Pois, você trabalha com codebase grande e provavelmente você já pediu para a IA fazer uma mudança e ela voltou com código completamente fora do contexto, ela gastou várias horas iterando até você desistir e fazer tu mesmo e aí você pensou, IA não funciona para mim, só funciona com codebase simples, vou desistir disso. A verdade é que a diferença entre ganhar 10 horas ou perder 10 horas corrigindo o código que a AI escreveu está na gestão do contexto. E nesse vídeo aqui eu vou te mostrar como usar a AI de forma segura para escrever código em codebases complexos com mais de 300 mil linhas de código e aumentar a sua produtividade. Eu sou o Valdemar Neto, eu falo sobre desenvolvimento de software e liderança técnica. Eu já trabalhei em grandes empresas de todo mundo, como o Totorx e a Atlassian, e também sou cofundador da Tech Leads Club, uma comunidade para devs sênior mais. Então, esse conteúdo aqui é baseado na minha experiência, em pesquisas e também em mais de 2.000 devs que eu tenho contato no meu dia a dia. Bom, o que a gente vai ver aqui de forma prática é técnicas avançadas de context engineering, spec driven e eu vou falar para vocês o que é RPI, que é um termo que as pessoas estão falando agora também e porque ele faz todo sentido. Todas essas técnicas são comprovadas, eu vou trazer aqui muita base de coisas que a gente já testou, também vou deixar referências para vocês verem de pessoas que testaram até pesquisas com mais de 100 mil desenvolvedores, CodeBases com mais de 300 mil linhas de código, eu também testei em CodeBases grandes, então tudo aqui é muito prático. A primeira coisa que a gente tem que falar é da janela de contexto, ela que comanda tudo aqui e isso aqui é a coisa principal que a gente tem que tomar cuidado. Primeira coisa, as LLMs são stateless, ou seja, elas não guardam estado, você sempre tem que passar o estado para elas. Quando você usa Cursor, Cloud Code, Ecopilot, ele sempre pega o contexto e passa no teu prompt, pega a resposta. Então, essa é a primeira coisa. As janelas de contexto, hoje, chegam até um milhão de tokens, mas era bem comum a gente ter só 200 mil tokens. Eu sugiro sempre tentar ficar nessas janelas de 200 mil tokens. Um milhão de tokens ter mais, ter uma janela maior, não quer dizer que é melhor, quer dizer que, na verdade, já tem mais chance de alucinar. Por quê? Se a LLM é stateless, se ela é uma máquina de probabilidade, quando você manda mais tokens, ela tem que calcular mais probabilidade. É que nem uma pessoa, você faz várias perguntas para ela ao mesmo tempo, ela vai se perder na resposta, ela vai demorar, ela vai responder sem sentido. A LLM é a mesma coisa, você mandar muitas coisas para ela, ela vai te dar uma resposta sem muito sentido. Então, o que a gente vai fazer aqui nesse vídeo? A gente vai olhar como otimizar o máximo para ficar dentro dessa aqui, que é a Smart Zone que a gente chama, e acima de 60%, a gente chama de Dumb Zone no contexto, porque assim... Tem muita coisa para ela pensar, ela provavelmente vai alucinar e a gente não quer ficar lá. Então, o sugerido por ferramentas, cursos, todo mundo é tentar ficar aqui nos 40% para ter uma boa efetividade, a gente vai ver quais técnicas a gente vai aplicar para isso. Imaginem esse caso aqui, isso aqui é um Codebase de exemplo que eu uso para o meu curso da Tech Leaders Club, E aqui é um

CodeBase que tem lógica de domínio complexo. Ele é um serviço de streaming similar ao Netflix. Então, tem o módulo que é o módulo de Billing, o módulo que é de conteúdo, o módulo de Identity, Autenticação e Autorização. Aqui eu tenho o Monolith, que é só de exemplo. Mas imagina o seguinte, imagina que... Eu comecei a usar IA agora, eu entro nesse projeto, eu vou começar a fazer prompts para IA fazer alterações aqui. Eu tenho a minha estrutura, como todo projeto, cada projeto sempre tem sua estrutura, sua arquitetura, seus trade-offs, as decisões que foram feitas... Cada projeto tem um contexto, tem uma estrutura... Assim como ser humano, a IA precisa entender. Se eu der um prompt nesse projeto aqui, a IA vai sair fazendo coisas De onde? Baseado, primeiro, no conhecimento que ela foi treinada da internet e ela vai ter que escanear todo o projeto, trazer tudo isso para o contexto para aí começar a tomar decisões. Isso é muito ruim. Um código modular como esse aqui, o meu no caso, até ajuda. Mas imagina um código como esse aqui, imagina que em vez de ser modular fosse só um monolitão que eu tivesse aqui. Aí eu tenho aqui dentro serviços, eu tenho vários serviços que não tem relação um com o outro, estão todos juntos e tal, é muito comum em MGC. Isso aqui então, galera, para a IA é horrível, um código que não é modular, ela não entende nada. Então, se você não tiver um bom guia, se você não tiver um bom design de código, cara, não tem como sair um bom output da IA. Então, agora a gente vai ver como melhorar isso. Mesmo que o teu código não tenha uma boa arquitetura, uma boa estrutura, não siga bons padrões, tu pode sim adotar a IA e ter uma boa produtividade. Bom, para a gente entender como mudar o contexto necessário para as LLMs e manter a janela de contexto um tamanho bom, a gente tem que entender dois conceitos muito importantes. O primeiro é Progressive Disclosure e o segundo é On Demand Loading. Progressive Disclosure é você ir entregando para a LLM de forma gradual as coisas que ela precisa, ela ir descobrindo conforme a mudança que ela está fazendo. E on-demand loading é configurar para ela carregar sob demanda certos arquivos baseados em prompts que você dá. Então, imagina esse caso aqui. Um caso de Progressive Disclosure é você ter arquivos específicos de markdown, de contexto, em diretórios diferentes ou separados por responsabilidade. Então, no meu root eu tenho as minhas guidelines genéricas e tal, mas eu tenho um módulo no meu projeto que tem regras diferentes. Lá dentro tem outro markdown específico para esse projeto, quando ela for fazer uma alteração, ela vai carregá-lo. Então, esse tipo de coisa ajuda bastante, ter arquivos de contexto por responsabilidade e no diretório certo para ela ir carregando. Outra coisa muito importante é on-demand loading, talvez a mais importante das coisas que a gente está falando. As ferramentas mais modernas, Cursor, Cloud Code, têm maneiras de você carregar contexto sob demanda para nutrir o contexto da LLM para ela saber o que fazer. Eu uso muito o Cursor, vou mostrar aqui na prática, e eu uso muito o Rules. As Rules do Cursor você pode configurar para ela sempre carregar, ou para ela carregar às vezes, ou ela carregar sempre, mas sob demanda carregar contexto extra. Eu vou mostrar como fazer isso. E o Cloud Code acabou de lançar as skills, e as skills são muito parecidas com as rules, você pode fazer a mesma coisa com markdowns. Tem um padrão chamado agents.md, ele não é tão flexível assim, eu ainda uso muito rules ou skills no Cloud Code para fazer isso que eu vou mostrar para vocês aqui. Então, olha só aqui agora que massa nesse Codebase. A primeira coisa que vocês vão fazer em qualquer Codebase é fazer ele gerar uma documentação básica. Pode pedir para ele mesmo gerar, para a própria AI

gerar, e depois alterar ali a estrutura. Então, a primeira coisa que vocês vão ter é algum tipo de Architecture Guidelines para o projeto de vocês, um Markdown, independente do que for. Esse aqui é o meu, então explica toda a estrutura do meu projeto, Os princípios que eu sigo e tal, está tudo aqui, a minha estrutura, todas nessas guidelines. Aqui são as guidelines gerais do projeto, ela até está meio grande. Depois, eu vou ter outros guidelines de outras coisas que eu vou colocando. Então, eu uso o Domain Driven Design, Estratégico. Também tenho aqui a teoria do Domain Driven Design Estratégico, que ele vai carregar sob demanda, também tenho aqui. Eu tenho como identificar domínios no meu Domain Driven Design, também tenho aqui dentro. Aqui também tem as guidelines, na verdade, essa aqui que é a guideline de como a IA vai identificar os domínios. Como analisar, também tem um guideline, o que é um domínio, o que é um subdomínio e tal, seguindo os princípios de domínio. Isso aqui são coisas do meu projeto, então vocês podem ter o que vocês quiserem. Feature Folders, como estruturar as pastas do projeto, também está aqui com exemplos de faça e não faça... Isso aqui são documentos que a gente vai nutrindo ao longo da vida do projeto. Modular Architecture Guidelines, Refectory Plans, tudo está aqui dentro. Só que ele não carrega isso aqui sempre. Aí, eu vou ter minhas rules aqui no curso... As rules carregam automaticamente, mas nem sempre elas são aplicadas. Por exemplo, a minha Architecture Rules, eu digo que sempre aplica ela. Está vendo? Quer dizer que ele sempre vai aplicar o Architecture Guidelines, o Feature Folders? Não. Eu digo sempre aplica, então o que ele faz? Quando ele carrega, ele carrega só esse contexto aqui, tá vendo? E aí, eu digo para ele... Você deve ler o Architecture Guidelines ou o Modular Architecture Guidelines quando estiver criando ou modificando módulos, tiver dúvidas sobre design patterns... Então, ele vai carregando sobre... Sob demanda, se ele está só alterando uma lógica de domínio, ele não vai carregar esses arquivos sob demanda, ele não precisa, mas ele sabe que ele pode carregar. Então, essa é a primeira coisa, nutrir o projeto e estruturar ele de uma forma que ele vá carregando coisas sob demanda. Então, aqui eu boto isso no contexto, don't always apply true, mas ele vai carregando isso quando ele precisa. Depois eu tenho outros tipos de regras que são regras que têm algum gatilho. Por exemplo, análise de complexidade. Essa eu boto always apply false, ou seja, ela não é aplicada, só é aplicada quando alguém diz, chama diretamente complexo análise e manda analisar uma complexidade. Tem outra aqui de identificar domínios. Então, essa daqui quando alguém pedir no chat identify domains, ele vai carregar e vai aplicar todas as coisas aqui. O que ele faz? Ele carrega aqueles arquivos que eu mostrei, o Domain Identification Guidelines, DDD Strategic Design Theory, tudo isso ele carrega para poder fazer. Então, o que eu quero dizer aqui? Ele vai carregando essas coisas sob demanda para aplicar quando necessário. E se você quer aprender tudo isso aqui na prática comigo, eu vou dar um workshop no dia 7 de fevereiro, ao vivo, de 6 horas, onde eu vou te mostrar como pegar um projeto complexo sem IA, preparar ele para usar IA, fazer mudanças em escala de segurança... Além disso, eu vou compartilhar vários desses guidelines que eu estou mostrando aqui, de arquitetura, de boas práticas... Então, aproveita, te inscreve aqui, são vagas limitadas, vai ser bem massa, vai ser bem imersivo, muito prático, muito mano a massa... Espero lá! Mas vamos voltar para o vídeo. E agora a gente vai para essa parte que é extremamente importante, que é o RPI. O que é RPI? Research, pesquisa, plan de planejamento e implementação. O que isso quer dizer? Primeira coisa que

a gente vai fazer, a gente vai focar em descobrir onde estão as coisas. Então, o que eu quero dizer aqui? A gente vai fazer um prompt primeiro para descobrir. Desse prompt, a gente vai gerar um plano, ou seja, a gente vai começar com um contexto grande. Nessa fase aqui, usar daqui a pouco uma janela de um milhão de tokens no Codebase grande para tentar funilar, até ok, mas tomem cuidado. No geral, eu tentei manter nos 200 mil e tentar manter o contexto baixo ali. O que o research vai fazer? Ele vai descobrir, vai pesquisar várias coisas e vai trazer as coisas que você buscou. Dessas coisas, você transforma isso em um plano. desse plano, tu vai implementando. Dessa maneira, tu consegue manter o teu contexto ali até 30%, 30 e poucos por cento e funciona muito bem. Por quê? Nessa fase aqui, tu pesquisou, tu validou, tu criou um plano, tu validou o plano, tu ajustou o plano. Quando a IA vai implementar, ela vai separar por partes nesse plano e ela não vai estar lidando com aquele contexto gigante, vou mostrar mais isso na prática. Outra coisa que é muito importante aqui, principalmente com quem trabalha com Cloud Code, com sub-agents, que você pode delegar uma coisa, que é muito bom, sub-agents no Cloud Code são muito bom porque você está numa janela de contexto, em vez de fazer algo nela, você delega para o sub-agent que vai lá, faz uma pesquisa para ti, te traz um output de volta, ou seja, em vez de você ter um monte de token no contexto, Ele fez toda a pesquisa e te trouxe só o output que pode ser só alguns tokens, isso é muito bom. Mas aqui pensa que sub-agents não é aquela ideia de tem um sub-agent que é front-end, back-end e não sei o que, não. Isso é gastar token todo. Você quer um sub-agent que seja específico de uma task, pode ser análise de complexidade, não sei, coisas que você precisa para a sua arquitetura, validador de arquitetura ou algo que vai te buscar quais arquivos não estão seguindo os padrões, você pode pensar dessa maneira, algo específico do que você está pedindo. Bom, agora eu vou te mostrar um exemplo prático da gente encher janelas de contexto e como isso é problemático, porque eu vou dar um prompt de quais arquivos não estão seguindo as guidelines do projeto e nem DDD tático. Só um exemplo, o meu projeto aqui é um projeto relativamente grande, tem bastante service, tem bastante coisa... Mas claro, não é um projeto como produção, mas vocês já vão ver o que ele vai fazer. Então, ele já carregou as guidelines, porque ele sabe quando alguém dá um prompt como esse, ele tem que carregar as guidelines certas e vai trazendo para o contexto. Olha só, meu contexto aqui já está em 43%, bastante coisa. E agora ele vai começando a carregar arquivo e vai botando para dentro. Então, ele está buscando bastante coisa. Aqui ele não chamou a LLM ainda, entendeu? Ele está carregando o contexto para ir chamar a LLM. Vamos lá, 50% e já me deu aqui o retorno agora, vamos ver. Então, se vocês forem ver aqui, foi 46%. O que eu quero dizer? Qualquer prompt que eu der para ele fazer alguma coisa nessa janela de contexto aqui, ele provavelmente vai alucinar, porque foi muita coisa que eu pedi para ele. Se eu pedir para ele mudar coisas, talvez se for relacionado, isso até vai acontecer, mas assim, ele já tem muita coisa no contexto, ele vai seguir botando mais coisas para poder fazer mudança. Isso Não é legal. O que a gente faz a partir de agora? Isso foi um research, ou seja, eu fiz uma pesquisa e tive um output com várias coisas que eu posso fazer. Então, aqui tem um plano recomendado, deletar transaction script, várias coisas do meu código e uma lista de coisas. Então, normalmente, sempre que a gente faz uma pesquisa, a gente vai tomar alguma ação. O que poderia ser uma ação aqui? Eu escolher uma coisa para começar. Eu

posso pedir para me dar uma coisa simples para eu fazer. Qual a coisa mais simples que eu possa fazer para começar? Vamos ver... Eu entendo que eu estou no research ainda, né? Então, a primeira coisa que ele disse que eu posso fazer aqui, de baixo impacto, é reorganizar uma estrutura de pastas para domain, que é core e model. Legal! Então, a partir daqui eu peço para ele gerar um plano disso. Outra coisa importante, sempre que fazer um plano, dizer quais são as verificações para fazer, build e os testes estarem passando. Outra coisa que eu sempre faço também em refatoração é garantir que a versão nova cobre 100% dos casos de uso que a outra versão cobria, porque às vezes ele tinha tira a coisa fora. E olha a janela de contexto como eu já estou, 53%. Está crescendo, né? Então, eu estou quase no meu limite. O que eu tenho que fazer a partir de agora? Afunilar para um plano de implementação e começar a partir dele. Opa, eu esqueci de botar no modo plano. Enquanto ele está fazendo o plano, outra coisa que vocês podem fazer também, que eu vou mostrar, é salvar esse output no arquivo. E a partir desse arquivo, vocês começarem um contexto novo para ele não ter que pesquisar tudo de novo. Mas o que é importante no plano, galera? O que o plano faz? Por que o plano é tão importante? Por que todo mundo, todas as ideias, todo mundo está adotando planos? Porque ele detalha exatamente o que ele precisa fazer. Olha, aqui tem o que você precisa rodar para validar, aqui você vai dar uma análise estrutural, aqui você vai ter o resultado esperado, vai ter os comandos que você vai rodar, vai ter como você vai mover os arquivos... Então, está tudo aqui detalhado o que ela vai precisar fazer, ou seja, ela não vai precisar fazer uma fase de pesquisa de novo. Ela não vai precisar carregar o contexto de novo. Isso é muito importante. Então, se eu der um build nisso aqui, ela vai carregar só esse contexto e vai aplicar ele. Isso é muito, muito importante para ela não alucinar. Mas isso não resolve todos os problemas, porque às vezes a gente tem uma mudança muito grande para fazer que vai impactar, vai gerar vários planos, mais de um plano e eu vou mostrar isso na prática. Mas antes disso, a gente tem que tirar uma dúvida muito clássica das pessoas quando eu falo sobre isso, é... Ah, mas isso é a mesma coisa que Spec Driven? Ou isso é diferente de Spec Driven? Não, Spec Driven é quando a gente usa uma spec como base para implementar coisas, então... Virou um guarda-chuva muito grande do spec-driven hoje, porque daí tudo isso aqui ficou dentro de spec-driven. Então, a gente tem os prompts detalhados, tem um PRD que você usa como base, tem os arquivos markdown, tem documentação, tem API specs... Tudo isso meio que virou a spec para escrever o código e o spec-driven virou meio que o padrão para a maneira que a gente desenvolve com LLMs nos dias de hoje. A gente sempre tem algum tipo de spec a partir da gente começa a implementar. Eu acho que ficou meio vago demais e as pessoas não sabem exatamente o que é spec-driven e tal... E esse formato que a gente está indo de research, planning e implement, ele é mais lógico, porque dentro dele você vai aplicar Spec Driven, você vai ter uma spec que você não vai ter, você pode ter um Research como base do Research, você pode ter um Design Doc, você pode ter um PRD, dali você vai gerar um plano... Então, é uma coisa mais lógica e é o caminho que as ferramentas estão indo. Então, você tem essa fase abstrata de pesquisa, a fase do planejamento e a fase de implementação. Agora eu vou te mostrar um exemplo muito massa de um contexto muito grande, de uma refatoração muito grande, como você vai fazer ela. Bom, imagina que eu quero mover um dos meus módulos que hoje não usa DDD tático. DDD tático é quando tem entidades e tal,

né? Então ele não tem, a minha parte subscription hoje não tem, Aqui eu já fiz, mas não tinha entidades, por exemplo, e tem um fluxo complexo dentro de um serviço, tem um serviço aqui, esse serviço aqui, ele tem mais de 13 serviços envolvidos numa transação. Eu queria refatorar para começar a usar também Driven Design Tático, foi parte do meu curso um exemplo que eu dei. Essa é uma refatoração muito grande. Um plano só vai ficar muito extenso, que um ser humano não vai conseguir revisar o output, vai ter que abrir um request gigantesco, a gente não quer isso. Então, nesses casos, a gente faz a parte de research e a gente salva em um arquivo que a gente chama de memória de longo prazo. Então, é esse caso aqui que tem... Refactoring change plan use case to use statistical DDD. É um exemplo. Então, o que eu fiz aqui? Em vez de fazer um plano e deixar o plano como eu mostrei para vocês ali antes, eu salvei ele em um Markdown. E esse plano aqui, Ele foi um research muito longo ali, que ele trouxe vários pontos de como fazer o problema e tal... Ele é extremamente explicativo... Aí, nesse tipo de plano, é muito importante você manualmente ajustar algumas coisas... Então, aqui é uma refatoração que para um ser humano levaria meses para fazer. Olha o detalhe gigante... Botar entidades, alterar services... Eu não estou focando muito na implementação, tá gente? Estou focando no exemplo do plano. Então, isso aqui é um plano grande. Se eu mandar ele executar isso aqui, a janela de contexto vai explodir também. Não dá, não tem o que fazer. E outra coisa que é importante com esse tipo de coisa, é legal você compartilhar com o time. Revisa o meu plano lá, o plano está certo, beleza... Próxima fase é você quebrar esse plano em subplanos de implementação. Então, você vai dizer, pega isso daqui e separa em fases de implementação. Aí, você vai criar, por exemplo, aqui a fase 1. Outbox, Pattern e Event Bus. Fase 2, Value Objects. Fase 3, Aggregates. Então, você vai quebrar em subplanos e cada vez que você executar um desses aqui, a janela de contexto vai se manter pequena, porque ele é pequeno, ele tem uma coisa muito prática para fazer... O que você tem que fazer? O pré-requisito, o plano 1 está feito e dá um contexto para IA do que carregar, qual parte carregar do documento... Quais arquivos aqui, não precisa carregar os arquivos porque ele já sabe qual arquivo ele vai mexer, não precisa caminhar por toda a estrutura do projeto, os comandos que ele vai rodar, então tem um plano perfeito aqui, você consegue fazer uma refatoração que era gigante, consegue separar ela aqui, por exemplo, 6 pull requests que você vai mandar para o seu time revisar, Todas elas no final tem uma validação, como a LLM vai testar, então tu vai rodar um build, tu vai rodar um teste, tem que estar passando, só dá sucesso quando isso aqui passa. Então, galera, dessa forma tu consegue ter segurança para fazer refatorações de grande escala em CodeBases grandes. Esses aqui são os fundamentos que eu uso, tem funcionado muito bem. Por quê? Porque o ser humano está no loop validando os planos, validando o output, revisando por requests e tu consegue ter muita, muita agilidade seguindo esses planos aqui. Beleza? Espero que tenham gostado, comenta aqui como vocês fazem, que ferramentas vocês usam, vai ser bem massa.