AI Capabilities and Human Consequences

Musk, Gates, and what one engineer's AI-built projects suggest about the future of human work

AL BAEZA · SEPTEMBER 2026 · SUMMARIZES VERSION 5 OF THE FULL PAPER

Musk has argued that AI will soon surpass human software development capabilities; Gates has argued that society must prepare for AI to displace large numbers of workers. Musk is describing a capability, Gates its consequences — the same event seen from two sides. Displacement will likely start in software, because software has properties — digital format, explicit rules, automatic testing — that make it unusually easy to automate; it will not stop there.

My Personal Evidence

Over 18 months, alone, with AI assistance, after a 40-year digital design career, I built five projects — all public and MIT licensed. The variety matters: a compiler, a desktop application, a code generator, a hardware design and a mobile app say far more together than a result holding only in my specialty. What they establish is still narrow — that the phenomenon occurs and how it works, not how widely it generalizes. The full paper states the limitations in detail.

PROJECT	WHAT IT IS	CORRECTNESS DECIDED BY
abCore16	16-bit CPU, compiler, toolchain	Simulation, bitfile on FPGA, logic analyzer
abDraw	Schematic capture tool	Daily use
abVisualRTL	FSM → SystemVerilog/VHDL generator	Testbench: 20,117 cycles, zero errors
abUART	UART + testbench	Bitfile on FPGA, verified with logic analyzer
B_Bot app	Android BLE app, new stack	Live hardware link

WHERE THE TECHNOLOGY IS FAMILIAR

AI does the building — coding, debugging, documentation, iteration. Existing experts become far more productive. Evidence: the four engineering projects.

WHERE THE TECHNOLOGY IS UNFAMILIAR

AI teaches the unfamiliar stack while work proceeds. The population able to build widens. Evidence: the Android app — 14 days to a working prototype in Kotlin and BLE.

The Limits of Text-Based AI

The AI-generated SystemVerilog for abCore16 was functionally correct and wrong in its timing — readable, passing inspection, misbehaving on hardware. Caught with the AMD Xilinx logic analyzer, which made the timing behavior visible on the running device. The rule: where correctness is decided by running code, expect large gains; where it's decided by physics — propagation delay, thermal behavior, clinical outcome — expect AI to get most of the way, and expect to still need an instrument and someone who can read it.

How the tools came to exist

abCore16 started as curiosity — could I build a microprocessor from scratch and run it on an FPGA board? Preserving its hand-drawn diagrams needed a tool nothing on the market fully met, so I built abDraw, adding features as the drawings demanded them. abVisualRTL followed the same pattern. The general case: an engineer who once had to file a request with a software department, or with an external tool vendor, and wait months or years, can now say *I'll build it.*

The consequences

- **Multiplication beats replacement as the threshold.** AI doesn't need to surpass an expert — only absorb routine implementation — for staffing to change. A hundred-engineer firm needs only twenty AI-augmented engineers to match its old output.
- **Software abundance is a real counterweight.** Cheap software makes previously irrational tools rational — abDraw has exactly one user and was worth building. If demand grows faster than labor-per-unit falls, employment could hold.
- **Adoption lag ends as a step, not a slope.** Years of firms reporting developers are "somewhat more productive," then a competitor demonstrates the full economics and the question becomes why staffing hasn't changed.
- **The entry level disappears first.** A field can grow in aggregate while losing the rung where judgment is learned. Retraining people into "architect" roles doesn't scale: too few roles, years of the experience the entry level used to provide, and a boundary that keeps moving while people are mid-retraining.
- **Wages have distributed wealth for two centuries.** People contribute labor, labor is necessary, and necessity confers bargaining power. Weaken the necessity and the mechanism weakens with it — and a tax code that already lets automation be written off while payroll is taxed makes this worse, not better.

> AI has already demonstrated the ability to dramatically reduce the human labor required to create sophisticated software — and this has the potential to fundamentally change how engineering organizations are structured.

Musk's timeline is probably wrong; the underlying trend is not. Whether AI can write software is settled — the useful question is what happens when one knowledgeable person with AI can accomplish what previously required an organization. AI is changing not merely how software is written, but who can create software.

The hopeful reading is the one worth steering toward: not fewer people making the same software, but far more people able to make the software they need — a retired engineer, a teacher with a classroom problem, a nurse who can see how a workflow should run — all able to build the tool instead of petitioning for it. Nothing about the technology decides which outcome we get. Policy choices still open decide it: how we tax automation versus labor, how we teach the next generation, how early we measure the disruption, and how we make sure the wealth AI creates reaches more than the people who own it.

STATEMENT ON AI USE

All software described here was built with generative AI — principally Google Gemini and Anthropic Claude, with OpenAI ChatGPT used less often — for architecture, code generation, debugging, testbenches and documentation, frequently developing with one model and reviewing with another. The author retained responsibility for architecture, requirements, engineering decisions, validation and acceptance. AI also assisted in preparing this document; a paper arguing that AI collapses the distance between an idea and its execution would forfeit its standing by obscuring its own methods.

Full paper, methodology, limitations and sourced artifacts: digitekxplorer.com · github.com/digitekxplorer