Junior, cujo objetivo é representar de forma visual e em apenas uma folha chamada *Canvas* (“tela”, em inglês), treze componentes fundamentais para a composição do plano do projeto.

Os pilares da metodologia PMC são:

- **Conceber** (0) – Respostas para as perguntas: Por quê? O quê? Quem? Como? Quando? Quanto?
- **Integrar** (1) – Garantir a consistência entre os blocos e a integração entre os componentes.
- **Resolver** (2) – Identificar os pontos em que o Canvas “travou”, indefinições, falta de informações etc.
- **Compartilhar** (3) – Servirá como base para outros documentos.

Entre os objetivos estão o engajamento de todos os envolvidos – tanto partes interessadas, ou seja, o cliente, quanto a equipe de desenvolvimento; a colaboração entre eles avaliando riscos, planejando as tarefas, entregas, atingimento do escopo, falhas de entendimento da demanda, entre outros.

A Figura 8 representa o processo de quatro etapas para construir o Canvas.

Figura 8 – Modelo Canvas para planejamento de projetos

Fonte: Rattmann/Canvas.

Os benefícios de utilizar essa metodologia podem ser verificados a curto e longo prazos, avaliando o processo de gerenciamento de projeto e a satisfação do cliente. A curto prazo, podemos considerar: diminuição de tempo de ciclo e custos reduzidos; planejamento mais realista quanto ao cronograma; comunicação melhor entre equipe e cliente; e, muitas lições aprendidas. A longo prazo, os benefícios incluem: maior rapidez nas entregas e, com isso, diminuição dos riscos; confiança e satisfação do cliente; e, aperfeiçoamento contínuo (Veras, 2014).

O PMC pode ser implementado em uma empresa para melhorar a maturidade em gerenciamento de projetos, e a metodologia pode contribuir para o alinhamento estratégico.

A documentação de requisitos de negócio pode ser realizada por meio de artefatos como o Canvas, personas, histórias do usuário, entre outros. Falando especificamente sobre o Canvas, existe uma plataforma de design própria para facilitar o desenvolvimento do *software*. Ao criar um Canvas do produto, é

possível combinar a metodologia ágil com a experiência do usuário, complementando histórias de usuários com personas, *storyboards*, cenários, esboços de protótipos e outros.

O *storyboard* funciona como uma espécie de rascunho, que auxilia não apenas na construção da narrativa, mas também no planejamento do design e na composição das cenas, sendo ainda bastante útil na fase de edição, já na etapa de pós-produção. O visual desse esboço é semelhante ao de uma história em quadrinhos.

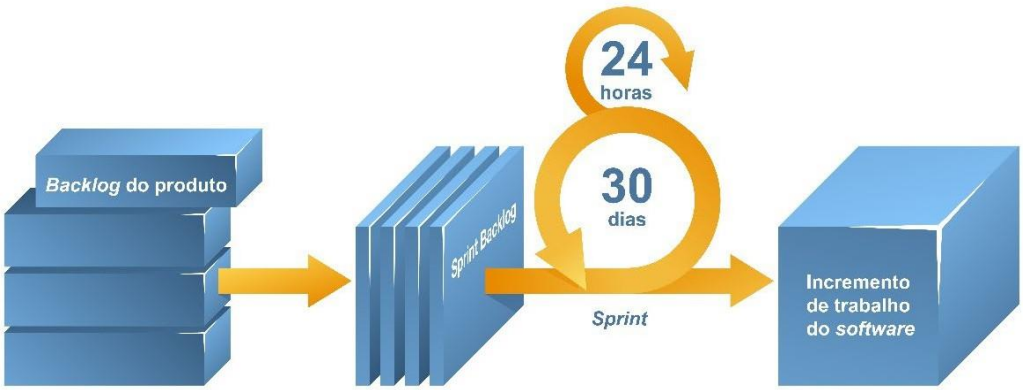
TEMA 4 – *BACKLOG* E PRIORIZAÇÃO

4.1 *Backlog* do produto

O *backlog* do produto contém todas as tarefas pendentes que precisarão ser executadas para a entrega final do *software* e para manter as entregas contínuas; por isso, ele precisa estar organizado e com as prioridades definidas. O *backlog* é adaptado e construído de acordo com os sucessivos refinamentos, novos requisitos, entregas, ou seja, ele muda constantemente.

A organização das tarefas listadas deve levar em consideração o mapeamento de dependências e a definição de ordem de execução; priorizar é imprescindível, como a Figura 9 demonstra pela metodologia *Scrum*. Para facilitar tal organização, falaremos sobre algumas técnicas que ajudam e facilitam nessa tarefa. Contudo, lembre-se de que há outras técnicas, e a escolha vai depender de vários fatores e da experiência do PO.

Figura 9 – *Backlog* do produto é uma lista de tarefas



Crédito: Dmitry Kovalchuk/Shutterstock.

O que pode preocupar o PO, por ser responsável pelo *backlog* do produto, é como gerenciar essa lista. Existem ferramentas muito úteis, e trataremos aqui da mais utilizada e conhecida: o *Kanban*.

O significado do termo vem das palavras japonesas *kan* (“placa”) e *ban* (“quadro”), ou seja, “placas de sinalização”. Seu objetivo, desde sua origem, sempre foi comunicar seu conteúdo de forma clara e concisa. Segundoi Taichi Ôno, o criador do Kanban, a ideia é “Produzir apenas o que é necessário, quando for necessário e na quantidade necessária”.

No desenvolvimento de *software*, os quadros Kanban ganharam espaço para representar as etapas do trabalho e o foco no fluxo estruturado, visível e flexível, auxiliando no gerenciamento.

Com o uso do Kanban e o aumento do conhecimento sobre seu funcionamento, ficou evidente que ele é útil não apenas para o desenvolvimento de *softwares*, mas para qualquer processo repetível, como manufatura, vendas, recrutamento etc. A partir disso, foram definidos alguns princípios básicos para o método Kanban:

- Visualizar o fluxo de trabalho;
- Limitar o trabalho em andamento;
- Medir e gerenciar o fluxo;
- Tornar as políticas de processos explícitas;
- Usar modelos para reconhecer oportunidades de melhoria do processo.

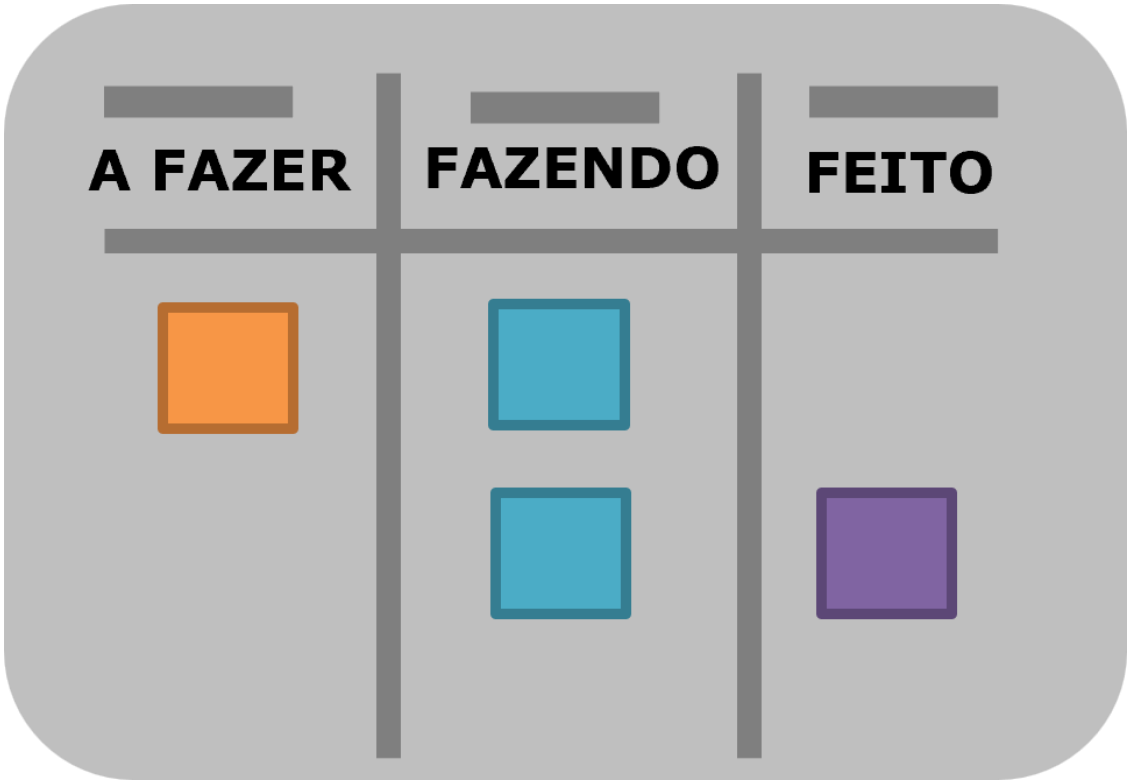
Para adotar a abordagem Kanban, alguns elementos precisam ser pensados para iniciar seu uso. O quadro Kanban pode ser um quadro branco, a parede, um quadro digital, enfim, algo visível para apresentar as informações para a equipe. O quadro deve conter colunas, para mostrar as etapas do processo, e linhas, que separam os projetos ou clientes. Os cartões Kanban representam as tarefas, que são alocadas na coluna correspondente ao estágio em que estão. São móveis, podendo ser deslocadas, dentro do quadro, para outras colunas, conforme são modificadas ou concluídas.

Algumas regras precisam estar claras para trabalhar com o Kanban, como acontece com qualquer método de gerenciamento de projetos. A movimentação dos cartões (tarefas) só pode ser feita pelos integrantes da equipe envolvidos com a respectiva tarefa. Se a tarefa estiver na fase de testes, apenas quem estiver testando é que poderá movimentar o cartão para a fase “testando”. Os

cartões devem conter apenas as histórias dos usuários (requisitos), lembrando que são resumidas, porém bem definidas.

A Figura 10 ilustra um quadro Kanban para o gerenciamento do desenvolvimento de *software*.

Figura 10 – Quadro Kanban com as tarefas



Créditos: shmai/Shutterstock.

4.2 Técnicas de priorização

Após coletar os requisitos iniciais do projeto, e pensando em metodologias ágeis, como decidir por onde iniciar e quais requisitos atender? Existem técnicas para incluir uma prioridade para cada tarefa:

- Priorizar por valor de negócio *versus* riscos;
- Priorizar por testes de suposição;
- Priorizar por *Business User Cost* (BUC);
- Priorizar por *scorecard*;
- Priorizar por MoSCoW;

Há outros métodos, como *kano model*, *story mapping* (“jornada do cliente”), *buy a feature* (“compre uma funcionalidade”) e RICE (“alcance, impacto, confiança e esforço”).

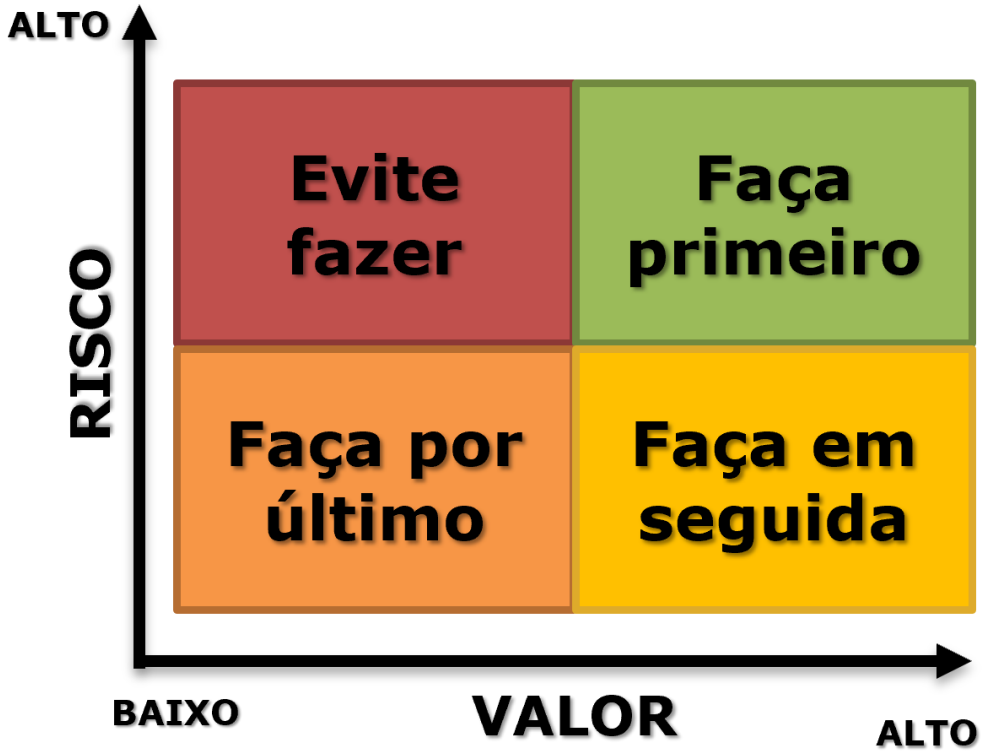
4.2.1 Método 1: valor de negócio *versus* risco

São definidas classificações de acordo com a prioridade dos requisitos, comparando-os com os riscos. Para priorizar, classifique e posicione os itens entre (Massari, 2014):

- Alto risco/Alto valor: faça primeiro esses itens;
- Baixo risco/Alto valor: faça em seguida esses itens;
- Baixo risco/Baixo valor: faça por último esses itens;
- Alto risco/Baixo valor: evite fazer esses itens.

Observando a Figura 11, há quatro quadrantes que podem ser classificados como alto ou baixo risco, alto ou baixo valor e, a indicação, conforme o quadrante, sugere como priorizar a tarefa, de acordo com o ponto onde está o cadastro: evite ou faça primeiro, ou faça por último ou faça em seguida.

Figura 11 – Priorizar *backlog* por valor de negócio e riscos



4.2.2 Método 2: testes de suposição

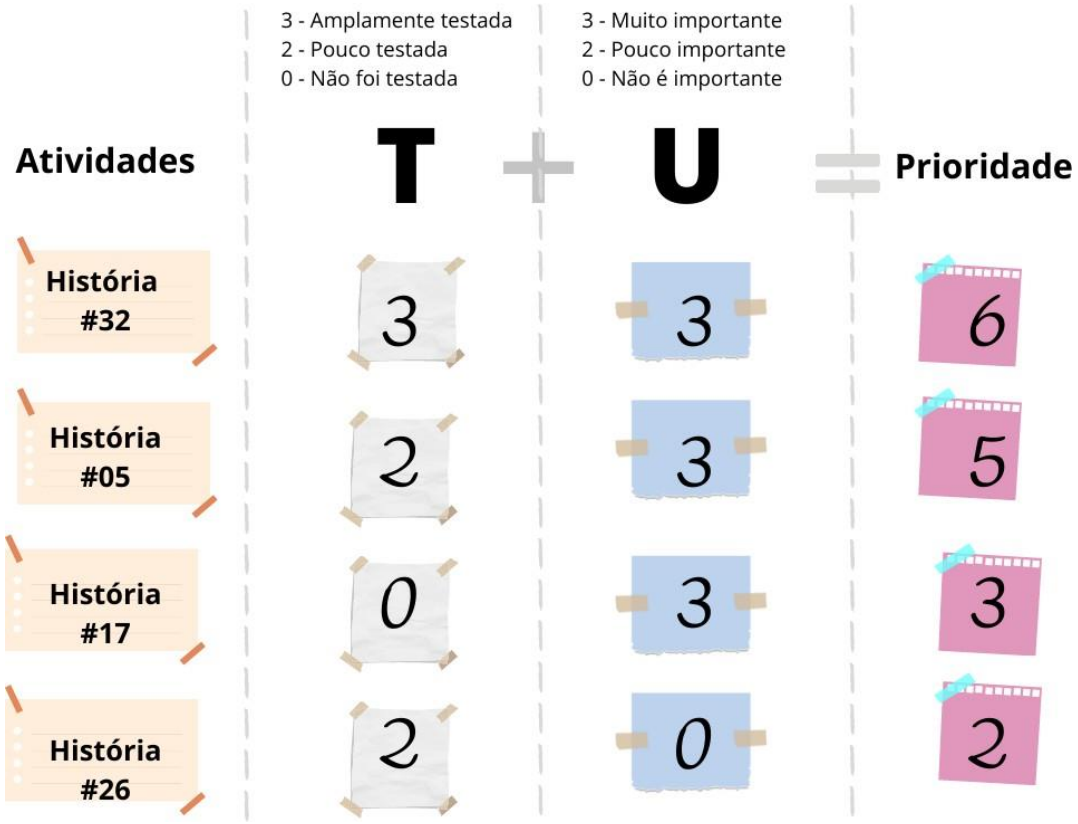
Esse método indica a prioridade de requisitos e funcionalidades por meio da validação de suposições (hipóteses) e de sua relevância para o usuário final,

utilizando critérios e uma escala de valores.

- T = o quanto a suposição foi testada
- U = relevância para o usuário

A prioridade será o resultado da soma dos critérios (Figura 12):

Figura 12 – Priorização de *backlog* de produto pelo método de suposição

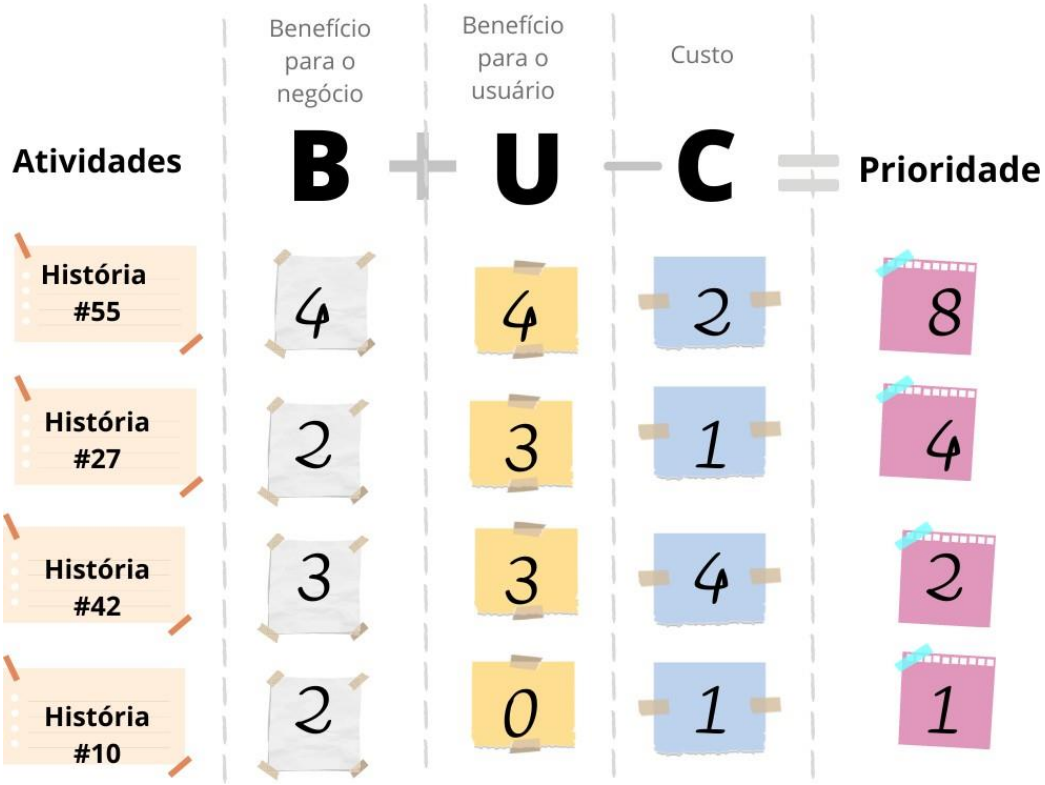


4.2.3 Método 3: por *Business User Cost* (BUC)

O *Business User Cost* (BUC) analisa os benefícios perante o negócio e o usuário em relação aos custos. Definimos uma escala para cada critério e utilizamos algumas perguntas-chaves, como: Qual o lucro que essa funcionalidade trará? Reduzirá custos da empresa? Tratá mais clientes? O usuário ficará mais satisfeito? O usuário quer essa funcionalidade? Quanto tempo será necessário? Quanto custará?

A Figura 13 apresenta um exemplo desse método de priorização.

Figura 13 – Priorização de *backlog* de produto pelo método BUC



4.2.4 Método 4: por *scorecard*

Esta técnica utiliza um cartão com critérios, seus respectivos pesos e suas funcionalidades (*features*), os quais recebem notas. Após calcular os pesos, teremos as notas de cada um; ordenando-s, teremos a lista do *backlog* priorizada. Observe, na Figura 14, um exemplo de *scorecard*.

Figura 14 – Priorização de *backlog* de produto por *scorecard*

Critério	Experiência do usuário	Aumento da receita	Melhorias	Retenção usuários	Total
História #4	60	80	80	80	74
História #1	50	40	80	80	54
História #2	80	60	40	20	49
História #3	20	80	60	60	42

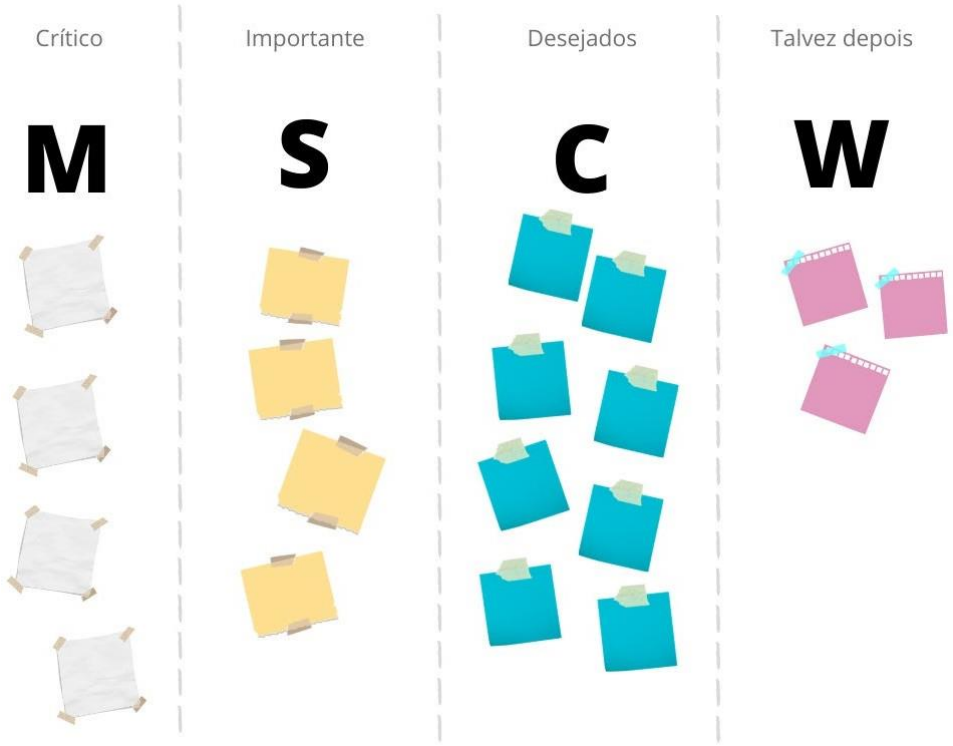
4.2.5 Método 5: por MoSCoW

Esta técnica atribui, aos requisitos de um *software*, os valores possíveis, como o seu próprio nome indica. As funcionalidades e requisitos atendem a quatro condições:

- (***M**ust-have*) “Deve ter”: é um requisito/funcionalidade imprescindível para a operação do produto de *software*;
- (***S**hould-have*) “Deveria ter”: não é um requisito/funcionalidade vital, mas é importante para o produto final;
- (***C**ould-have*) “Poderia ter”: é um requisito/funcionalidade que agrega valor ao produto, mas não é imprescindível;
- (***W**on't have*) “Não terá”: é um requisito/funcionalidade que pode agregar valor ao produto, mas não é vital e não trará diferencial.

A Figura 15 mostra um exemplo de apresentação desse método.

Figura 15 – Priorização de *backlog* de produto pela técnica de MoSCow



A escolha do método de priorização deve ser feita em conjunto com as partes interessadas, com os desenvolvedores, para entenderem as complexidades técnicas e verificar o ponto de vista de usuários e do cliente. Sempre é importante avaliar o objetivo e os requisitos, e aplicar a priorização do *backlog* do produto com frequência.

TEMA 5 – HISTÓRIAS DO USUÁRIO

O contexto de modelo ágil incentiva a colaboração e comunicação entre as partes envolvidas, buscando agregar valor de forma contínua e viabilizando, rapidamente, mudanças que surgirem.

Aquela documentação extensa é substituída por outras ferramentas mais simples e que cumprem esse objetivo, como a história de usuário. São muito interessantes por alinharem o cliente e a equipe de desenvolvimento, e utilizam uma mesma linguagem, facilitando a comunicação e transmitindo necessidades com clareza (Massari, 2014).

Na abordagem ágil, uma história de usuário é uma descrição curta, informal e em linguagem simples sobre o que o produto de *software* deve fazer para obter o resultado esperado. A base de uma história de usuário precisa conter as seguintes informações:

COMO UM <usuario>

DESEJO <necessidade>

PARA QUE <objetivo>

Observe, na Figura 16, alguns exemplos de modelos para escrever histórias de usuário.

Figura 16 – Exemplos de histórias de usuários

Fonte: Rattmann/Canvas.

Para criar as histórias de usuário, devemos utilizar o conceito INVEST:

- ***I**ndependent* (“Independente”): pode ser implementada em qualquer ordem;
- ***N**egotiable* (“Negociável”): pode ser removida a qualquer instante se não for útil;
- ***V**aluable* (“Valorosa”): entregar valor;
- ***E**stimable* (“Estimável”): capaz de ser estimada;
- ***S**mall* (“Pequena”): caber em uma *Sprint* (duas a quatro semanas);
- ***T**estable* (“Testável”): possível de ser validada.

Apesar de bastante úteis na comunicação dos requisitos, as histórias de

usuário ainda não contêm informações importantes sobre o comportamento do novo *software*, precisando ser complementadas com alguns critérios de aceitação.

A equipe de desenvolvimento avalia todos os cartões de histórias de usuários e os classifica em temas, decompondo-os em tarefas que serão executadas, além de estimar o esforço necessário para a codificação em termos de custos e tempo.

Introduzir metodologias ágeis em uma empresa exigirá mudança de cultura, e precisará de tempo para seu amadurecimento, experiência e conhecimento das equipes (Sommerville, 2018).

A maneira mais eficiente é iniciar com pequenos grupos, em projetos menores e com desenvolvedores entusiasmados pela abordagem ágil. Um projeto bem-sucedido desenvolvido com a prática ágil pode ser o fator chave para disseminá-lo na organização.

FINALIZANDO

Nesta abordagem, tratamos das metodologias ágeis para desenvolvimento de *software*; com elas, a coleta de requisitos é espalhada ao longo de todo o projeto, produzindo melhores resultados. No entanto, há de se considerar o porte do projeto, da empresa, e experiência e motivação da equipe.

Verificamos que o *Manifesto Ágil* surgiu para oferecer um conjunto de atividades que auxiliam na construção de soluções que levem em conta o *feedback* constante do cliente, e seu maior envolvimento ao longo do projeto, focando mais nas pessoas e na comunicação, e menos em processos e na quantidade de documentação. Na metodologia ágil, a ênfase inicial é o entendimento e visão geral do produto, especificando as atividades prioritárias, implementando o *software* em ciclos curtos, sempre revisando e preenchendo lacunas deixadas anteriormente. Os requisitos são dinâmicos e podem mudar ou aumentar; são revisados em cada ciclo do desenvolvimento, com refinamentos sucessivos e várias técnicas para entender as necessidades de todos os envolvidos com interesse no produto final.

Estudamos algumas ferramentas para avaliar os requisitos e refiná-los, compreendendo o que mais agrega valor para o cliente, como os *wireframes*, personas, jogos ágeis e histórias de usuários.

Depois, analisamos as principais práticas de mercado utilizadas para o planejamento e controle das várias etapas do projeto, por meio de abordagem ágil, como o Canvas.

Com tantas tarefas para implementar, tratamos de como armazenar a lista de atividades com o nome de *backlog* do produto e a importância de priorizar as tarefas de acordo com as necessidades do cliente. Analisamos vários tipos de priorização, e entendemos que a escolha do método certo para o projeto deve ser feita com as partes interessadas e com a equipe, observando o ponto de vista do cliente.

Por fim, concluímos que, em abordagens ágeis como o *Scrum*, os requisitos são escritos como histórias de usuários. A história de usuário é um objetivo a ser atendido ou uma explicação informal pela visão que o usuário tem do produto a ser construído. Ela representa os requisitos dos usuários, e a equipe de desenvolvimento avalia todas as histórias, classificando-as em temas e decompondo-as em tarefas a serem executadas.

Percebemos por meio dessa discussão que, no contexto de desenvolvimento ágil, a colaboração e comunicação entre as partes envolvidas são fundamentais para viabilizar as mudanças que surgem, e atingir o objetivo maior, que são as entregas contínuas e com qualidade e, como prevê a metodologia ágil, entregas rápidas, atendendo ao cronograma mais curto e dentro do orçamento previsto.

REFERÊNCIAS

FERREIRA, M. B. **Métodos ágeis e melhoria de processos**. 1. ed. Curitiba: Contentus, 2000.

FOGGETTI, C. **Gestão ágil de projetos**. 1. ed. São Paulo: Pearson, 2014.

FUNDAMENTOS do Kanban. **Kanban Tool**, 2022. Disponível em: <https://kanbantool.com/pt/guia-kanban/fundamentos-do-kanban>. Acesso em: 15 ago. 2022.

MANIFESTO FOR AGILE SOFTWARE DEVELOPMENT. Disponível em: <https://agilemanifesto.org/>. Acesso em: 15 ago. 2022.

MASSARI, V. L. **Gerenciamento ágil de projetos**. 2. ed. Rio de Janeiro: Brasport: 2014.

SOMMERVILLE, I. **Engenharia de software**. 9. ed. São Paulo: Pearson, 2018.

VERAS, M. **Gerenciamento de projetos**: Project Model Canvas (PMC). Rio de Janeiro: Brasport: 2014.

VIDAL, A. **Agile Think Canvas**. 1. ed. Rio de Janeiro: Brasport, 2017.