Desenvolvimento de Sistemas

Requisitos: técnicas de extração, testes A/B, MVP e *lean inception*

Introdução

Os requisitos de *software* nada mais são que a descrição dos recursos e das funcionalidades de um sistema que será desenvolvido para resolver um problema real. Tais requisitos transmitem as expectativas dos usuários do produto de *software*.

Conforme informações do *site* Acervo Lima (c2022), um requisito, com base no padrão IEEE 729, é definido da seguinte forma:

1. É uma condição ou uma capacidade necessária de um usuário para resolver um problema ou atingir um objetivo
2. É uma condição ou uma capacidade que deve ser atendida ou tida por um sistema ou um componente do sistema para satisfazer um contrato, um padrão, uma especificação ou outros documentos formalmente impostos
3. É uma representação documentada de uma condição ou de uma capacidade como em 1 e 2

Os requisitos descrevem como um sistema deve agir, aparecer ou executar. Para isso, quando os usuários solicitam um *software*, eles fornecem informações aproximadas do que o novo sistema deve ser capaz de fazer. Os requisitos diferem de

um usuário para outro e de um processo de negócios para outro.

Assim, no processo de desenvolvimento de *software*, a fase de requisitos é a primeira atividade de engenharia de *software*. É uma fase dominada pelo usuário e que traduz as ideias ou as visões em um documento de requisitos. Os requisitos são a base do projeto do sistema. Se não estiverem corretos, o produto final também conterá erros.

Técnicas de extração

Extrair requisitos significa buscar todas as informações necessárias para o desenvolvimento de um *software* a fim de atender às necessidades do usuário. A análise de requisitos pode parecer uma tarefa fácil, mas a comunicação entre o desenvolvedor e o cliente pode ser ambígua e levar à má interpretação das necessidades.

Além disso, é preciso considerar as diversas opiniões dos usuários, pois alguns podem ter necessidades específicas ou usarem as ferramentas de *software* de maneira bastante própria.

Logo, para realizar essa tarefa, existem várias técnicas que contribuem para obter um bom entendimento dos requisitos de *software*. Conheça a seguir (ou relembre) as principais:

Entrevista

É uma das técnicas mais tradicionais, gerando bons resultados na fase de levantamento de requisitos. É importante ter um plano de entrevista para que não haja distrações, as quais tornam a entrevista muito longa e pouco produtiva.

Brainstorming

É uma técnica utilizada para geração de ideias que consiste na exposição e na exploração de ideias. Um grupo de pessoas expõe, de maneira organizada, suas ideias, que devem ser ouvidas e anotadas de maneira visível para todos. Os participantes têm um papel fundamental e devem ser encorajados a combinar ou enriquecer as ideias.

JAD (*joint application design*)

É uma técnica que substitui as entrevistas individuais por reuniões em grupo, promovendo a interação da equipe de desenvolvedores com os usuários. Os desenvolvedores nem sempre saberão quais são os requisitos necessários para a produção do *software*, e por isso é tão importante a participação dos usuários, que fornecerão todos esses requisitos para que os desenvolvedores cheguem à conclusão do que será necessário para desenvolver o produto desejado.

Por fim, uma documentação padrão é preenchida durante as reuniões e assinada por todos no final, deixando registrado tudo o que foi realizado.

Prototipagem

É uma técnica na qual se utiliza um *software* já existente como referência para ajudar aqueles clientes que têm mais dificuldade de expressar as características do *software* que desejam.

Se não houver um *software* pronto que possa ser comparado ao produto final desejado pelo cliente, pode-se usar a prototipagem para criar um produto com características semelhantes às do produto desejado. Porém, lembre-se de que o protótipo deve ser um *software* diferente do produto final, e não uma versão inicial do *software* que será desenvolvido.

Pode ser necessário que o protótipo sofra mudanças para avaliar novas ideias, e, se ele se tratar do mesmo *software* que será entregue, essas mudanças podem resultar em *bugs* inesperados e em um produto com uma qualidade duvidosa. Assim, logo que o *software* for finalizado, o protótipo pode ser descartado.

A extração de requisitos é importantíssima para o desenvolvimento de um bom produto de *software*, pois, “para projetar *software* de boa qualidade, é necessário compreender como ele será utilizado e como sofrerá alterações ao longo do tempo” (PETERS, 2001).

Todas as técnicas de extração de requisitos apresentam vantagens e desvantagens que devem ser consideradas. Inclusive, a utilização de mais de uma técnica pode contribuir para o preenchimento de possíveis lacunas de levantamento, tendo em vista que nenhuma técnica é completa em razão das inúmeras variáveis de complexidade.

Análise orientada a objetos

Na fase de análise do sistema ou análise orientada a objetos do desenvolvimento de *software*, os requisitos do sistema são determinados e as classes e os relacionamentos entre elas são identificados. Com essa abordagem, tem-se as seguintes vantagens no desenvolvimento de *software*:

- É possível se concentrar nos dados, e não nos procedimentos.
- Os princípios de encapsulamento e ocultação de dados ajudam o desenvolvedor a desenvolver sistemas que não podem ser adulterados por outras partes do sistema.
- Permite-se o gerenciamento eficaz da complexidade do *software* em virtude da modularidade.
- É possível a atualização de sistemas pequenos para sistemas maiores com mais facilidade do que em sistemas construídos com base em uma análise estruturada.

As três técnicas de análise que são usadas em conjunto umas com as outras para a análise orientada a objetos são:

Modelagem de objetos

A modelagem de objetos tem como objetivo desenvolver a estrutura estática do sistema em termos de objetos, identificando objetos, classes e relacionamentos para o sistema. O processo pode ser visualizado nas seguintes etapas:

- Identificação de objetos e agrupamento em classes
- Identificação dos relacionamentos entre as classes
- Criação de diagrama de modelo de objeto do usuário
- Definição de atributos do objeto do usuário
- Definição das operações que devem ser executadas nas classes

Modelagem dinâmica

Após a análise do comportamento estático do sistema, o comportamento com relação ao tempo e às mudanças externas precisa ser examinado. O processo de modelagem dinâmica pode ser visualizado nas seguintes etapas:

- Identificação dos estados de cada objeto
- Identificação de eventos e análise da aplicabilidade das ações
- Construção de diagrama de modelo dinâmico, composto de diagramas de transição de estado
- Expressão de cada estado em termos de atributos de objeto
- Validação dos diagramas de transição de estado desenhados

Modelagem funcional

A modelagem funcional é o componente final da análise orientada a objetos, mostrando os processos que são executados em um objeto e a forma como os dados mudam à medida que se movem entre os métodos. O processo de modelagem funcional pode ser visualizado nas seguintes etapas:

- Identificação de todas as entradas e de todas as saídas de dados
- Construção de diagramas de fluxo de dados mostrando dependências funcionais
- Informação do objetivo de cada função
- Identificação de restrições
- Especificação de critérios de otimização

Lean inception

O principal problema com a criação de produtos no modelo tradicional é, além do tempo gasto, o engajamento de muitos recursos na organização, o que muitas vezes causa a perda do tempo de lançamento (também conhecido como *time to market*).

O *lean inception*, por outro lado, reduz o tempo de criação de produtos, algo fundamental para empresas que precisam de respostas rápidas para se manterem competitivas no mercado e de um direcionamento para o que realmente precisa ser feito.

A tradução da expressão "*lean inception*" é algo próximo de "começo magro" ou "começo enxuto", chamando-se assim por dois motivos:

1. A duração do processo inicial no desenvolvimento de *software* é menor, removendo tudo que não se refere ao produto, tornando-o "enxuto".
2. O resultado final do início é o entendimento do produto mínimo viável (mais conhecido como MVP, sigla em inglês para *minimum viable product*), conceito principal do movimento *lean startup*.

A aplicação do modelo *lean inception* combina os pilares do *design thinking* (colaboração, experimentação e empatia), buscando focar o aprendizado de personas e jornadas, com os pilares do movimento *lean startup* (construção, medição e aprendizado), com o propósito de auxiliar no entendimento e na construção do MVP e da avaliação arquitetônica, sempre considerando o contexto digital e tecnológico em que o cliente se encontra.

Uma "concepção enxuta" é útil quando a equipe precisa desenvolver iterativamente um MVP. A propriedade central de um MVP, embora seja um termo muitas vezes mal interpretado, é saber se vale a pena continuar construindo um

produto. Por isso, são escolhidos recursos com base no teste de suposições sobre o que é valioso para os usuários do momento. É preciso, assim, entender quem são os usuários, qual é a atividade que eles fazem que o produto suporta e como medir se eles acham o produto útil.

O *lean inception*, na prática, é um *workshop* dividido em várias etapas e atividades nas quais a colaboração é um ponto fundamental para direcionar a equipe na construção do produto ideal.

Lean inception na prática

O *lean inception* consiste em uma série de atividades, normalmente programadas ao longo de uma semana. Veja a seguir um exemplo de cronograma descrevendo cada um dos processos.

Este é um exemplo de cronograma, então não o trate como algo fixo. Encare-o como um bom exemplo de como as coisas podem fluir.

Observação: UX é a sigla utilizada para a expressão em inglês “*user experience*”, ou “experiência do usuário”, em português.

MVP

No desenvolvimento de sistemas, muitas vezes se concentram tanta atenção e tantos esforços no lançamento de novos recursos que se esquece de otimizar o recurso mais precioso: o tempo.

Quando você está trabalhando em um projeto, o objetivo de todos é alcançar mais com menos – agregar valor e ajudar o projeto a crescer com menos custos e com recursos razoáveis –, e, para tanto, é necessário focar ações certas nos momentos certos e reduzir ao máximo o desperdício.

Em desenvolvimento de *software*, o maior desperdício que pode existir é passar anos levantando requisitos e implementando um sistema que pode nunca vir a ser usado, pois ele resolve um problema que não interessa mais aos usuários. Sendo assim, se é para um sistema falhar, é melhor que falhe rapidamente, pois o desperdício de recursos será menor.

O que é MVP?

MVP, como já dito anteriormente, é a sigla em inglês para "*minimum viable product*", ou "produto mínimo viável", em português. Essa é a primeira versão de uma solução que conta apenas com os recursos necessários para o funcionamento.

Uma premissa fundamental por trás da ideia do MVP é você produzir um produto real que possa ser oferecido aos clientes e observar o comportamento real deles com o produto ou o serviço. Então, o principal benefício de um MVP não poderia ser outro: entender o interesse dos clientes no produto.

Além de permitir validar uma ideia para um produto sem construir o produto inteiro, um MVP também pode ajudar a minimizar o tempo e os recursos que você poderia comprometer para construir um produto que não terá sucesso.

É importante ter em mente o “V” do MVP: o produto **deve** ser viável. Isso significa que deve permitir que os clientes concluam uma tarefa ou um projeto inteiro e deve fornecer uma experiência de usuário de alta qualidade. Um MVP não pode ser uma interface de usuário com muitas ferramentas e recursos semiconstruídos; deve ser um produto de trabalho que a empresa seja capaz de vender.

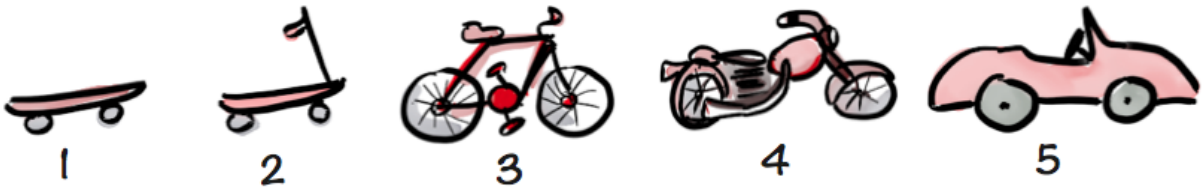

Figura 10 – Exemplo de evolução de um MVP
Fonte: Kniberg (2016)

Imagine um cenário no qual a necessidade do cliente é ir do ponto A até o ponto B mais rápido. Como solução, o cliente solicita a construção de um carro. Essa seria uma ótima solução, mas entregá-la levará tempo até que o produto final (carro) esteja pronto. Então, enquanto o produto não é concluído, como você pode solucionar o problema desse cliente agora? Para esse projeto, serão realizadas iterações com o cliente, que serão as entregas.

Comece pensando em um *skate*. Isso mesmo! Se o problema do cliente é ir do ponto A ao ponto B mais rápido e você precisa solucioná-lo agora, entregue a menor coisa para começar a solucionar o primeiro problema e em seguida colete *feedbacks* – em outras palavras, entregue uma solução mínima e viável.

Figura 11 – Primeira iteração
Fonte: Kniberg (2016)

É improvável que o cliente fique satisfeito, mas tudo bem. Nessa fase, a intenção não é tentar deixar o cliente feliz, e sim solucionar o problema dele e aprender durante o processo para, no fim, entregar a melhor solução possível. O *skate* pode não ser “ótimo”, mas é um produto viável e que ajuda o cliente a ir de A até B. O projeto não está finalizado; é apenas a primeira de muitas iterações.

Note como é possível aprender algumas coisas realmente surpreendentes. Suponha que o cliente diga que odeia o *skate*, você pergunta o porquê e ele diz que não suporta a cor. Então você questiona: “Somente a cor?”, e o cliente diz: “Sim, faça azul! O resto está tudo certo”. Você acabou de economizar muito dinheiro não construindo o carro, não é mesmo? Não é provável, mas é possível.

Depois de testar o *skate*, o cliente diz: “OK, é divertido. Ele me leva aos lugares mais rápido, mas é instável. Eu caio com muita facilidade”. Então, na próxima iteração, você tenta resolver esse problema ou pelo menos aprender mais:

Figura 12 – Segunda iteração
Fonte: Kniberg (2016)

Na segunda iteração, você entrega um patinete, e agora o cliente pode se locomover sem cair. Contudo, apesar de ser uma solução melhor, o cliente ainda precisa de um carro, pois o patinete serve apenas para pequenas distâncias. Então, na próxima versão, o produto se transforma em algo como uma bicicleta:

Figura 13 – Terceira iteração
Fonte: Kniberg (2016)

É possível que, a essa altura, o cliente descubra que a bicicleta já é suficiente, pois passar por áreas estreitas, por exemplo, é muito útil e não seria possível com o carro. Você acabaria, assim, economizando bastante tempo e dinheiro. Contudo, talvez o cliente queira mais – às vezes, ele precisa viajar para outra cidade, e o passeio de bicicleta é muito lento e cansativo. Então, na próxima iteração, você entrega algo que atende à necessidade do momento:

Figura 14 – Quarta iteração
Fonte: Kniberg (2016)

Novamente, talvez o cliente fique satisfeito com a motocicleta, e o projeto termine antes do planejado. Porém, de acordo com a avaliação do cliente, é possível, de fato, que o projeto acabe exatamente com o mesmo carro originalmente imaginado. Ainda assim, é muito mais provável que você tenha *insights* vitais ao longo do caminho e acabe propondo algo ligeiramente diferente:

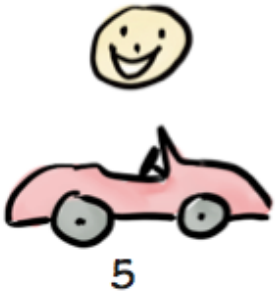

Figura 15 – Quinta iteração
Fonte: Kniberg (2016)

O cliente fica feliz! Por quê? Porque aprendeu, ao longo do caminho, que aprecia ar fresco no rosto, e então você acaba finalizando o projeto entregando um carro conversível. Afinal, o cliente conseguiu um carro, mas um carro melhor do que o planejado originalmente.

O que não é um MVP?

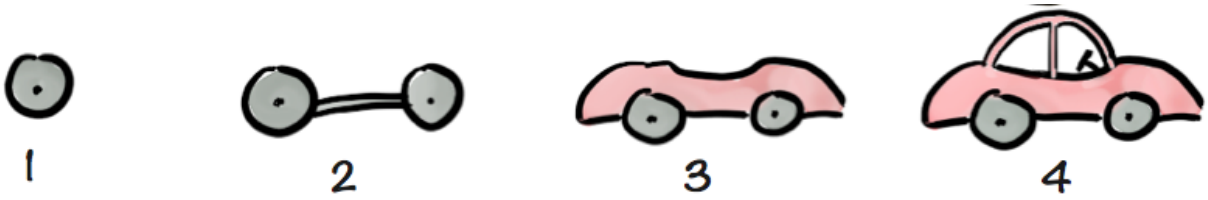

Figura 16 – Exemplo do que não é um MVP
Fonte: Kniberg (2016)

Muitos projetos falham, pois consistem em construir 100% do produto e entregá-lo completo no final. Contudo, quando o desenvolvimento ágil é apresentado como uma alternativa, as pessoas às vezes recusam a ideia de entregar um produto inacabado. Para exemplificar melhor, continue considerando o exemplo de projeto para o cliente ir do ponto A até o ponto B mais rápido.

Na primeira iteração, é preciso entregar algo, e, com o tempo que você teve para trabalhar no projeto, só foi possível entregar um pneu dianteiro.

Figura 17 – Primeira iteração
Fonte: Kniberg (2016)

A reação do cliente foi: “Por que você está entregando um pneu para mim? Eu pedi um carro! O que eu vou fazer com um pneu?”. O produto fica mais perto de ser feito a cada entrega, mas o cliente ainda fica irritado porque não pode realmente usar o produto.

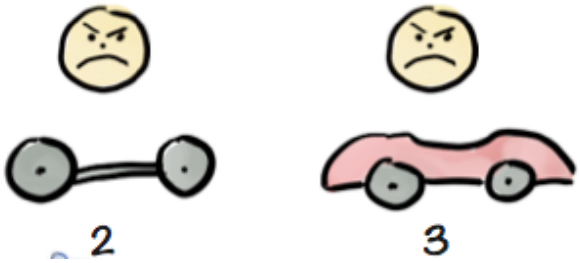

Figura 18 – Segunda e terceira iterações
Fonte: Kniberg (2016)

Finalmente, quando o produto está pronto, o cliente dá um *feedback*: “Obrigado! Até que enfim! Por que você não entregou o que eu queria em primeiro lugar e pulou todas as outras entregas inúteis?”.

Figura 19 – Quarta iteração
Fonte: Kniberg (2016)

Como você pôde ver, o cliente ficou satisfeito com o produto final, porque foi o que ele pediu. Porém, na realidade, isso geralmente não é verdade. Muito tempo se passou sem nenhum teste real do usuário, então o produto provavelmente está repleto de falhas de *design* com base em suposições incorretas sobre o que as pessoas precisam. Portanto, o “rosto sorridente” no fim é bastante idealista.

O MVP não é um plano, nem um rascunho, nem uma parte de um produto final. É, na verdade, uma versão totalmente funcional que está pronta para ser lançada no mercado. Os MVPs são divulgados publicamente para receberem *feedback* honesto do mercado e para se orientarem na direção certa.

Muitas vezes, as pessoas confundem um MVP com um protótipo, achando que é a mesma coisa. A diferença é que um protótipo é uma amostra ou um plano básico do produto ou do serviço. Normalmente, os protótipos são usados para explicar um conceito maior. Esse modelo é especialmente adequado para o desenvolvimento de *software*. **Você pode “transformar” o produto à medida que avança**, ao contrário, por exemplo, do *hardware*, no qual você basicamente precisa reconstruir todas as vezes.

No entanto, mesmo em projetos de *hardware* há um enorme benefício na entrega de protótipos para observar o cliente e aprender como ele usa o produto. O único ponto é que as iterações tendem a ser um pouco mais longas (meses em vez de semanas).

Até mesmo empresas automobilísticas reais, como Toyota e Tesla, fazem muitos protótipos (esboços, modelos em 3D – três dimensões –, modelos de argila em escala real etc.) antes de desenvolverem um novo modelo de carro.

Exemplos de MVP

Amazon

A Amazon é um dos exemplos de MVP mais bem-sucedidos. Jeff Bezos iniciou o mercado no início dos anos 1990 como uma livraria *on-line*. O *site* era um MVP que surgiu de uma lista de ideias de um *brainstorming* realizado por Bezos sobre quais eram os produtos que ele poderia vender com sucesso na Internet.

De 20 opções diferentes, Bezos escolheu cinco produtos, que incluíam vídeos, livros, *softwares*, computadores e até discos compactos, e criou um *site* MVP com foco inicial nesses produtos:

Figura 20 – Primeira versão do *site* Amazon

Fonte: Mansera (2017)

Facebook

A ideia inicial de Mark Zuckerberg era conectar todos do *campus*. Ele começou com o Facemash, um *site* no qual os usuários comparavam duas fotos e escolhiam a que gostavam mais.

Depois o *site* evoluiu para o chamado “Thefacebook”, um verdadeiro MVP das redes sociais, que foi aberto para uso em quatro das principais universidades americanas – Harvard, Stanford, Columbia e Yale. Após um ano de testes nos públicos segmentados, o acesso foi aberto para todos, e foi assim que começou a história do Facebook.

Figura 21 – Primeira versão do *site* Thefacebook

Fonte: Mansera (2017)

Uber

Quando a Uber (então chamada de “UberCab”) foi lançada em 2009, ela só funcionava em iPhones ou via SMS e estava disponível apenas em São Francisco, na Califórnia. O MVP da Uber foi suficiente para provar que a ideia de um serviço barato de carona compartilhada tinha mercado.

O aprendizado e os dados validados do primeiro aplicativo ajudaram a Uber a escalar os negócios rapidamente para onde estão hoje. Agora, a Uber está avaliada em 68 bilhões de dólares e é atuante em quase 80 países em todo o mundo.

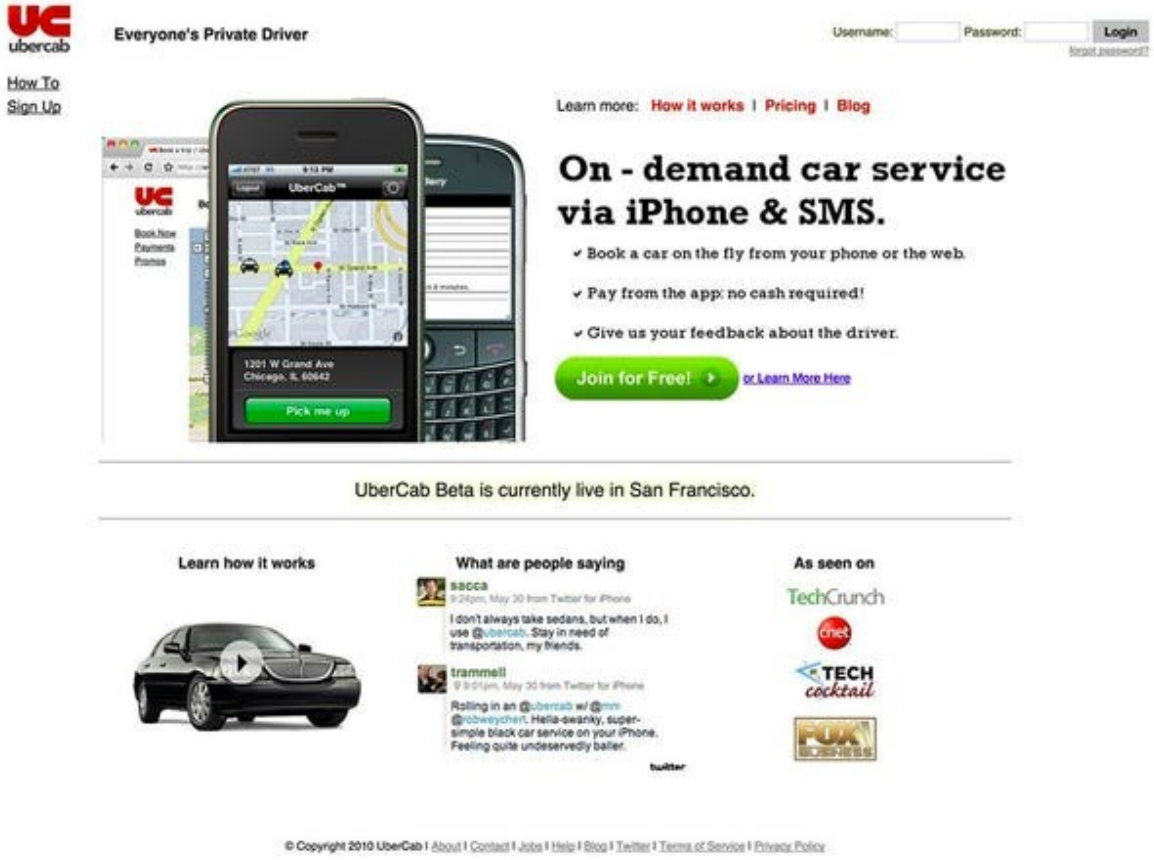

Figura 22 – Primeira versão do *site* UberCab
Fonte: McAlone (2016)

Construindo um MVP

A grande pergunta que surge quando se fala sobre MVP é: “Como saber quais são os recursos adequados para um MVP?”. A definição de MVP vem de duas palavras-chave: “mínimo” e “viável”.

- **Mínimo** – Significa que uma solução tem um conjunto limitado de recursos.
- **Viável** – Significa que os recursos são suficientes para que o aplicativo funcione bem.

Confira as etapas de criação de um MVP:

Etapa 1: comece fazendo uma pesquisa de mercado.

Por meio de pesquisas, analise se a ideia do MVP se encaixa nas necessidades de mercado. Observe também os concorrentes e como o seu MVP pode se destacar.

Etapa 2: idealize a agregação de valor.

Que problemas minha solução pode resolver? Por que as pessoas usariam minha solução? Que valor o novo produto oferece aos usuários? Com base nas respostas, você terá uma visão mais clara das qualidades e dos defeitos principais do seu produto e poderá definir o valor (não necessariamente monetário) dele.

Etapa 3: mapeie o fluxo do usuário.

Pense nos recursos básicos do produto que você precisa para atingir o valor proposto na etapa anterior. Você precisa olhar para o produto pela perspectiva dos usuários, desde a abertura do *software* até o processo final.

Etapa 4: priorize os recursos do MVP.

Quando se está desenvolvendo um MVP no contexto de *software*, é muito fácil desviar a atenção para outros recursos em vez de se manter no objetivo principal. Imagine, por exemplo, que o seu MVP se trata de um sistema de gestão de pedidos para um restaurante.

Objetivando entregar mais valor para o seu usuário, você pensa em outros recursos para o sistema, como autenticação de usuário, geração de extrato financeiro ou registro de atividades dos usuários.

Se o foco é “gerenciar pedidos”, você estaria ignorando o principal. Se o problema não é resolvido, não serão recursos extras que manterão seu usuário-alvo na plataforma. Agora, se você já tem essa solução pronta e não identificou nenhuma melhoria para ela, então pode concentrar seus esforços em outras funcionalidades.

Assim, após identificar todos os recursos do MVP, você deve categorizá-los com base na prioridade:

- Prioridade alta
- Prioridade média
- Prioridade baixa

A melhor forma de fazer isso é por meio de um *product backlog*, no qual todos os recursos podem ser mapeados, as prioridades podem ser definidas e o progresso do MVP pode ser acompanhado.

Para criar seu próprio *product backlog*, você pode usar a ferramenta Trello.

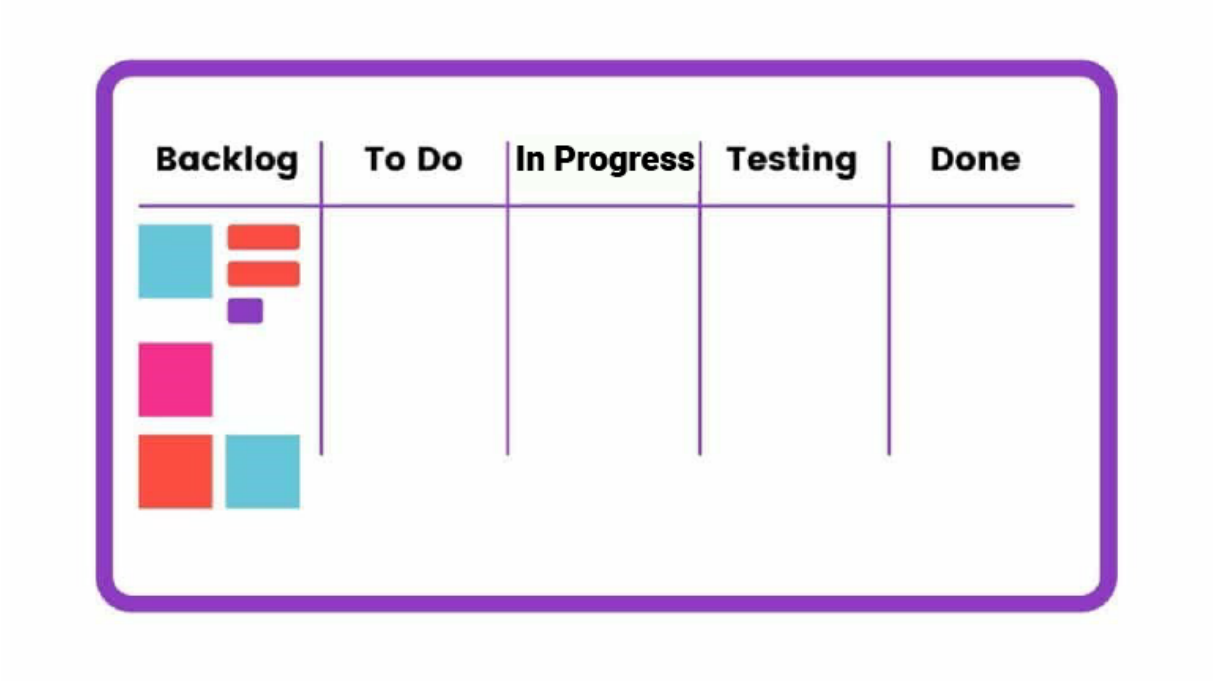

Figura 23 – Exemplo de *product backlog* Fonte: Souza (s.d.)

Etapa 5: construa e lance.

Depois de definir todos os principais recursos e aprender as necessidades do mercado, você pode iniciar a construção do seu MVP. Tenha em mente que um MVP não deve ter qualidade inferior à de um produto final, pois precisa atender às necessidades do cliente.

Etapa 6: analise o *feedback*.

Construindo um MVP, lembre-se da regra mais importante: você deve desenvolver um produto bem balanceado de acordo com os desejos de seus clientes. Em seguida, você pode continuar o fluxo de desenvolvimento e adicionar mais funcionalidades com base no *feedback* recebido dos primeiros usuários e nos dados analíticos coletados.

Além disso, esteja pronto para fazer melhorias e *upgrades* constantes no MVP lançado para torná-lo mais relevante para o mercado.

Testes A/B

Os testes A/B, também conhecidos como *split tests*, são usados para escolher, entre duas versões de um sistema, aquela que desperta mais interesse nos usuários. As duas versões são praticamente idênticas, porém uma implementa um requisito A e a outra implementa um requisito B. Em outras palavras, o objetivo é descobrir o requisito que deve ser adotado no sistema.

Testes A/B são muito úteis quando se constrói, por exemplo, um MVP, pois permitem que os desenvolvedores descubram as funcionalidades que mais atraem os usuários e aquilo que entrega mais valor no *software* em questão. Outro cenário bastante comum são testes A/B que envolvem componentes de interfaces visuais.

Imagine, por exemplo, dois *layouts* da página de entrada de um *site*. Um teste A/B pode ser usado para decidir qual deles atrai mais os usuários e quais são os elementos responsáveis por esse engajamento (cores, botões, textos usados, posicionamento etc.).

Para realizar testes A/B, é preciso haver duas versões de um *software*, chamadas de “versão de controle” (sistema original com requisitos A) e “versão de tratamento” (com requisitos B).

Tenha sempre em mente que esses sistemas devem ser praticamente idênticos, e não dois sistemas totalmente diferentes. Um exemplo prático seria um sistema de *e-commerce* (comércio eletrônico) que tem um algoritmo de recomendação para recomendar produtos mais relevantes para o usuário.

Na versão de controle, é usado um algoritmo de recomendação tradicional, enquanto, na versão de tratamento, é usado um algoritmo diferente, supostamente mais eficaz. Logo, o teste A/B, nesse cenário, tem como objetivo descobrir qual é o melhor algoritmo de recomendação para ser incorporado ao sistema.

Encerramento

Muitos problemas que surgem ao longo do desenvolvimento de um sistema são causados por falta de entendimento do que precisa ser feito. Quando há um planejamento e uma modelagem adequados do *software* que será construído, é mais fácil encontrar os pontos de melhora e realizar as correções.

Os requisitos são essencialmente todas as informações realizadas para o funcionamento de um sistema que permitem que qualquer um conheça o *software*. Além disso, durante o levantamento de requisitos, é possível identificar quais são os recursos viáveis e inviáveis a serem aplicados no projeto. O objetivo é entender o que o cliente precisa e o que ele espera no final.

Portanto, pode-se afirmar que um levantamento de requisitos pode definir o sucesso ou o fracasso do time de desenvolvimento.

Este conteúdo proporcionou o aprendizado de diversas técnicas e abordagens para fazer o levantamento de requisitos e construir o produto certo. Em alguns casos, uma abordagem pode ser mais eficaz que outra, mas não se esqueça de que você pode sempre extrair um pouco de cada técnica e combiná-las para chegar a um resultado mais satisfatório.