Resenha – “Managing Technical Debt”, de Steve McConnell

O artigo *“Managing Technical Debt”*, escrito por Steve McConnell, fala de um tema super importante para quem trabalha com desenvolvimento de software: a famosa dívida técnica. O autor explica de forma bem clara o que isso significa, por que ela acontece e, principalmente, como as empresas podem lidar com ela sem se enrolar.

Logo de cara, McConnell compara a dívida técnica com a dívida financeira. É uma metáfora muito boa, quando uma equipe de software faz um atalho técnico para ganhar tempo, ela está “pegando um empréstimo”. Isso ajuda a entregar o produto mais rápido, mas, lá na frente, essa decisão cobra juros — ou seja, o código fica mais difícil de manter, aumenta a chance de bugs e o time gasta mais tempo para fazer mudanças simples.

O autor divide a dívida técnica em dois grandes tipos. A primeira é a dívida não intencional, que surge sem querer, como quando um programador júnior faz um código ruim, ou quando a empresa compra outra que já traz um sistema cheio de problemas. Já a segunda é a dívida intencional, quando a equipe decide conscientemente fazer algo mais rápido, mesmo sabendo que vai ter um custo no futuro. É aquela velha história do “vamos lançar assim mesmo e arrumamos depois”.

Dentro da dívida intencional, McConnell ainda separa em curto prazo e longo prazo. A de curto prazo é mais tática tipo, “precisamos entregar essa versão até sexta”. A de longo prazo é estratégica algo como “vamos focar só nessa plataforma por enquanto, e expandimos depois”. O ponto é que nem toda dívida é ruim: às vezes, faz sentido assumir uma, desde que você saiba o que está fazendo e tenha um plano pra pagar depois.

O artigo mostra também que há dívidas “focadas” e “não focadas”. As focadas são aquelas grandes, identificáveis como decidir pular uma etapa específica. Já as não focadas são o pesadelo do desenvolvedor: pequenos atalhos, variáveis mal nomeadas, falta de comentários, códigos duplicados. Esse tipo de bagunça é como dívida de cartão de crédito se você não controla, ela explode.

McConnell fala bastante sobre a ideia de “serviço da dívida”. Assim como na economia, toda dívida técnica tem juros. Quanto mais você adia o pagamento, mais caro fica manter o sistema funcionando. Em alguns casos, as empresas gastam tanto tempo apenas “apagando incêndio” que quase não sobra energia pra inovar. Ele até menciona um conceito parecido com o “índice de endividamento” — se uma empresa está gastando mais tempo resolvendo problemas antigos do que criando valor novo, é sinal de alerta.

Outra parte muito interessante é quando ele fala sobre transparência. Muitas vezes, o problema não é ter dívida técnica, mas o fato de ninguém saber exatamente o quanto deve. Ele sugere registrar cada atalho técnico como se fosse um “déficit” dentro do sistema de controle de bugs ou backlog do Scrum. Assim, dá pra acompanhar e priorizar o pagamento dessas dívidas com clareza. Ele até faz uma analogia boa: é como quando você passa o cartão várias vezes e só percebe o tamanho da fatura quando a conta chega.

O autor também comenta que nem todas as equipes têm a mesma “capacidade de crédito técnico”. Um time experiente, que escreve código de qualidade, pode se dar ao luxo de assumir mais dívida de forma controlada. Já um time que vive criando bugs sem querer está sempre no vermelho não pode pegar mais empréstimo técnico sem quebrar o sistema.

Outro ponto forte do texto é como McConnell aborda a comunicação entre técnicos e gestores. Ele observa que os executivos costumam ser mais tolerantes à dívida técnica do que os desenvolvedores. Pra resolver isso, ele recomenda usar uma linguagem que os dois lados entendam: falar de dívida técnica em termos de dinheiro e impacto no negócio. Por exemplo, em vez de dizer “nosso código está sujo”, dizer algo como “40% do nosso orçamento de desenvolvimento está sendo gasto só pra manter o sistema atual”.

McConnell também explica por que vale a pena pagar a dívida técnica não necessariamente toda, mas o suficiente pra manter o sistema saudável. Ele lista vários motivos práticos: reduzir o tempo de lançamento de novas versões, permitir adicionar novas funcionalidades com mais facilidade, melhorar a qualidade do código e diminuir o número de defeitos relatados por clientes.

No fim, ele alerta contra a ideia de fazer um “mutirão” pra resolver tudo de uma vez. Segundo ele, isso raramente funciona, porque vira um projeto caro e sem foco. A melhor estratégia é ir pagando aos poucos, incluindo tarefas de redução de dívida dentro do fluxo normal de trabalho. É como pagar parcelas de um financiamento sem deixar de viver.

O artigo termina com alguns exemplos numéricos bem práticos pra ajudar a decidir se vale a pena assumir uma dívida técnica. Ele mostra cenários comparando o custo de fazer um código “bom” agora versus o custo de fazer rápido e consertar depois e como, às vezes, uma terceira opção intermediária (rápida, mas bem planejada) é a mais inteligente. Essa parte é muito útil pra desenvolvedores e gestores que precisam tomar decisões sob pressão.

No geral, *“Managing Technical Debt”* é um texto essencial pra quem trabalha com software. McConnell consegue explicar um conceito complexo de forma bem direta, com exemplos que todo mundo que já participou de um projeto vai reconhecer. A leitura é leve, mas cheia de lições práticas principalmente a de que dívida técnica

não é algo pra ter medo, e sim pra gerenciar com consciência. O segredo é saber quando vale a pena “pegar emprestado” e, claro, ter disciplina pra pagar a conta depois.