EVSOLID · CONTEÚDOS

A IA deixou uma coisa óbvia: funcionalidade importa menos do que usabilidade

Sabe o que acho mais engraçado em toda essa revolução da IA?

Ela deixou algumas coisas tão evidentes que chega a ser um **“tapa na cara”** de quem trabalha com desenvolvimento de software.

Durante muitos anos, boa parte da discussão sobre sistemas girava em torno do:

“O que ele faz?”

Tem financeiro?

Tem CRM?

Tem workflow?

Tem relatórios?

Tem automação?

Tem integração?

Tem inteligência artificial?

A quantidade de funcionalidades praticamente definia o valor percebido de um software.

Só que isso está mudando muito rápido.

RESPOSTA DIRETA

O que você precisa saber

Com a IA reduzindo o custo de construir funcionalidades, usabilidade e disciplina de produto se tornam ainda mais decisivas. Software gera valor quando as pessoas conseguem alcançar benefícios reais com pouco esforço, contexto familiar e fluxos claros.

Neste artigo

1. Fazer a funcionalidade deixou de ser a parte mais difícil
2. O “o que” perdeu espaço para o “como”
3. Valor percebido = benefício percebido ÷ esforço percebido
4. O ClickUp deixa uma pergunta muito interessante
5. Produto também é cultura
6. Talvez o maior trabalho esteja em esconder a complexidade
7. E agora surge um problema novo: a IA pode acelerar o feature creep
8. Antes o custo do código nos obrigava a dizer “não”
9. Mais funcionalidade também significa mais decisões
10. A IA pode aumentar o numerador e destruir o denominador
11. Talvez produto agora seja muito mais sobre dizer “não”
12. O desenvolvedor vai precisar pensar cada vez mais como usuário
13. O usuário brasileiro precisa entrar nessa discussão
14. A IA aumenta o valor de UX, produto e conhecimento de mercado
15. E, para sorte de quem gosta de programar…
16. O código está ficando barato. Decidir está ficando caro.

Fazer a funcionalidade deixou de ser a parte mais difícil

Hoje conseguimos desenvolver um módulo de complexidade razoável em uma fração do tempo que levaríamos alguns anos atrás.

Coisas que poderiam consumir dias ou semanas de programação agora podem ter uma primeira versão funcional em poucas horas — algumas vezes, em menos de uma hora.

A IA escreve código, cria consultas, sugere estruturas, monta componentes, encontra erros e acelera absurdamente o desenvolvimento.

E é justamente aí que aparece a parte interessante.

Quando construir uma funcionalidade fica barato, **ter a funcionalidade deixa de ser o grande diferencial**.

O problema passa a ser outro:

como alguém realmente vai usar aquilo?

O “o que” perdeu espaço para o “como”

É relativamente fácil dizer:

“Vamos criar um módulo financeiro.”

Muito mais difícil é responder:

Como o usuário cadastra uma conta?

Quantos campos devem aparecer?

O que precisa estar visível imediatamente?

O que pode ficar escondido?

Precisa abrir outra tela?

Dá para resolver em dois cliques?

O usuário precisa conhecer algum conceito técnico para entender aquilo?

Como alguém que executa essa tarefa cinquenta vezes por dia gostaria de trabalhar?

E, principalmente:

isso faz sentido para a pessoa que realmente vai usar o sistema?

Esse é o ponto em que nenhuma quantidade de código resolve um produto ruim.

Você pode ter uma funcionalidade tecnicamente impecável e, ainda assim, entregar uma experiência péssima.

Valor percebido = benefício percebido ÷ esforço percebido

Talvez uma forma simples de entender boa parte do mercado de software seja esta:

Valor percebido = Benefício percebido ÷ Esforço percebido

Normalmente olhamos apenas para o numerador.

Quantas funcionalidades existem?

Quanto tempo economiza?

Quantos processos automatiza?

Quantos relatórios entrega?

Mas existe um denominador que frequentemente ignoramos:

quanto esforço o usuário precisa fazer para obter esse benefício?

E esforço não significa apenas quantidade de cliques.

Inclui tempo para aprender o sistema, configuração inicial, treinamento da equipe, mudança de hábitos, necessidade de tomar decisões, adaptação dos processos internos e

até o medo de fazer alguma coisa errada.

É perfeitamente possível criar um produto mais poderoso e, ainda assim, reduzir seu valor percebido.

Imagine dois sistemas.

O primeiro possui 100 funcionalidades.

É extremamente flexível.

Permite criar campos, automações, visualizações, workflows e inúmeras configurações.

Mas exige que alguém compreenda toda essa estrutura antes de conseguir utilizá-lo bem.

O segundo possui apenas 30 funcionalidades.

Mas o usuário entra e imediatamente entende:

Clientes → Tarefas → Agenda → Financeiro

Tecnicamente, o primeiro pode ser muito superior.

Mas, para determinados usuários, o segundo entrega valor mais rápido.

E isso muda completamente a discussão.

O melhor produto não é necessariamente aquele que possui o maior benefício potencial.

É aquele em que o usuário consegue **capturar esse benefício com o menor atrito possível**.

O ClickUp deixa uma pergunta muito interessante

Olhe para o ClickUp.

É uma plataforma extremamente poderosa.

Tem tarefas, projetos, documentos, dashboards, automações, formulários, campos personalizados, múltiplas visualizações e uma infinidade de possibilidades.

Sob vários aspectos, consegue fazer muito mais do que inúmeras plataformas brasileiras.

Então surge uma pergunta que nós, desenvolvedores de software no Brasil, deveríamos estudar com muito cuidado:

por que uma ferramenta tão poderosa não simplesmente eliminou produtos brasileiros muito mais simples?

Se software fosse apenas:

funcionalidade × preço

isso provavelmente aconteceria.

Mas não acontece assim.

Porque capacidade técnica não é a mesma coisa que aderência.

Existe outra variável:

o esforço necessário para entender e adotar o produto.

Talvez uma plataforma brasileira faça muito menos. Ao avaliar ferramentas, vale considerar também como escolher um software de gestão para a empresa.

Mas o usuário entra e reconhece imediatamente a lógica do sistema.

A terminologia faz sentido.

Os fluxos parecem familiares.

O atendimento é próximo.

A implantação parece menos assustadora.

A interface se aproxima da forma como aquela empresa já trabalha.

E aí acontece algo interessante:

o produto tecnicamente inferior pode ter uma aderência maior.

Não porque o usuário seja menos sofisticado.

Mas porque o software exige menos energia para entregar o resultado que ele procura.

Produto também é cultura

Essa discussão fica ainda mais importante no mercado brasileiro.

Nós consumimos muita referência americana de SaaS.

E deveríamos.

Existem produtos excelentes sendo construídos lá fora.

Mas existe uma diferença enorme entre usar uma referência e imaginar que comportamento de usuário é universal.

Não é.

O brasileiro possui hábitos próprios.

Nossa relação com WhatsApp é diferente.

Nossa relação com PIX, boleto e cobrança é diferente.

Nossa relação com atendimento humano é diferente.

Nossa burocracia é diferente.

Nossa maneira de organizar pequenas, médias e até grandes empresas possui particularidades.

E a forma como muitas equipes aprendem a utilizar software também é diferente.

Um produto pode ser perfeitamente lógico para uma empresa de tecnologia americana e pouco intuitivo para uma empresa brasileira que passou dez anos organizando sua operação em WhatsApp, Excel, e-mail e papel.

Isso não significa que devemos construir software simplório.

Significa que precisamos construir software **sofisticado sem obrigar o usuário a sentir toda essa sofisticação**.

Talvez o maior trabalho esteja em esconder a complexidade

Esse ponto é fundamental.

Um sistema pode ter dezenas de módulos, centenas de tabelas, milhares de regras de negócio e inúmeras automações.

O usuário não precisa enxergar tudo isso.

Na verdade, talvez um bom produto seja justamente aquele em que a complexidade fica do lado de dentro.

A complexidade deveria ser problema do software, não do usuário.

Quando o sistema transfere sua própria complexidade para quem utiliza, algo está errado.

Cada opção adicional exige uma decisão.

Cada configuração exige entendimento.

Cada tela exige interpretação.

Cada novo conceito aumenta o custo cognitivo.

Por isso, adicionar funcionalidades não significa automaticamente adicionar valor.



Escolhas de produto também consistem em remover o que não ajuda a pessoa a concluir uma tarefa.

E agora surge um problema novo: a IA pode acelerar o feature creep

Durante anos, o excesso de funcionalidades já era um problema conhecido em desenvolvimento de software.

Existe até um nome para isso:

feature creep.

É quando o produto vai acumulando funcionalidades, opções, menus, configurações e exceções até ficar progressivamente mais complexo.

Antes, existia uma espécie de barreira natural contra isso:

desenvolver era caro.

Uma nova funcionalidade exigia planejamento.

Exigia horas de programação.

Exigia orçamento.

Exigia priorização.

Alguém precisava decidir se aquilo realmente valia o esforço.

A IA começou a destruir essa barreira.

Hoje é muito fácil surgir uma ideia em uma reunião e alguém dizer:

“Isso é simples. Dá para fazer.”

E provavelmente dá mesmo.

Esse talvez seja um dos maiores perigos da IA para produtos de software.

Não estamos apenas acelerando o desenvolvimento.

Podemos estar acelerando também o **acúmulo de complexidade**.

Antes o custo do código nos obrigava a dizer “não”

Parece estranho dizer isso, mas a dificuldade de desenvolver funcionava como um filtro.

Se uma funcionalidade levaria três semanas, a equipe precisava discutir:

Vale a pena?

Quantas pessoas vão usar?

Qual problema resolve?

É prioridade?

Existe outra forma de resolver?

Com IA, uma parte dessa restrição desaparece.

Se a funcionalidade leva três horas, a tendência é simplesmente fazê-la.

E isso pode gerar uma nova lógica perigosa:

“Já que é fácil fazer, vamos fazer.”

Mas facilidade de desenvolvimento nunca deveria ser critério de produto.

Porque o custo de uma funcionalidade não termina quando o código fica pronto.

Depois ela precisa ser mantida.

Precisa funcionar com as próximas versões.

Precisa aparecer em algum lugar da interface.

Precisa ser documentada.

Precisa ser explicada ao usuário.

Precisa ser entendida pelo suporte.

Precisa conviver com centenas de outras funcionalidades.

Cada nova funcionalidade aumenta um pouco o universo de possibilidades do produto.

E possibilidades têm custo.

Mais funcionalidade também significa mais decisões

Imagine um sistema simples.

O usuário entra e possui três caminhos.

Agora adicionamos novas opções.

Depois outras.

Depois personalização.

Depois filtros avançados.

Depois automações.

Depois múltiplas visualizações.

Tecnicamente, o produto ficou muito melhor.

Mas agora, antes de executar uma tarefa, o usuário precisa decidir:

Qual visualização usar?

Qual filtro aplicar?

Qual status selecionar?

Qual automação configurar?

Qual template escolher?

Qual campo preencher?

Qual fluxo seguir?

A funcionalidade que deveria reduzir trabalho pode começar a produzir um novo tipo de trabalho:

tomar decisões sobre como utilizar o próprio software.

Essa é uma armadilha perigosa.

A IA pode aumentar o numerador e destruir o denominador

Voltando à fórmula:

Valor percebido = Benefício percebido ÷ Esforço percebido

Imagine um produto com:

Benefício: 80

Esforço: 15

Valor percebido:

80 ÷ 15 = 5,3

Agora adicionamos diversas funcionalidades.

O benefício sobe:

80 → 95

Excelente.

Só que o sistema fica mais difícil de aprender, configurar e utilizar.

O esforço sobe:

15 → 30

Resultado:

95 ÷ 30 = 3,1

Criamos um produto objetivamente mais poderoso.

Mas reduzimos seu valor percebido.

Esse é o paradoxo.

A IA pode permitir que o numerador cresça muito rapidamente.

Mas, se não prestarmos atenção, o denominador cresce ainda mais.

Talvez produto agora seja muito mais sobre dizer “não”

Com IA, criar ficou fácil.

Por isso, talvez a capacidade mais valiosa de uma equipe de produto passe a ser justamente o contrário:

não criar.

Não colocar aquele botão.

Não adicionar aquela configuração.

Não criar mais um campo.

Não criar um novo menu.

Não permitir dez formas diferentes de executar a mesma tarefa.

Não transformar toda sugestão de cliente em funcionalidade.

Isso exige maturidade.

Porque desenvolver algo dá sensação de progresso.

Remover algo nem sempre dá.

Mas um produto não melhora apenas quando ganha coisas.

Às vezes ele melhora muito mais quando perde.

O desenvolvedor vai precisar pensar cada vez mais como usuário

Talvez esse seja um dos maiores impactos da IA sobre equipes de desenvolvimento.

O programador deixa de gastar tanta energia perguntando:

“Como vou construir isso?”

e começa a precisar gastar muito mais perguntando:

“Isso deveria existir?”

Depois:

“Como alguém vai usar isso?”

E depois:

“Como faço para que essa pessoa quase não precise pensar para usar?”

São perguntas de natureza completamente diferente.

E talvez sejam muito mais importantes.

O usuário brasileiro precisa entrar nessa discussão

No nosso mercado, isso fica ainda mais evidente.

Antes de adicionar qualquer funcionalidade, deveríamos entender profundamente como o brasileiro realmente trabalha.

Não como acreditamos que ele deveria trabalhar.

Não como um artigo de UX diz que ele trabalha.

Não como uma empresa americana organiza seu processo.

Mas como ele trabalha de verdade.

Onde ele clica primeiro?

O que ele não entende?

Qual palavra utiliza?

Que informação procura?

O que pergunta ao suporte?

O que insiste em continuar fazendo pelo WhatsApp?

O que continua controlando no Excel mesmo existindo uma funcionalidade dentro do sistema?

Essa última pergunta talvez seja uma das mais importantes.

Se o usuário prefere manter uma planilha paralela em vez de utilizar aquilo que você criou, não adianta imediatamente concluir:

“Ele não aprendeu a usar.”

Talvez você precise perguntar:

“Por que a planilha é mais fácil?”

Essa resposta pode valer mais do que muitas reuniões de produto.

A IA aumenta o valor de UX, produto e conhecimento de mercado

Existe uma enorme ironia nisso tudo.

Quanto melhor a IA fica programando, mais valiosas ficam justamente as atividades que não são simplesmente programação.

Entender o problema.

Entender o usuário.

Observar comportamento.

Escolher prioridades.

Eliminar etapas.

Construir uma arquitetura sustentável.

Tomar boas decisões.

Conhecer profundamente o setor.

A IA consegue gerar cinco alternativas de interface rapidamente.

Mas alguém precisa decidir qual delas faz sentido.

Consegue criar uma nova funcionalidade em poucas horas.

Mas alguém precisa ter maturidade suficiente para dizer:

“Não vamos colocar isso no produto.”

Talvez essa seja uma das competências mais importantes da próxima geração de equipes de software.

E, para sorte de quem gosta de programar…

Ainda existem problemas suficientemente difíceis para nos divertir.

Sistemas grandes continuam tendo arquitetura, legado, performance, concorrência, segurança, integrações, consistência de dados e anos de regras acumuladas.

A IA ajuda muito.

Mas ajudar é diferente de assumir todo o contexto e a responsabilidade por uma decisão arquitetural.

Ainda existe uma enorme diferença entre:

“funciona”

e

“essa é uma boa solução para continuar funcionando daqui a cinco anos”.

Talvez isso também mude.

Provavelmente vai.

Mas, por enquanto, quem gosta da parte realmente difícil da engenharia de software ainda tem bastante trabalho pela frente.

O código está ficando barato. Decidir está ficando caro.

Talvez essa seja uma das principais mudanças provocadas pela IA no desenvolvimento de software.

Durante muito tempo, transformar uma ideia em código era caro.

Agora esse custo está despencando.

Só que isso não significa automaticamente produtos melhores.

Pode significar exatamente o contrário.

Podemos entrar em uma era de softwares gigantescos, cheios de funcionalidades produzidas rapidamente, mas cada vez mais difíceis de entender e utilizar.

Por isso, o diferencial começa a migrar.

Não será simplesmente quem tiver mais funcionalidades.

Será quem compreender melhor o problema.

Quem conhecer melhor o usuário.

Quem entender profundamente o comportamento do mercado em que atua.

Quem construir os melhores fluxos.

Quem souber esconder a complexidade.

E, principalmente, quem tiver disciplina suficiente para não transformar facilidade de desenvolvimento em excesso de produto.

Talvez o grande desafio da IA no software não seja:

“Até onde conseguimos construir?”

Mas:

“Até onde conseguimos construir sem tornar o produto pior?”

Porque, daqui para frente, fazer será cada vez mais fácil.

Escolher o que fazer continuará difícil.

Escolher o que **não fazer** talvez seja ainda mais difícil.

E é aí que produto, usabilidade e conhecimento real do usuário passam a valer muito mais do que simplesmente código.