Blogue Fragmentos do Caos

A verdade nasce onde o pensamento é livre.

O programador que sabe código e o programador que sabe pensar

Publicado em 2026-11-15 11:28:00



FC-CHRONIC-NEWS · CRÓNICA · TECNOLOGIA · CONHECIMENTO

programador que sabe pensar

Porque a excelência em software começa muito antes da primeira linha de código

Francisco Gonçalves · FC-Chronic-News · 11 de Outubro de 2026

CONTEXTO

O *Future of Jobs Report 2025*, do World Economic Forum, oferece um dado revelador: entre as competências nucleares indicadas pelos empregadores, o pensamento analítico surge com 69%, o pensamento criativo com 57% e o pensamento sistémico com 42%. A programação aparece com 17%. A leitura não é que programar deixou de ser importante; é precisamente a inversa: num mundo saturado de ferramentas, cresce o valor de quem sabe pensar, enquadrar, decidir e transformar conhecimento em solução. [3]

APIs ou plataformas tecnológicas como quem colecciona peças numa montra. Tudo isso é necessário em determinados momentos. Nada disso, por si só, cria um grande programador.

A linguagem é uma ferramenta. O compilador é uma ferramenta. A *framework* é uma ferramenta. A cloud, o sistema operativo, a base de dados e agora a inteligência artificial são ferramentas. A profissão começa num ponto anterior a todas elas: **compreender um problema suficientemente bem para imaginar uma solução que ainda não existe.**

> O programador comum aprende a utilizar ferramentas. O grande programador aprende a ver problemas. O extraordinário consegue ver soluções onde os outros ainda só conseguem ver o problema.

Há uma ideia persistente, sobretudo quando se observa a profissão de fora, de que o melhor programador é aquele que domina mais linguagens. Como se conhecer C++, Java, Python, Rust, JavaScript, Go e meia dúzia de *frameworks* produzisse automaticamente capacidade de engenharia. Não produz. Pode produzir velocidade de execução, fluência sintáctica e até código tecnicamente elegante. Mas um programa pode estar impecavelmente escrito e resolver magnificamente o problema errado.

Frederick Brooks explicou, ainda em 1987, que a dificuldade essencial do software não desaparece com uma nova ferramenta ou uma nova linguagem. A complexidade fundamental está na própria construção conceptual do sistema: perceber o domínio, representar as relações certas, distinguir o essencial do acessório e tomar decisões que sobrevivam ao contacto com a realidade. [2]

É exactamente por isso que a primeira competência de um bom programador não é escrever uma função. É formular perguntas. O que pretende realmente o cliente? Qual é a causa e qual é apenas o sintoma? Que restrições existem? Onde estão os estados impossíveis? Que dependências podem falhar? Qual é o custo de

Antes do algoritmo vem o mundo

Um programador não trabalha verdadeiramente sobre código. Trabalha sobre representações do mundo. Quando cria software hospitalar, aproxima-se da medicina, da fisiologia, da privacidade e da ética. Quando trabalha em cibersegurança, precisa de compreender redes, criptografia, mas também comportamento humano, engenharia social e incentivos. Num sistema financeiro, economia, risco e estatística deixam de ser conhecimentos periféricos. Na indústria, entram física, automação, sensores, tolerâncias e processos reais.

Até a experiência do utilizador exige mais do que desenho de ecrãs: exige psicologia cognitiva, percepção, atenção, memória de trabalho e compreensão do erro humano. E quando entramos na inteligência artificial, as fronteiras tornam-se ainda mais porosas: matemática, estatística, linguística, epistemologia, neurociência, filosofia da mente e ética começam a conversar com o software.

algorítmico — uma atitude intelectual que ultrapassa largamente a escrita de programas. [1]

> Quanto mais áreas um programador consegue compreender, maior é o espaço de soluções que consegue imaginar.

A universidade dá fundamentos; a maturidade não vem num diploma

Uma boa universidade pode ensinar matemática, algoritmos, estruturas de dados, arquitectura, sistemas operativos, redes, bases de dados, engenharia de software e método científico. Tudo isto é precioso. O erro está em confundir a conclusão de um currículo com a formação de um solucionador de problemas maduro.

ninguém documentou; limitações orçamentais; requisitos contraditórios; falhas em produção; escolhas que parecem óptimas hoje e desastrosas dois anos depois. Cinco anos de experiência podem produzir um profissional excelente, naturalmente, mas o calendário, sozinho, não fabrica profundidade. **Experiência não é tempo decorrido; é problemas enfrentados, erros compreendidos e modelos mentais corrigidos.**

Com os anos forma-se aquilo a que chamamos intuição técnica. O engenheiro experiente olha para uma arquitectura e sente que algo não está certo antes de conseguir demonstrar formalmente onde está a falha. Não é misticismo: é reconhecimento de padrões comprimido por milhares de decisões anteriores. A experiência transforma memória episódica em heurística.

visível

Durante décadas, escrever código era também o gargalo material da profissão. Era necessário traduzir cada ideia para sintaxe, linha após linha. A inteligência artificial começou a reduzir esse custo. Já consegue gerar funções, testes, documentação, consultas SQL, interfaces, transformações de dados e estruturas inteiras de aplicações em segundos.

A consequência não é que o programador deixe de ser necessário. É que **o valor se desloca da produção mecânica de código para a qualidade do raciocínio que o antecede e para a capacidade de validar o que foi produzido.** A investigação recente em engenharia de software descreve precisamente esta transição: mais desenho, coordenação, avaliação, segurança e responsabilidade humana num ambiente de desenvolvimento aumentado por IA. [4]

Um estudo publicado na *Science* em 2026, analisando mais de 30 milhões de *commits*, estimou que a IA já escrevia cerca de 29% das funções Python nos Estados Unidos. Mais interessante ainda: os ganhos de produtividade apareceram sobretudo entre

Isto é quase uma demonstração experimental daquilo que durante anos foi apenas intuição profissional. Uma ferramenta poderosa amplifica quem já possui modelos mentais sólidos. Quem sabe arquitectura, conhece o domínio e detecta incoerências utiliza a IA como multiplicador. Quem não compreende profundamente o problema pode apenas passar a produzir erros — agora a uma velocidade extraordinária.

BOX DE FACTOS

- 69% dos empregadores inquiridos pelo World Economic Forum consideravam o pensamento analítico uma competência nuclear em 2025; programação: 17%. [3]
- A literatura recente da ACM sobre engenharia de software na era da IA sublinha que a complexidade conceptual e humana continua central. [4]
- Um estudo publicado na *Science* em 2026 encontrou ganhos associados ao uso de IA sobretudo entre programadores mais experientes. [5]
- Estudos de produtividade mostram ganhos elevados em tarefas como documentação e

O programador do futuro será cada vez menos apenas programador

É aqui que a profissão se torna extraordinariamente interessante. O programador do futuro terá de ser, cada vez mais, arquitecto de sistemas, investigador, analista do domínio, crítico de modelos, avaliador de riscos, mediador entre pessoas e máquinas e, em muitos projectos, uma espécie de tradutor entre diferentes campos do saber.

Não precisa de ser médico para construir software médico, nem neurocientista para desenvolver sistemas de interacção inteligente. Precisa, porém, de ter curiosidade suficiente para entrar nesses territórios sem medo, aprender a sua linguagem, compreender os seus mecanismos essenciais e conversar competentemente com quem sabe mais do domínio.

Essa curiosidade transversal é talvez uma das competências mais difíceis de ensinar — porque nasce também de uma disposição perante o mundo. Há quem veja apenas o ticket no sistema de tarefas. Outros

> A IA poderá escrever as dez mil linhas de código.
> O engenheiro terá de descobrir quais são as dez mil linhas que vale a pena escrever.

No final, a excelência nunca esteve na quantidade de sintaxe armazenada na cabeça. Esteve na qualidade dos modelos mentais. Na capacidade de abstrair sem perder a realidade. De simplificar sem ser simplista. De combinar conhecimentos que aparentemente não tinham relação. De antecipar o que pode correr mal. E de construir uma solução que, depois de pronta, pareça tão natural que todos se perguntem porque não existia antes.

É isso que separa o codificador do engenheiro. E é também por isso que, paradoxalmente, quanto melhor a inteligência artificial aprender a programar, **mais importante se tornará saber pensar.**

profissional em engenharia de software. A afirmação de que a excelência "não se ensina numa universidade" deve ser entendida como uma distinção entre fundamentos transmissíveis por formação académica e julgamento profissional construído através de experiência, exposição interdisciplinar e resolução continuada de problemas. Não desvaloriza a formação superior, cuja importância para matemática, ciência da computação e método permanece central.

Do mesmo modo, referências a anos de experiência não constituem um limiar rígido. A qualidade e diversidade dos problemas enfrentados, a capacidade de aprendizagem, a mentoria e a autonomia intelectual podem acelerar ou atrasar fortemente o desenvolvimento profissional. A evidência recente sobre IA sugere que ferramentas generativas podem aumentar produtividade, mas os resultados variam com tarefa, contexto e experiência. [4] [5] [6]

Referências e leitura internacional

1. **Jeannette M. Wing — "Computational Thinking", Communications of the ACM, 2006**
   Texto seminal sobre abstracção, decomposição e pensamento computacional como competência intelectual para além da programação.
2. **Frederick P. Brooks Jr. — "No Silver Bullet: Essence and Accidents of Software Engineering", IEEE Computer, 1987**

Relatório internacional sobre competências: pensamento analítico, criatividade, literacia tecnológica, curiosidade e pensamento sistémico.

4. **ACM Transactions on Software Engineering and Methodology — "Software Engineering by and for Humans in an AI Era", 2025**
   Análise sobre produtividade, experiência do programador e centralidade do factor humano na engenharia de software aumentada por IA.

5. **Science — "Who is using AI to code? Global diffusion and impact of generative AI", 2026**
   Estudo de grande escala sobre adopção de IA na programação e diferenças de impacto entre profissionais experientes e iniciantes.

6. **McKinsey Digital — "Unleashing developer productivity with generative AI", 2023**
   Estudo experimental sobre ganhos de produtividade com IA generativa em documentação, geração, refactorização e tarefas de maior complexidade.

7. **Harvard Business Review — "AI Tools Make Coders More Important, Not Less", 2025**
   Reflexão sobre o valor crescente de programadores capazes de dirigir e avaliar ferramentas automáticas de produção de código.

8. **Business Horizons — "Navigating software development in the ChatGPT and GitHub Copilot era", 2024**
   Revisão sobre a transformação das funções, competências e práticas de desenvolvimento na era dos modelos generativos.

Coautoria editorial e enquadramento documental:
Augustus Veritas — AI Assistant

FC-Chronic-News · canonical · 11-10-2026

☁️ GitHub Pages

🛰️ CodeBerg Pages

🌌 **Fragmentos do Caos:** Blogue • Ebooks • Carrossel

👁️ Esta página foi visitada ... vezes.

Contactos