Hello everyone! Let's talk about AI. I understand this topic is too noisy. And frankly, it's a vast one. So, let's narrow it down and discuss the real help AI provides to software developers. And of course, we'll talk about when we'll all be replaced by machines.

I've heard polarized opinions, ranging from admiration for AI's omnipotence to disparaging ridicule of its stupidity. Let's look at the facts.

AI helps programmers in the following ways: it generates code from prompts, explains what is happening in the program, gives advice, traces simple chains of variable assignments, and searches for something in the code. Quite a lot!

Big things are seen from a distance. Let's jump back a quarter of a century and look at the tools my colleagues and I had at our disposal. Specifically, in my experience, I used DOS (not Microsoft) and later MS-DOS until Windows 95 was introduced. The text editor for programs was little different from today's Linux console. At best, it was something like Norton Commander with pseudo-graphics and an abundance of hotkeys. Of course, there was no code highlighting or command prompting whatsoever. Debugging tools were rudimentary, essentially printing variable values to the screen.

I remember my admiration for the Borland Delphi development environment (~1999-2000; are you aware of the Y2K problem?). It was my first visual editor that suggested command completions and (oh my god!) could generate the visual interface code sections. In a few clicks (the mouse already worked!), I could create a desktop application with buttons and drop-down lists and launch it. Isn't that like today's AI?

I remember my confusion: if a machine can create a stub application on its own, perhaps it's not worth getting involved in IT, because soon it will be able to write the rest itself?

And now, a quarter of a century later, we're back at the same point. Machines are great at generating code, but they're not capable of fully creating software. Why is that?

If we're discussing software to support businesses, from a bakery to a ship's engine, its purpose is to simplify and reduce the cost of operating the business. For example, to replace the boatswain yelling "full speed ahead" with the trumpet and the ten sailors pulling the cranks of the ship's engine. Microcontrollers are now doing this, yelling (calling) each other. They also contain software. Or to simplify the accounting of a bakery: money, flour, and jam. If people's market value weren't so high in the post-industrial era, businesses wouldn't need as much IT.

At the core of a business are its business processes. And the more complex the business, the more complex the processes. If you thought of a coffee shop as an example of a simple business, well, no, you could write a thick book about how it works. It's the same with software. A simple interface with a few steps doesn't always mean there's something trivial underneath.

Software developers negotiate with business owners (or their representatives, such as analysts), seeking to determine which processes can be automated and what measurable benefits this will yield for the business. Often, business owners themselves can't precisely articulate these processes. Furthermore, any organization has politics, budgets, conflicts of interest, and external factors that aren't included in the model (as a reminder, a smart machine is based on a statistical model). Let’s not forget about data access permissions within comprehensive security systems, legacy applications, and other technical issues.

How is an AI supposed to figure this out, negotiate with everyone, and create a comprehensive solution? Huh? It appears easier to build a business from scratch using an AI-native approach.

Another aspect is the software developer's understanding of what they're doing. I've attached a simple JS file with various array sorting algorithms.

A trivial task? Not exactly. Solution quality depends not only on the data size but also on how the data is distributed within the array; speaking more broadly, what is the data? The execution result shows how much the number of iterations and the time spent differ. You might ask, why do we need an array when we have powerful databases?

In databases, sorting and searching don't happen magically; they follow similar algorithms that take optimization into account. It's not for nothing that we have, for instance, clustered indexes and binary indexes (and other tools). I believe this topic deserves a separate post — one that explains it simply.

It's obvious in the JS file that the AI doesn't see the style issue. The code for shuffling arrays was copied rather than put into a function. A human wouldn't have missed it.

Finally, the economic aspect. Please share in comments the names of profitable companies whose business is exclusively related to AI. Its development requires an insane amount of resources. What will this mean for us, the end users? Will they twist our arms with subscription prices in the long run, because we are enterprise users and can pay much more than ordinary users? Or, worse yet, will the costs be hidden in the price of the traditional cloud services we use anyway? I recall the case in which Yandex (a tech giant) switched from Oracle to PostgreSQL in 2016 because "Oracle licenses cost as much as the Golden Bridge."

Even as a tech lead, I face a hypothetical choice: hire a supremely skilled programmer with an orchestra of AI agents or a regular development team. On the one hand, it's cheaper to pay one person, but what if they leave the company and no one else can manage the orchestra? Or what if a task arises that can't be solved through agents? For example, integration with a large and complex system. It seems to me that in “marathon”, a team looks more reliable.

In sum, I have a calm, non-reverent attitude toward AI technology as it applies to our work. It's a useful tool that I use alongside other, time-tested ones. Today, AI is certainly not the Holy Grail and doesn't do our work for us.

What do you think?