AI Capabilities and Human Consequences

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

Al Baeza · DigiTekXplorer · El Paso, Texas

ABSTRACT

Elon Musk predicts AI will surpass human software development capabilities within twelve to eighteen months. Bill Gates argues governments must prepare for AI to displace large numbers of workers. These are two halves of one question: how quickly AI can replace human cognitive labor, and what happens to a society if it does.

I take the first question seriously because of what I watched happen in my own projects. Over roughly 18 months, working alone with AI assistance after a forty-year career in digital design, I built a 16-bit microprocessor ecosystem with a C-like compiler and full toolchain, a schematic capture application now in daily use, a finite-state-machine editor that generates synthesizable SystemVerilog and VHDL, a UART carried to a bitfile on an FPGA board, and an Android Bluetooth Low Energy application in technologies I had never used. Work that would conventionally have taken years took months. All source is public. These projects are evidence, not subject matter: what Musk describes is already at work in every one of them.

Two years ago I would have told a college student that software was among the safest careers available. I no longer believe that. Experienced developers who can architect systems will remain necessary; the routine coding that has always been the profession's entry point is the part AI does best. Senior judgment retained, entry level removed — that pattern is not confined to software, and the industries behind it have no plan.

> AI can now build remarkable things with very few people. The central question is where the displaced people go, how fast they can get there, and who is planning for it.

1. Two Predictions, One Question

Musk's twelve-to-eighteen-month figure is the wrong thing to argue about; the number reads as emphasis rather than estimate. The durable claim beneath it is testable, and I believe it is true: *the human labor required to turn an engineering idea into working software is falling sharply, now.*

Gates is asking the harder question. He has spent a career arguing that technology makes everyone more productive, and now asks what happens when productivity stops requiring many people. If Musk is even approximately right, Gates' concern becomes more urgent — yet the two are almost never discussed together, which is how a society ends up well informed about a capability and unprepared for its consequences.

The question I want answered is narrower than either: **how quickly can displaced people retrain, is it even possible to retrain large numbers at once, and retrained to do what?** The last part gets skipped. Every reassuring account of past automation waves needs all three, usually assumed rather than demonstrated.

Position and disclosure. I am a retired electrical engineer with more than forty years in digital design, beginning in embedded systems and ending as an FPGA developer. The projects in Sections 3 and 4 were built after retirement — no employer, budget, team or deadline — with substantial AI assistance, principally Google Gemini and Anthropic Claude, with OpenAI ChatGPT used less often. I worked across models deliberately: main development in one, a second asked to review its output. Disagreement between two models is a cheap and surprisingly effective signal that something needs a human look. This paper itself began in ChatGPT before moving to Claude. A paper about AI displacing human work would forfeit its standing by concealing its own methods.

Why this is happening now. No single breakthrough produced it. Three things arrived together: accelerated computation, vast amounts of digital data, and new AI algorithms. Computation got fast enough — graphics processors and the hardware built after them turned training runs that would have taken decades into runs that take weeks. The digital world we had been building for fifty years supplied the data — text, code, images, records, transactions — in quantities no earlier generation of researchers could have used, and it keeps accumulating faster than anyone can read it. And the algorithms caught up, with the transformer and what followed it converting that computation and that data into systems that handle language, code and images well enough to do real work. Any one of the three alone would have produced nothing much. Together they produced agents that can work on our data directly.

2. Why Software Goes First

Software is being automated early not because programmers are less skilled, but because software has a rare combination of properties that make it well suited to machine work.

The output is a digital format — no material, no shipping, no plant.	Compilation and execution give immediate feedback.
The rules are explicit and formally specified.	The AI can generate, inspect, modify and test its own output.
The work can be tested automatically.	Work decomposes into relatively independent tasks.
Enormous quantities of training data exist.	No physical body is required to do the job.

The first property is the broadest, and it is not only about output: the inputs, the design and the test results can all be represented digitally. Any system for which that is true is a candidate for automation, whatever the field. The list is a diagnostic for every other occupation — the more of these properties a job has, the sooner it is exposed. Accounting, legal research, technical writing, financial analysis, customer service and much of marketing and administration share most of them. Trades and care work share few, until robotics supplies the missing body.

There is a larger version of this observation. Humans are analog beings, and we have spent fifty years building an almost entirely digital world around ourselves — medical records, financial systems, engineering data, legal filings, logistics — nearly everything that matters now exists primarily as digital data. That was rational when the only agents in the system were people using digital tools. It looks different now that the agents can themselves be digital. A software agent operating on medical records or market data is native to that environment in a way we are not: machine speed, no fatigue, the whole corpus at once. We digitized the world's information for ourselves to work on, and we are now the slower of the two kinds of worker who can reach it.

Software is not a special case to reason about on its own. It is the leading indicator, and a leading indicator can be measured before the rest arrives.

3. My Personal Evidence

I have no industry statistics. I have five completed projects, built alone with AI, which establish something narrower and harder to dismiss: an experienced engineer working with AI can produce, in months, systems that conventionally required a team and years. Full documentation lives in the repositories and at digitekxplorer.com; what matters here is the shape of the result.

PROJECT	WHAT IT IS	HOW CORRECTNESS WAS DECIDED
abCore16	16-bit CPU ecosystem: C-like compiler, assembler, disassembler, simulator, SystemVerilog core	Toolchain self-consistency; simulation
abDraw	Schematic capture with pinned nets, auto-routing, buses, multi-sheet netlisting	Daily use on real drawings
abVisualRTL	Visual FSM editor generating synthesizable SystemVerilog and VHDL	20,117 simulated cycles against a reference model, zero errors
abUART	SystemVerilog UART with generated testbench, carried to hardware	Bitfile running on an Arty S7-50
B_Bot app	Android BLE control application in Kotlin — a language and stack I had never used	Live communication with hardware

The variety is the point. A result holding only in Python, or only in my specialty, would be far weaker than one holding across a compiler, a desktop application, a code generator, a hardware design and a mobile app. Artifacts are public and MIT licensed: abCore16_dev, FPGA_dev, abDraw, abVisualRTL.

Two distinct things are happening in that table, and mixing them up is why most public discussion of AI reaches the wrong conclusions.

WHERE THE ENGINEER KNOWS WHAT TO BUILD

AI does the building — coding, debugging, documentation, iteration.

Effect: experienced engineers become far more productive.
Evidence: the four engineering projects.

WHERE THE ENGINEER DOES NOT KNOW THE TECHNOLOGY

AI teaches it while the work proceeds, so a project needing months of study to begin can start immediately.

Effect: the population able to build widens.
Evidence: the Android application — fourteen days to a working prototype in Kotlin and BLE.

The first predicts fewer developers producing the same software. The second predicts more people producing more software. Both are happening at once, which is why the employment picture is hard to forecast — and why confident predictions in either direction should be distrusted, including mine.

One practice is worth reporting because it cost nothing and caught real errors: using a second model to review the first one's work. A model has no independent access to whether its own output is right, and it is least reliable exactly where it is most confident. A different model, given the same specification and asked to critique rather than produce, has no stake in the first answer. Agreement

is not proof — both can be wrong in the same way — but disagreement is a cheap, high-yield signal that a human should look. Where generation has become nearly free, the scarce resource is anything that tells you where to aim attention.

One thing did not come from the model in any of the five projects: the judgment. No AI proposed abDraw; the need arose from friction only its user could feel. The architectural commitments — instruction set, net model, oversampling scheme — were decided by me. Recognizing that an output was wrong, including when the model was confident it was right, was human. So was deciding something was finished and could be relied upon. That is a professional act, not a technical one, and it is the part I would advise a young person to build a career on.

> What these five projects establish is narrow but not small: AI has already demonstrated the ability to dramatically reduce the human labor required to create sophisticated software, and that capability can fundamentally change the structure of engineering organizations.

4. How the Tools Came to Exist

None of these projects began as a demonstration of AI. Each began as a need, and the sequence is the most transferable thing in this paper.

abCore16 started as curiosity: could I build a microprocessor from scratch and get it running on an FPGA board? An 8-bit design quickly morphed into a more capable 16-bit one — simple enough to understand completely, complete enough to be real. I had hand-drawn diagrams I wanted to preserve properly, so I went looking for a drawing package. My requirements were not exotic: free, easy to use, electrical schematics as well as block diagrams and mind maps, multiple sheet types in one document, and the features a working engineer expects from a commercial tool. Nothing satisfied all of them. So I built abDraw, and I built it while using it — capturing the abCore16 diagrams surfaced the missing features, and each was added as the drawing demanded it. The result meets my requirements exactly, because its only user wrote it.

abVisualRTL arrived the same way: I was transcribing state diagrams into HDL by hand, mechanical work that goes stale the moment the diagram changes, so the diagram became the source instead. Each tool answered friction in the previous one. Nobody specified this program of work; it accumulated.

The general case is not about drawing packages. Consider an engineer at a large aerospace company who says: *I need a tool that analyzes this class of structural problem.* Historically the answer was “we don't have one,” “submit a request,” or “ask the vendor” — then requirements,

approval, staffing, implementation, testing, deployment. Six months later, perhaps, the tool arrives, if it survived prioritization. The engineer can now say instead: *I'll build it.* A small sentence and a profound change.

THE BOUNDARY THAT IS COLLAPSING

Historically	Domain expert → software department or tool vendor → tool
Emerging	Domain expert + AI → tool

Electrical engineer → engineering tool	Financial analyst → modeling tool
Mechanical engineer → simulation tool	Scientist → research pipeline
Architect → design tool	Clinician → data-analysis tool

Often the gatekeeper was not even an internal software department. Much of my career involved requesting features — and sometimes fixes — from tool vendors. An FPGA team that needed something from its toolchain filed a request with Xilinx and waited; if enough other customers wanted the same thing it might appear in a release months or years later, and if not, never.

The person who understands the problem is increasingly the person who can build the solution. That boundary was never a law of nature; it was a consequence of implementation being expensive enough to require a department.

The economics follow. If a specialized tool conventionally costs on the order of half a million dollars to develop, it needs a large user population to justify the investment, and many useful tools never clear that bar. Suppose AI reduces the effective cost to a few tens of thousands — the figures are illustrative, and mine were spent in evenings rather than dollars, but the ratio matches what I observed. A tool serving twenty engineers now makes sense. That inverts the shape of the software universe: away from one product serving millions, toward millions of specialized tools serving small groups or single individuals. abDraw has exactly one user, and it was worth building.

5. The Limits of Text-Based AI

Any paper of this kind should report a failure. The AI-generated SystemVerilog for the abCore16 core was functionally correct and wrong in its timing — readable, well structured, passing inspection, misbehaving on hardware. Correcting it required the AMD Xilinx logic analyzer to make the timing behavior visible on the running device, followed by manual repair. Post-place-and-route simulation reaches the same finding faster; an embedded logic analyzer is what design engineers reach for often, because it reports what the fabricated device actually did rather than what the timing model predicted.

The reason is structural. AI performs best where the artifact and the truth about it are the same kind of thing: a Python function is text, and its correctness is decided by running the text. In hardware description the artifact is text and the truth is physics. So the useful question for any occupation is: *what decides whether your output is correct?* If the answer is running the code, expect large gains. If it is a physical measurement — propagation delay, thermal behavior, mechanical tolerance, clinical outcome — expect AI to get you most of the way, and expect to need an instrument and someone who can read it.

I should be careful about how permanent I claim this boundary is. A logic analyzer trace is itself digital — numbers representing timing and logic levels — and so are the timing reports, constraint files and place-and-route results around it. Nothing about that data is beyond a model; what is missing today is integration and training, not representability. When AI is built into vendor toolchains such as Xilinx's and trained on real timing data, I expect it to assist with timing much as it now assists with code. The boundary I hit belongs to the current tooling, not to what machines can reason over.

Note what the boundary is not. The AI handled a compiler, an assembler and a simulator, then stumbled on when a signal arrives. Difficulty for a model is not proportional to difficulty for a person, so intuitions about which jobs are "too hard to automate" are unreliable. Careers should not be planned on them.

6. Fewer People, Same Output

Debate about AI and jobs usually turns on whether AI can "really" do the work. That sets the bar in the wrong place. *Replacement* requires AI to surpass expert human performance across a whole job. *Multiplication* requires only that routine implementation be absorbed, so one professional with AI produces what five produced before. Both reach the same employment outcome, and the second has a far lower threshold — it is already observable.

A company with a hundred engineers does not need superhuman AI to change its staffing; it needs twenty AI-augmented engineers to match its old output. The decisive question is never whether the machine equals the person, but how much less human labor a given output requires.

The counterweight is real, and Section 4 is where it comes from. AI can reduce the people required per unit of software while increasing the software produced, because it makes previously irrational software rational. If demand grows faster than labor-per-unit falls, employment could hold. That is a genuine optimistic case — but it depends on an output expansion large enough to absorb a sharp fall in labor per unit, an empirical question nobody can currently answer.

The change arrives suddenly, not gradually. Capability and adoption are different things, and most firms sit in the gap — held there by existing staff, management structures, legacy systems, procurement, regulation and inertia. Hence a risk few people account for: the employment effect likely arrives as a *step* rather than a slope. For several years companies report developers are somewhat more productive. Then a few demonstrate the full economics, competitive pressure makes the comparison unavoidable, and the question changes from "how do we use AI?" to "why are we staffed for the old curve?" Gradual capability, sudden reorganization. Planning for the slope and receiving the step is how institutions — and workers — get caught.

7. The Entry-Level Problem

Young people ask me what is safe to study now. Two years ago I would have said software without hesitation. My answer has changed, and the reason is *which* jobs go, not how many.

The work that grows is specification, architecture, verification and judgment — senior work. The work that shrinks fastest is routine implementation — the traditional entry point, and the way every senior engineer I know acquired judgment. A field can expand in aggregate while its entry level disappears. That is worse than a shrinking field, because it stays invisible for about a decade and then presents as a shortage of exactly the judgment-heavy people the field depends on.

> If routine implementation was the training ground and it is going away, the path to senior judgment has to be rebuilt deliberately. No market mechanism will do this, and the cost of neglect arrives on a decade's delay.

What I now tell students: the durable skills are the ones AI cannot supply for you. Deciding what is worth building. Specifying it precisely enough that an implementation can be judged against it. Knowing your field's ground truth and being able to read the instrument that reports it. Recognizing a confident wrong answer. Accepting professional responsibility for a result. Depth in a domain other than programming is worth more now than fluency in a language, because the domain tells you whether the output is right.

And the honest part: I cannot name a category of cognitive work I am confident will be untouched five years from now. Advising a nineteen-year-old to pick a safe field assumes I can see a stable landscape, and I cannot. The best I can offer is that adaptability, judgment and the ability to learn a new stack in weeks — which AI itself now makes possible — are more defensible than any particular skill.

8. Can People Retrain Fast Enough?

Every optimistic account of automation rests on one assumption: displaced workers move into new work. It has often been true. It has never been tested against this combination of conditions, and each attacks the assumption differently.

Speed Retraining takes years; a step-function reorganization takes quarters. Previous transitions gave a generation. This one may give a hiring cycle.

Breadth Earlier waves automated one category and left others open. AI reaches routine cognitive work broadly, and robotics extends the logic to physical work. Retraining requires somewhere to go.

Direction The surviving roles are more skilled, not less. Displacing people upward asks them to acquire judgment, the slowest thing to teach and impossible to compress into a short course.

Scale Retraining a few thousand workers is a program. Retraining millions at once, into fields that are themselves being automated, is not a program anyone has run.

Age A displaced fifty-five-year-old accountant faces different arithmetic than a twenty-five-year-old, and most retraining policy is written as though everyone is twenty-five.

And retrained into what? The honest answer in my own field is system architecture and verification — the judgment work AI does not supply. But a profession cannot be composed entirely of architects. The role exists because there is implementation to direct, and it has always been a small fraction of any engineering organization. Telling a displaced population to become architects is arithmetic that does not close: too few positions, years of the experience the entry level used to provide, and a boundary of what still needs a human that keeps moving inward while people are mid-retraining. Aiming a retraining program at a receding target is a failure mode nobody has planned for.

I do not conclude that retraining is impossible. I conclude that it is being assumed rather than planned, and that public discussion rests almost entirely on that assumption. If it fails, it fails after the displacement — the worst possible order in which to find out.

9. Who Owns the Machines

For two centuries, wealth in industrial economies has been distributed primarily through wages: people contribute labor, labor is necessary, and necessity confers bargaining power. Weaken the necessity and the mechanism weakens with it. The question shifts from *who has a job?* to *who owns the productive systems?* — models, compute, semiconductor manufacturing, robotics, energy, data, intellectual property and capital. All capital-intensive and concentrated by nature.

> A society can experience the greatest productivity expansion in its history and one of its most severe employment disruptions at the same time. These are not contradictory. They are the same event described from two sides.

The incentives already lean that way, and not by design. An employer hiring a person pays payroll taxes on those earnings; a company investing in automation can generally write the cost off as a business expense, often immediately. Gates is right that this is a bias built into the tax code favoring machines over people — rules written when machines helped workers rather than replaced them. Worth naming because it is one of the few levers already in place and adjustable: displacement is not just about what the technology can do — it is about incentives we chose, and can choose differently.

The danger is not that people become poor. It is enormous total wealth whose distribution has come loose from employment, inside institutions with no other distribution mechanism at scale. That is where Gates' concern lands, and where his proposed instrument is weakest: legally reserving certain occupations for humans is hard to administer and harder to justify — why the customer service representative and not the software engineer, the accountant and not the teacher? The likely result is preserving *jobs* rather than *people*, and those are different objectives.

I do not propose a solution, and I distrust papers of this kind that do. My contribution is narrower: evidence from the ground that the mechanism is real, already operating, and available to one retired engineer in El Paso with a laptop and a development board.

10. Limitations

1. **Five projects by one engineer are examples, not a survey.** They establish that this happens and show how it works. They say nothing about how common it is across engineers, fields or organizations.
2. **Durations come from memory, not records,** and every conventional-effort comparison is an estimate made by an interested party about a project never attempted the other way. What actually happened is fact; what would have happened is my judgment. I report figures rather than ratios for this reason.
3. **Conditions were unusually favorable:** no employer, budget, deadline, stakeholders, legacy code, compliance regime or team. Much of this paper concerns organizations, and organizations have all of these.
4. **Forty years of expertise cannot be separated from these results.** It shaped every one of them. The Android case is the only evidence the effect extends past deep expertise, and even there judgment came from an adjacent field.
5. **The evidence is dated.** Observations were made in 2025 and early 2026 with the models then available. A paper arguing that change is rapid must accept that it describes a moment.
6. **Sections 6 through 9 are argument, not evidence.** They reason outward from a handful of examples to organizations and economies. I am an engineer, not an economist; readers should hold Sections 3 and 5 to a much higher standard than what follows. I do too.

11. What Should Be Done Now

Governments	Prepare for a change that arrives suddenly rather than gradually, and protect people rather than occupations. Re-examine a tax code that taxes what companies pay their workers while letting them write off the machines that replace them (Section 9). Measurement is the cheapest useful action available now: current instruments will not detect a transition that appears as reduced hiring rather than rising unemployment.
Universities	Examine engineering and programming courses for relevance in an AI-enabled discipline. The graduate who can specify, direct and verify is worth more than the one who implements quickly. Teach verification as a first-class subject. And teach students — in high school, not only university — how to use AI to learn rather than how to use AI to do their homework. Those are different skills, and only one of them survives graduation.
The professions	Address the junior problem deliberately. Apprenticeship has to be paid for by someone once it is no longer a byproduct of routine work.
Companies	Pilot genuinely AI-native teams rather than distributing AI tools across the existing structure; only the first reveals what the new operating model costs. Measure outcomes, not activity. Fund verification apparatus deliberately — when generation is cheap, the constraint moves to checking.
Individuals	Learn where the models fail in your own field, as Section 5 did for hardware timing. That knowledge is now part of the craft, and it is not in any training course.

12. Conclusion

Musk's timeline is probably wrong, and arguing about it has consumed attention the underlying trend deserves. Whether AI can write software is settled. The useful question is what happens when one knowledgeable person working with AI can accomplish what previously required an organization — and my answer, from five projects and eighteen months, is that a great deal happens very quickly and almost none of it is being planned for.

The uncomfortable half has to be said plainly, because it is the half left out of product announcements. The same mechanism that let one retired engineer replace a team is the mechanism a company uses to decide it needs twenty engineers instead of a hundred, and the profession's entry point is the first thing it takes. Musk describes the speed of that change; Gates describes its consequences. What nobody has supplied is a credible account of where the displaced people go and how fast they can get there. That account needs to be written before it is needed, not after.

I would not want the warning mistaken for pessimism about the technology, because my own experience has been the opposite. I built things after retirement that I could not have built in a career, including in technologies I never learned, and I enjoyed it enormously. AI is a genuinely

creative instrument in the hands of someone who knows what they want. A retired engineer, a teacher with a classroom problem, a nurse who can see how a workflow should run, a graduate student with an instrument and no budget — all can now build the tool instead of petitioning for it. The long tail of software that was never worth building for a small audience is a real gain, and it is arriving for people who never had a software department to ask.

The proposition I would leave with a reader: AI is changing not merely how software is written, but who can create software. That is the outcome worth steering toward — not fewer people making the same software, but far more people able to make the software they need. Nothing about the technology decides which we get. That is a matter of what we choose to tax, teach, measure and pay for — ordinary decisions, made in time, by people who understood what was coming.

> If one person can now accomplish what once required an organization, the question is not how few people we will need. It is how many people we are prepared to hand that capability to.

STATEMENT ON AI USE

The author used generative AI extensively throughout all software described here — principally Google Gemini and Anthropic Claude, with OpenAI ChatGPT used less often — for architectural discussion, code generation, debugging, testbench generation, documentation and iterative refinement. Major portions of a project were frequently developed with one model and reviewed with another. The author retained responsibility for architecture, requirements, engineering decisions, validation, testing and final acceptance. AI was also used in preparing this manuscript as an editorial and structural collaborator, beginning in ChatGPT and continuing in Claude; the observations, case-study material, the failure reported in Section 5 and the judgments throughout are the author's own.

This disclosure is placed prominently rather than in a footnote. A paper arguing that AI collapses the distance between an idea and its execution would forfeit its standing by obscuring its own methods.

OUTSTANDING BEFORE PUBLICATION

Sourced citations for the Musk and Gates statements, and for the tax treatment of wages versus equipment discussed in Section 9 · development durations per project · the specific timing defects in Section 5 and which model produced that HDL · a short bibliography connecting these observations to published work on developer productivity, adoption lag, retraining outcomes and labor economics. Versions 1–4 retain the fuller case-study detail on the five projects.