codurance | CRAFT AT HEART

Além do Hype:
Adoção Estratégica de Código com IA para Líderes de Tecnologia

Autor: Matt Belcher
Head de Tecnologia Emergente da Codurance

codurance.com/pt

Índice

Sobre o Autor

Como Head de Tecnologia Emergente da Codurance no Reino Unido, Matt Belcher é responsável pela estratégia da empresa para aproveitar tecnologias emergentes. Ele garante que a região acompanhe as rápidas tendências tecnológicas.

Matt ajuda líderes empresariais a entender como essas inovações tecnológicas podem resolver desafios do mundo real. Com mais de 15 anos de experiência em consultoria de software, atuou em diversos setores, incluindo serviços financeiros, varejo e setor público, tanto no Reino Unido quanto na Europa. Ao longo desse tempo, trabalhou com uma ampla gama de tecnologias e desempenhou diversos papéis, desde desenvolvedor de software até CTO fracionado.

Matt é um palestrante frequente em conferências, meetups e painéis de discussão no Reino Unido. Também é escritor e recentemente foi citado no white paper da **Tech UK** intitulado *“Making AI Work for Britain”*.

Apaixonado por ajudar organizações a **maximizar seus investimentos em tecnologia**, ele foca na interseção entre **pessoas, processos e tecnologia** que, segundo ele, é onde os problemas mais interessantes geralmente estão.

Matt Belcher
Head de Tecnologia Emergente, Codurance
matthewbelcher
matthew.belcher@codurance.com

Prefácio

A história do desenvolvimento de software é uma história de revoluções, cada onda de inovação reformulando como escrevemos código, construímos sistemas e pensamos sobre complexidade.

A ascensão da internet foi um desses pontos de virada. Ela conectou sistemas e pessoas, acelerando não apenas o código que escrevemos, mas as expectativas colocadas sobre ele. Depois veio a era dos Ambientes de Desenvolvimento Integrados (IDEs) – Visual Basic, Delphi e o VisualAge da IBM para Java, que prometeram produtividade rápida através da simplicidade de apontar e clicar. Elas cumpriram em velocidade, mas frequentemente às custas da sustentabilidade. Bases de código tornaram-se emaranhadas com lógica gerada automaticamente que os desenvolvedores tinham dificuldade de entender, refatorar ou escalar.

Uma geração mais madura de ferramentas seguiu. IntelliJ IDEA, Eclipse e ReSharper não apenas apoiavam a escrita mais rápida de código, mas ajudavam a remodelá-lo melhor. Elas colocaram a refatoração no centro, enfatizando a manutenibilidade e o controle do desenvolvedor. Também vimos inúmeras plataformas de low-code e no-code surgirem com a ambição de democratizar o desenvolvimento. Essas ferramentas frequentemente brilham no início, permitindo protótipos em horas, mas frequentemente cedem sob o peso da complexidade que elas próprias criam.

Em cada ciclo, a promessa de progresso veio com uma armadilha: à medida que nossas ferramentas se tornaram mais poderosas, elas frequentemente também introduziram novas formas de complexidade acidental. Hoje, estamos na beira de outra revolução: a ascensão da programação assistida por IA.

Alguns argumentam que ferramentas de codificação com IA como Copilot, Cursor, Amazon Q e outras marcam o começo do fim da programação como a conhecemos. Seus evangelistas falam de “vibe coding”, um novo modo de desenvolvimento onde os desenvolvedores interagem com a IA em linguagem natural, gerando aplicações funcionais a uma velocidade impressionante. Para muitos engenheiros experientes, essas ferramentas oferecem um potente acelerador, fazendo-os começar mais rápido do que nunca. Mas essa velocidade frequentemente carrega um custo oculto.

Como este relatório revela, o código gerado por IA é maior, mais complexo e mais difícil de manter. Desenvolvedores se veem avançando rapidamente, apenas para alcançar um ponto onde percebem a necessidade de começar de novo. O código parece “legado” em poucas horas. O controle escapa, a menos que seja recuperado com disciplina deliberada. A habilidade não está apenas em acionar a IA, mas em permanecer como o arquiteto da solução.

Os insights compartilhados aqui são oportunos e vitais. Eles falam não apenas sobre o que as ferramentas de IA podem fazer, mas sobre o que é necessário para usá-las bem. Seja você um adotante relutante ou um convertido ao vibe coding, uma verdade permanece: essas ferramentas não são o fim da programação, mas sua próxima grande fase. Elas exigem novas habilidades, disciplina renovada e pensamento fresco sobre qualidade de software.

E como toda revolução que veio antes, aqueles que prosperam serão os que evoluem – com as mãos ainda firmemente no volante.

Mashooq Badar
Cofundador e Software Craftsman na Codurance

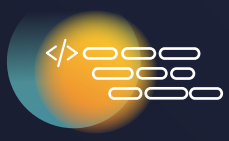
Resumo Executivo

Não faltaram afirmações ousadas sobre ferramentas de assistência de código com IA. De keynotes em conferências a demonstrações polidas, a mensagem foi clara: essas ferramentas estão prontas para revolucionar a forma como construímos software. Mas por trás do hype, permanece uma lacuna notável: a falta de uma análise fundamentada e baseada no mundo real sobre como essas ferramentas funcionam na prática, particularmente no contexto do desenvolvimento de software em nível empresarial.

Como uma consultoria de software que trabalha todos os dias com clientes para projetar, construir e modernizar sistemas complexos, a Codurance queria entender claramente qual impacto essas ferramentas provavelmente terão sobre o código que escrevemos, os fluxos de trabalho que seguimos e os padrões de engenharia que mantemos. Isso nos permitiria ajudar nossos clientes a desbloquear melhor o potencial dessas ferramentas enquanto mantêm uma disciplina de engenharia sólida.

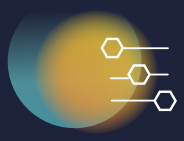
Para explorar isso, organizamos uma série de eventos práticos, focados e apenas para convidados com engenheiros de software seniores. Os participantes completaram dois desafios de programação: um usando ferramentas de assistência de código com IA e outro sem. Em seguida, realizamos uma análise estruturada das bases de código resultantes e coletamos feedback detalhado dos participantes.

Os principais pontos observados:

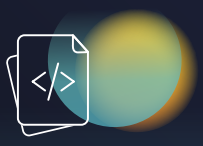


Soluções assistidas por IA tinham, em média, **4x mais linhas de código** do que versões sem IA.

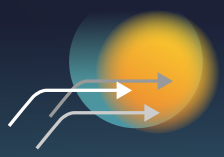
Os desenvolvedores experimentaram um claro **aumento inicial de produtividade** ao usar ferramentas de IA. Progresso rápido, menos fricção e entrega mais veloz.



Um tema comum foi a **perda de controle**. Vários desenvolvedores descreveram as ferramentas como “fugindo deles”, gerando código mais rápido do que podiam avaliar de forma significativa.

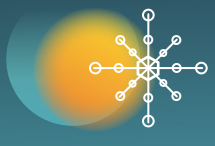


A duplicação de código foi mais comum, frequentemente impulsionada pela tendência da IA de gerar estruturas de código repetitivas.



No entanto, esse **aumento frequentemente se estabilizou rapidamente e caiu drasticamente**, à medida que fazer alterações em bases de código cada vez mais infladas ou desconhecidas se tornava mais demorado.

***Prompt engineering*[1] surgiu como uma habilidade crítica.** Aqueles que conseguiam articular intenções de forma clara e iterativa obtiveram melhores resultados.



A complexidade ciclomática foi quase 4x maior, tornando o código gerado por IA mais difícil de manter e compreender.

Nenhum dos participantes sentiu que havia dominado qualquer ferramenta de codificação com IA, destacando **uma lacuna de habilidades e a necessidade de capacitação estruturada** para extrair o máximo dessas ferramentas.

1 https://cloud.google.com/discover/what-is-prompt-engineering#what-is-prompt-engineering

No início, os desenvolvedores experimentaram um claro aumento de produtividade. As ferramentas de IA tornaram mais rápido e fácil começar e entregar algo funcional. Mas esse aumento frequentemente teve vida curta. À medida que os projetos avançavam, muitos desenvolvedores relataram que a produtividade estagnava e eventualmente diminuía, à medida que a complexidade gerada pela IA tornava o código mais difícil de entender, adaptar e controlar. De fato, um grupo comentou que, apesar de sua base de código ter apenas algumas horas de existência, já parecia que estavam trabalhando com uma base legada.

Um tema comum foi a sensação de perda de controle. Muitos dos desenvolvedores sentiram que a ferramenta de IA estava liderando o processo de codificação, em vez de assisti-lo. Nenhum participante alegou ter domínio das ferramentas utilizadas, e a maioria reconheceu que ainda estavam tentando compreendê-las, apesar de muitos já as utilizarem regularmente em seu trabalho diário. Isso reflete uma verdade mais ampla: como qualquer ferramenta de engenharia, a assistência de código com IA requer aprendizado deliberado para ser utilizada de forma eficaz. É irrealista assumir que simplesmente fornecer uma ferramenta de IA a um desenvolvedor levará automaticamente a um código melhor ou de maior qualidade.

O dia também destacou que a engenharia de prompt é uma habilidade fundamental. Desenvolvedores que conseguiam articular claramente suas intenções obtiveram resultados muito melhores do que aqueles que dependiam de entradas mínimas ou sugestões padrão.

Deve-se observar que, embora o foco principal deste relatório seja a assistência à geração de código, ferramentas de codificação com IA também oferecem outras funções. Estas frequentemente incluem recursos como geração de documentação e a capacidade de interrogar a base de código em linguagem natural, ambos altamente úteis em bases de código grandes e complexas.

Ferramentas de assistência de código com IA oferecem valor real e prático, especialmente em desenvolvimento inicial, estruturação e prototipagem rápida. Mas ainda não são substitutas prontas para produção ao julgamento de engenharia.

Em conclusão, **ferramentas de assistência de código com IA oferecem valor real e prático, especialmente em estágios iniciais de desenvolvimento, criação de estruturas e prototipagem rápida. Mas ainda não são substitutos prontos para produção em relação ao julgamento de engenharia.** Elas devem ser adotadas com cuidado, orientadas por diretrizes claras e apoiadas por capacitação em nível de equipe, se quiserem contribuir para uma entrega de software sustentável e de alta qualidade. É vital que essas ferramentas permaneçam como “assistentes” e que os desenvolvedores mantenham controle firme, o que requer disciplina e diretrizes claras para adoção.

Este relatório compartilha o que observamos, o que aprendemos e o que líderes técnicos precisam considerar a seguir.

Nota sobre Vibe Coding

O termo *Vibe Coding* ganhou enorme popularidade desde que foi criado em fevereiro de 2025 pelo cofundador da OpenAI, Andrej Karpathy. No entanto, seu significado original parece ter se perdido em meio ao hype e ao burburinho de marketing que o cercam. Para contextualizar, sua mensagem completa foi:

> ***"There's a new kind of coding I call "vibe coding", where you fully give in to the vibes, embrace exponentials, and forget that the code even exists. It's possible because the LLMs (e.g. Cursor Composer w Sonnet) are getting too good. Also I just talk to Composer with SuperWhisper so I barely even touch the keyboard. I ask for the dumbest things like "decrease the padding on the sidebar by half" because I'm too lazy to find it. I "Accept All" always, I don't read the diffs anymore. When I get error messages I just copy paste them in with no comment, usually that fixes it. The code grows beyond my usual comprehension, I'd have to really read through it for a while. Sometimes the LLMs can't fix a bug so I just work around it or ask for random changes until it goes away. It's not too bad for throwaway weekend projects, but still quite amusing. I'm building a project or webapp, but it's not really coding - I just see stuff, say stuff, run stuff, and copy paste stuff, and it mostly works."***

Escrever software de produção para sistemas corporativos, onde bom design, testabilidade e manutenibilidade são cruciais, é exatamente o oposto do *Vibe Coding*. Acreditamos que essa distinção é importante. *Vibe Coding*, por definição, entrega o controle total à ferramenta de IA, o que, em alguns casos, pode ser aceitável (como em protótipos rápidos). No entanto, na maioria das situações em que engenheiros de software escrevem código de produção para dar suporte a processos de negócio importantes, a qualidade do código é essencial.

Por esse motivo, é imperativo que os engenheiros permaneçam no controle do código gerado e, mais importante ainda, do design do software que estão desenvolvendo. *Vibe Coding* para qualquer coisa além de código descartável é um caminho rápido para o acúmulo de dívida técnica e para sistemas cada vez mais difíceis de modificar.

Introdução

As ferramentas de assistência de código com IA surgiram como um dos desenvolvimentos mais comentados na engenharia de software nos últimos anos. Seu potencial para acelerar entregas, reduzir trabalhos repetitivos e aumentar a produtividade dos desenvolvedores as tornou extremamente atraentes, especialmente em um momento em que as equipes de software estão sob pressão crescente para entregar mais, mais rápido, enquanto as lideranças executivas são pressionadas a “fazer mais com menos”.

No entanto, apesar de todo o barulho e das promessas de marketing, há surpreendentemente poucas evidências do mundo real sobre como essas ferramentas se comportam em condições típicas de desenvolvimento, além de demonstrações cuidadosamente preparadas e vitrines polidas. A maioria dos comentários até o momento se concentrou na produtividade individual ou em experimentações iniciais. Muito menos tem sido dito sobre as implicações dessas ferramentas para os fluxos de trabalho em equipe, a qualidade do código ou a entrega de software em escala empresarial.

O relatório *State of DevOps 2024*[2], por exemplo, indicou que, embora as ferramentas de IA possam melhorar a produtividade individual, elas podem prejudicar o throughput geral de entrega das equipes. Voltaremos a esse ponto mais adiante neste relatório.

Como consultoria de software que atua na interseção entre engenharia e entrega, quisemos ir além da teoria. Nossos clientes confiam em nós não apenas pela expertise técnica, mas também por um julgamento sólido. Para aconselhar com confiança sobre o uso de ferramentas de assistência de código com IA, precisávamos entender não apenas o que essas ferramentas podem fazer, mas o que elas de fato fazem quando utilizadas na prática, por desenvolvedores experientes em bases de código reais.

Para investigar isso, reunimos engenheiros seniores de diversos setores em uma série de eventos práticos. Cada participante recebeu dois desafios de programação: um com o apoio de ferramentas de assistência de código com IA, e outro sem qualquer suporte de IA. Os desafios foram projetados para simular tarefas típicas de desenvolvimento sob restrições de tempo. Após os eventos, conduzimos uma análise estruturada dos repositórios gerados e coletamos feedback direto dos desenvolvedores.

Este relatório compartilha os resultados desse experimento: o que observamos, o que aprendemos e o que isso significa para equipes de engenharia e líderes técnicos que desejam adotar essas ferramentas de forma responsável. Os achados destacam o valor que as ferramentas de IA podem trazer, mas também os riscos, a curva de aprendizado e o papel essencial da supervisão humana.

2 https://cloud.google.com/devops/state-of-devops

Metodologia

Para ir além da especulação e obter uma visão fundamentada de como as ferramentas de assistência de código com IA funcionam na prática, projetamos um exercício estruturado, com tempo limitado, envolvendo engenheiros de software experientes. Esta seção descreve os objetivos, abordagem, ferramentas e critérios de avaliação utilizados.

3.1 Objetivos

O exercício teve como objetivo explorar as seguintes questões:

1. Qual o impacto das ferramentas de assistência de código com IA na qualidade, complexidade e manutenibilidade do código?
2. Como a experiência do desenvolvedor difere ao usar ferramentas de IA em comparação com o trabalho sem elas?
3. Quais habilidades ou comportamentos são necessários para usar essas ferramentas de forma eficaz em um contexto empresarial?
4. Onde essas ferramentas oferecem maior valor e onde introduzem riscos potenciais?

Nossa intenção não era produzir um ranking definitivo das ferramentas, mas construir uma imagem mais clara de como essas tecnologias afetam os fluxos de trabalho reais no desenvolvimento de software.

3.2 Formato

O estudo foi conduzido como uma série de eventos presenciais de codificação. Os participantes eram engenheiros de software seniores de diversos setores, cada um com algum nível de experiência no uso de ferramentas de desenvolvimento com IA.

Cada participante completou dois desafios de programação:

- Um desafio utilizando ferramentas de **assistência de código com IA**
- Um desafio **sem qualquer suporte de IA**

Cada desafio teve duração aproximada de **2,5 horas.** Os participantes foram divididos em dois grupos – A e B, com o Grupo A usando ferramentas de assistência de código com IA pela manhã e o Grupo B à tarde. Os dois desafios foram projetados para ter complexidade e escopo comparáveis.

Os participantes submeteram suas soluções no GitHub, o que nos permitiu analisar as bases de código resultantes de maneira consistente e estruturada. Insights qualitativos foram reunidos por meio de observação direta, entrevistas informais e sessões de debriefing pós-desafio.

Os participantes foram aconselhados a trabalhar em pares, para simular o modelo de Programação em Par que é comumente usado em equipes de desenvolvimento Ágil em ambientes empresariais.

Os próprios desafios foram projetados para simular tarefas comuns apresentadas a desenvolvedores de software empresariais. Os participantes foram solicitados a construir uma aplicação de software que consistia em um frontend web, um conjunto de APIs de backend e um repositório de dados para armazenar o conjunto de dados que lhes foi dado. Eles estavam livres para implementar sua solução em qualquer linguagem de programação de sua escolha. Entre os participantes, as seguintes linguagens foram utilizadas:

- **C#**
- **TypeScript**
- **Python**
- **Java**

Os participantes receberam um conjunto de requisitos de produto que eles precisavam implementar. Esses requisitos incluíam tarefas como: busca e filtragem nos conjuntos de dados com base em certos critérios, exibição de itens individuais dentro do conjunto de dados, funcionalidade estilo carrinho de compras e design moderno de UI/UX.

3.3 Ferramentas Utilizadas

Embora tenhamos dado alguma orientação sobre ferramentas de assistência de código com IA que poderiam ser usadas, deixamos em aberto qual ferramenta os participantes adotariam. Ao longo do grupo, uma variedade de ferramentas diferentes foi utilizada::

- **GitHub Copilot**
- **Cursor**
- **Amazon Q**
- **Jetbrains Junie**
- **Roo Code**

Embora não seja estritamente uma ferramenta de assistência de código por si só, muitos dos participantes complementaram as ferramentas acima com o uso do ChatGPT também.

Principais Conclusões

Nossa análise das respectivas bases de código produzidas no dia, juntamente com o feedback coletado dos desenvolvedores ao longo do processo, nos permitiu reunir alguns insights interessantes. Nesta seção, resumimos esses achados em uma série de conclusões principais.

As bases de código eram significativamente maiores com assistência de IA

Em média, as soluções assistidas por IA continham **4x mais linhas de código** do que as submissões sem IA, apesar do mesmo tempo dado para ambas. Além disso, embora os desafios tenham sido deliberadamente projetados para exigir mais tempo do que o fornecido, aquelas soluções que foram escritas usando ferramentas de assistência de código com IA consistentemente implementaram mais das funcionalidades exigidas do que aquelas sem IA.

Complexidade ciclomática foi mais de 4x maior em código gerado por IA

Complexidade ciclomática é uma métrica que quantifica a complexidade do software ao medir o número de fluxos independentes através do código. Soluções assistidas por IA exibiram significativamente mais lógica de ramificação, caminhos condicionais e estruturas aninhadas.

Um aumento na complexidade ciclomática significa que o número de testes automatizados necessários também aumenta. Essa complexidade adicional também torna o código mais difícil de ler e entender. Mas, crucialmente, isso torna as modificações no código mais custosas e frequentemente resulta em design frágil.

Embora fosse esperado que as soluções com IA fossem mais complexas do que suas contrapartes sem IA, já que simplesmente continham mais linhas de código, **a quantidade de complexidade adicional foi maior do que o antecipado**. Embora as ferramentas de IA tenham ajudado os desenvolvedores a se moverem rapidamente, elas não os guiaram em direção a um design simples e código manutenível.

Duplicação de código foi mais prevalente nas submissões com IA

O código gerado por IA continha visivelmente mais blocos de lógica repetida e linhas semelhantes entre funções. Em vez de abstrair ou reutilizar lógica existente, as ferramentas de IA tipicamente regeneravam soluções semelhantes no próprio local, resultando em duplicação. Esse comportamento reflete sua arquitetura de geração token a token, que favorece a conclusão local em vez de decisões arquitetônicas. Como resultado, à medida que as bases de código cresciam, frequentemente havia grandes quantidades de código duplicado produzido. Em um ambiente corporativo, essa duplicação aumentaria o ônus de manutenção a longo prazo, tornando mudanças futuras mais propensas a erros, e reduzindo a eficiência geral do desenvolvimento. Em última instância, aumentando o custo e o risco conforme um sistema evolui dessa maneira.

Dificuldades com abstrações

As ferramentas de IA tiveram dificuldade em reconhecer e aproveitar abstrações existentes (como funções compartilhadas, componentes reutilizáveis ou interfaces) dentro da base de código. Em vez de identificar um padrão e reutilizar uma função ou componente existente, o comportamento padrão frequentemente tendia a gerar trechos de código duplicados ou quase duplicados. Essa tendência contraria o princípio DRY (Don't Repeat Yourself)[3] e levanta preocupações claras sobre a criação de ônus futuros de manutenção. Além disso, à medida que mais código era gerado, as ferramentas frequentemente falhavam em detectar oportunidades para novas abstrações. O código produzido era funcional, mas muito frequentemente não levava em conta bons princípios de design de software e oportunidades de aplicá-los.

Produtividade inicial foi alta, mas declinou rapidamente

Não há como negar, todos os desenvolvedores relataram que iniciar um problema com assistência de IA parecia significativamente mais rápido. A sobrecarga de escrever código boilerplate desapareceu, estruturas iniciais do projeto apareciam quase instantaneamente, e o progresso parecia incrivelmente rápido. No entanto, essa velocidade inicial frequentemente estagnava bruscamente. À medida que a complexidade crescia e a necessidade de mudanças mais sutis ou refatoração aumentava, integrar essas mudanças de forma suave com a base de código gerada por IA tornava-se notavelmente mais desafiador. O esforço necessário para guiar a ferramenta para a mudança específica correta parecia aumentar de forma não linear. De fato, um grupo que trabalhou com ferramentas de IA comentou que, apesar de sua base de código ter apenas algumas horas, já parecia que estavam trabalhando com uma base de código legado.

Desenvolvedores frequentemente se sentiam ultrapassados ou sobrecarregados

Talvez a observação mais sutil, mas crucial, foi o quão fácil pode ser ceder o controle inadvertidamente. Quando a IA está gerando código com aparência plausível rapidamente, a tentação de apenas aceitá-lo e seguir em frente é grande. No entanto, isso exige disciplina real. Sem revisão cuidadosa, pensamento crítico e uma mão firme guiando a direção geral, a IA pode facilmente começar a conduzir o processo de desenvolvimento, potencialmente levando por caminhos que não eram estrategicamente sólidos ou fáceis de manter. Manter o desenvolvedor firmemente no assento de arquiteto, usando a IA como uma assistente poderosa em vez de líder, é essencial.

A importância da Engenharia de Prompt

Ficou evidente que interagir de forma eficaz com essas ferramentas de assistência de código com IA é uma habilidade em si. Solicitações de alto nível e vagas deixavam os detalhes da implementação para a ferramenta de IA, que às vezes produzia resultados pouco úteis. Prompts

3 https://en.wikipedia.org/wiki/Don%27t_repeat_yourself

claros, específicos e ricos em contexto são essenciais para guiar a ferramenta de IA de maneira eficaz. “Engenharia de Prompt” pode parecer mais uma palavra da moda, mas o princípio subjacente de comunicar a intenção de forma clara e precisa é fundamental. Tratar a ferramenta de IA menos como uma varinha mágica e mais como um colega de equipe júnior incrivelmente rápido, mas um pouco literal, parece produzir melhores resultados.

Ciclo Planejar–Agir–Revisar

Houve um padrão interessante seguido por alguns grupos que chamamos de ciclo ‘Planejar–Agir–Revisar’, que pareceu gerar bons resultados. A etapa de ‘Planejar’ consistia em os grupos fornecerem os requisitos do desafio para a ferramenta de IA junto com algum contexto e pedirem para ela produzir um plano de implementação. Neste ponto, nenhum código era escrito, apenas um plano de como a solução poderia ser implementada. Esse plano então era revisado pelo grupo e, uma vez aprovado, o grupo passava para a etapa ‘Agir’. Aqui, uma parte do plano gerado pela IA era colocada em prática. O grupo pedia à ferramenta de IA que gerasse o código para aquela parte específica do plano de implementação. Uma vez gerado, o grupo então realizava uma etapa de ‘Revisar’. Aqui, o código gerado era analisado pelo grupo e, quando necessário, refinado. Cada uma dessas etapas era então realizada novamente em sequência até que o plano de implementação fosse completado.

Esse ato de quebrar o problema em componentes menores e ter uma etapa explícita de revisão, é claro, não é novidade no desenvolvimento de software. No entanto, essa abordagem ao trabalhar com ferramentas de assistência de código com IA permitiu ao grupo manter o controle da solução e não permitir que a ferramenta dominasse.

Fique atento ao tamanho dos lotes

Ligado aos achados mencionados anteriormente, muitos grupos acharam que a quantidade de código gerado pelas ferramentas de assistência de código com IA de uma só vez era excessiva. Sem um direcionamento cuidadoso, as ferramentas geravam alegremente volumes de código com aparência plausível tão grandes que geravam uma carga cognitiva tão alta para serem compreendidos de uma só vez, que parecia mais fácil aceitar cegamente as alterações do que tentar percorrer o grande conjunto de mudanças de código que havia sido produzido.

Código obsoleto

Nossa análise destacou alguns casos em que a ferramenta de IA havia gerado código obsoleto ou usava abordagens desatualizadas. Um exemplo que foi visto em dois repositórios foi o uso do ‘Create React App’, que foi descontinuado em fevereiro de 2025. Isso é algo que os desenvolvedores devem observar e que pode representar um possível risco de segurança, caso versões antigas de bibliotecas sejam utilizadas pela ferramenta de IA.

Recomendações

As ferramentas de assistência de código com IA oferecem grande potencial, mas apenas quando adotadas com intenção, supervisão e compreensão de seus pontos fortes e limitações. Com base em nossas descobertas, aqui estão nossas principais recomendações para líderes técnicos que desejam integrar essas ferramentas ao desenvolvimento de software empresarial de maneira responsável e eficaz.

1. Defina quando e como usar ferramentas de assistência de código com IA

Nem todo código é igual e nem todos os cenários de codificação se beneficiam igualmente da assistência com IA. Ferramentas de assistência e geração de código com IA realmente brilham em prototipagem rápida, criação de estrutura (scaffolding), geração de código boilerplate e provas de conceito (PoCs), onde a velocidade supera a necessidade de manutenibilidade a longo prazo.

Em situações onde qualidade de código e manutenibilidade importam, o uso dessas ferramentas deve ser cuidadosamente governado. É muito fácil entregar o controle à ferramenta de IA e ficar sentado enquanto ela gera volumes de código com aparência plausível. Ao fazer isso, a IA também está assumindo a tomada de decisões arquitetônicas para a base de código, e muito provavelmente não tomará as decisões corretas de design estratégico para o seu contexto específico e conjunto de trade-offs. É vital aqui que essas ferramentas apenas 'ajudem' e que o desenvolvedor permaneça no comando, garantindo que o design e a qualidade da base de código como um todo permaneçam no caminho certo.

Recomendamos estabelecer políticas de uso claras ou heurísticas que orientem as equipes sobre onde a ferramenta de IA agrega valor e onde ela pode criar risco.

2. Incorpore a Engenharia de Prompt ao conjunto de habilidades da sua equipe

A qualidade da saída da IA está diretamente ligada à qualidade da entrada fornecida pelo desenvolvedor. Os desenvolvedores devem aprender a expressar sua intenção de forma clara, decompor problemas incrementalmente e iterar com as ferramentas de IA. Engenharia de Prompt não é apenas para interfaces de chat com LLMs, ela também importa dentro do ambiente de desenvolvimento (IDE).

Recomendamos investir na capacitação em Engenharia de Prompt, construindo playbooks internos e facilitando o compartilhamento de prompts úteis entre as equipes. Considere os prompts como artefatos dentro do seu ciclo de vida de desenvolvimento de software (SDLC).

3. Trate o código gerado por IA como contribuição de um desenvolvedor júnior

As ferramentas de assistência de código com IA não são engenheiros seniores, elas são geradores de código entusiasmados com contexto limitado. O código gerado por ferramentas de IA deve ser revisado, testado e refatorado como qualquer outra contribuição vinda de um colega de equipe inexperiente porém bem-intencionado. Como resultado, os desenvolvedores devem ser encorajados a usar ferramentas de IA como colaboradoras, não como muletas.

4. Priorize a capacitação de desenvolvedores, não apenas o acesso

Em última análise, apesar do hype, esses assistentes de IA são fundamentalmente apenas ferramentas. Assim como dominar uma IDE complexa, um novo recurso da linguagem ou um framework de testes, desbloquear seu verdadeiro potencial exige um investimento consciente em aprender como utilizá-las de forma eficaz. Apenas ter acesso não se traduz automaticamente em ganhos significativos de produtividade ou em código melhor; a aplicação habilidosa vem com prática e esforço deliberado para entender suas capacidades e limitações.

Recomendamos que os desenvolvedores invistam tempo para realmente aprender a utilizar essas ferramentas corretamente, a fim de maximizar sua eficácia e também obter uma melhor compreensão sobre elas.

5. Monitore e meça os resultados certos

Linhas de código e produtividade percebida não são suficientes. Foque em indicadores significativos. Considere monitorar métricas como churn de código e tempo até a refatoração no código gerado por IA, juntamente com impactos em **métricas DORA**[4] como Lead Time e Taxa de Falhas em Mudanças.

Recomendamos configurar métricas leves ou pontos de verificação para monitorar o impacto que essas ferramentas estão causando no negócio.

6. Garanta pequenos lotes de mudanças

É extremamente fácil com ferramentas de assistência de código com IA gerar uma grande quantidade de código muito rapidamente. No entanto, embora isso possa ser desejável em alguns cenários, quando se está construindo aplicações empresariais dentro de uma equipe de entrega de software, isso pode, na verdade, ser prejudicial. De fato, o Relatório State of DevOps 2024 apontou isso explicitamente, afirmando que grandes lotes de mudanças de código gerado por IA podem ter um impacto negativo na produtividade geral da equipe e no fluxo de valor. Isso é importante de entender. Se, ao adotar ferramentas de IA, sua equipe agora está gerando

4 https://dora.dev/guides/dora-metrics-four-keys/

código em um ritmo muito mais rápido e em lotes muito maiores, será que o restante do seu ciclo de vida de desenvolvimento consegue se adaptar a isso sem sacrificar a qualidade?

Recomendamos garantir que, ao trabalhar com ferramentas de assistência de código com IA, os desenvolvedores mantenham os conjuntos de mudanças pequenos e façam commits frequentes ao repositório central. Em outras palavras, seguir as boas práticas de Integração Contínua.

7. Trabalhe de forma iterativa

Padrões como o ciclo 'Planejar–Agir–Revisar', mencionado na seção anterior, permitem que os desenvolvedores mantenham o controle do design do código e não permitam que a ferramenta domine. Dividir tarefas em subtarefas menores e passá-las para a ferramenta de IA também torna as mudanças mais gerenciáveis para revisão, seja refinamento manual ou pedindo que a ferramenta de IA gere o código com melhorias.

Recomendamos que os desenvolvedores sigam uma abordagem disciplinada ao utilizar ferramentas de assistência de código com IA, que permita um trabalho iterativo e revisão humana explícita – o chamado "humano no loop".

Conclusão

As ferramentas de assistência de código com IA não são mais teóricas; elas estão aqui, integradas aos fluxos de trabalho dos desenvolvedores modernos e capazes de acelerar dramaticamente certos tipos de trabalho com software. Nossa pesquisa demonstrou isso em primeira mão: engenheiros produziram mais código, mais rapidamente, e descreveram seu progresso inicial como mais fluido e menos exigente cognitivamente.

Mas velocidade não é o mesmo que qualidade.

O que ficou claro tanto pela análise do código quanto pela reflexão dos desenvolvedores é que, sem uso cuidadoso, o código gerado por IA frequentemente introduz complexidade oculta, duplicação e riscos de manutenção a longo prazo. As mesmas ferramentas que ajudam os desenvolvedores a começarem mais rápido podem tornar mais difícil terminar bem. Sem estrutura, habilidade e disciplina, as equipes correm o risco de confundir momento com progresso.

A análise também destacou que o uso eficaz dessas ferramentas ainda é uma competência em desenvolvimento. Engenharia de Prompt, domínio das ferramentas e intervenção reflexiva desempenham papéis críticos na determinação dos resultados. Simplesmente habilitar a IA no IDE não é suficiente; as organizações devem investir em como suas equipes aprendem, experimentam e se adaptam.

A oportunidade é real. A responsabilidade também.

A assistência de código com IA pode ser um acelerador poderoso quando usada nos contextos certos. Ela brilha especialmente em prototipagem, exploração e criação de estrutura (scaffolding). Esses são cenários onde a velocidade e a demonstração de resultados importam mais do que a qualidade do código produzido. Criticamente, nessas situações, o tempo de vida do software escrito geralmente é bastante curto (horas ou dias).

No entanto, para que as equipes se beneficiem dessas ferramentas em ambientes empresariais, elas precisam ser aplicadas com cuidado, com uma abordagem de “humano no loop”, fornecendo supervisão e governança do código gerado para garantir que o design e a arquitetura geral permaneçam sob controle dos desenvolvedores e não sejam entregues às ferramentas.

As equipes que tratarem a IA não como um atalho, mas como uma colaboradora – e que investirem em capacitação, criarem diretrizes de uso e reforçarem valores fundamentais de engenharia – serão aquelas que alcançarão seu pleno potencial sem comprometer a qualidade no longo prazo. Não devemos perder de vista as práticas técnicas fundamentais que demonstraram melhorar a qualidade do software, nem devemos ignorar as oportunidades que as ferramentas de assistência de código com IA apresentam.

Na Codurance, estamos empolgados com o potencial das ferramentas de codificação com IA e adoraríamos trabalhar com você para garantir que sua adoção seja um sucesso. Fale conosco para avaliar sua prontidão para trazer ferramentas de codificação com IA para suas equipes de desenvolvimento.

codurance

hello@codurance.com
www.codurance.com

A Codurance é uma consultoria global de software que ajuda empresas a construir uma capacidade técnica melhor e sustentável, que apoia o crescimento e a inovação.

Nós construímos software bem elaborado, confiável, seguro e fácil de modificar, que minimiza desperdícios e reduz custos e prazos de entrega.

Para mais informações, visite: **www.codurance.com**

Siga-nos nas redes sociais:
@codurance