

AI is not on a PhD student level.

FROM CLAIMS
TO CAPABILITIES

SPECIFY
VERIFY
TEST
MEASURE
UNDERSTAND
BUILD FOR REALITY

**THE BEST
SAFETY SYSTEM
IN THE WORLD**
Warning: it may occasionally fail.

SAFER
SMARTER
TOMORROW

**REAL
CONDITIONS
AHEAD**



OPERATING
ENVELOPE

UNCERTAINTY

EDGE CASES

FAILURE MODES

REALITY

Mateusz Chrzanowski

SYSTEM OF SYSTEMS ARCHITECT



ANALYTICAL EDITORIAL

AI is not on a PhD student level.

SYSTEMS / AI / SAFETY / REALITY

01

OPERATING BOUNDARY

Five seconds

I wanted one animated element on a website. A frame attached to a line was supposed to move across the screen, increase in size during the movement and finish as an active section of the page. The entire sequence would take roughly five seconds. Before I started, I asked the obvious question: can this actually be built with the tools available to me?

The answer was yes.

That answer mattered because I was not asking whether such an animation was theoretically possible. Of course it was. I was deciding whether this particular workflow could deliver it reliably enough to justify spending my time on it.

It could not.

We tried one approach, then another. The technology changed. The method changed. At some point the process had moved into 3D and then into Blender, which I had never intended to use for the website at all. Scripts were produced, I ran them, errors came back, another version followed. Eventually something moved, but the result was nowhere near what I wanted. After roughly two hours I removed the animation completely.

The failure itself is not interesting. Every tool has limits, every implementation can fail and some ideas turn out to be disproportionately expensive compared with the value they provide. The important part is that I asked about the limit before I began and did not receive a useful description of it. If the real answer was that the function was theoretically achievable but this particular environment could not reliably deliver it, that was the information I needed.

I did not need encouragement.

I needed to know the operating boundary of the tool.

5 s

TARGET INTERACTION

Result: function removed from scope.

~2 h

IMPLEMENTATION ATTEMPT



Fourteen hours for a homepage

I continued with a static version. After roughly fourteen hours of work across two days and around forty production deployments, the desktop homepage became reasonably acceptable. Not finished. Part of the background still required correction, the neural-line structure still did not look the way I wanted and the mobile version needed substantial additional work. But the main desktop composition had finally reached a point at which I could continue.

Some of the most revealing failures were not particularly sophisticated. At one stage, text had to sit inside vector frames whose geometry was already known. There was no subjective question about whether the composition looked balanced. The text either remained inside the permitted frame or it did not.

It did not.

The actual error became obvious once I looked at the relationship rather than another individual parameter. Text on one side of the composition was positioned relative to the wrong anchor. The same positioning logic had been reused for mirrored geometry. Once I explicitly described that relationship, the problem could be corrected.

This is important because all of the information required to verify the result already existed. The frame had defined bounds. The text had a measurable bounding box. A deterministic rule could establish whether the text remained inside the permitted area with the required padding.

No interpretation was required.

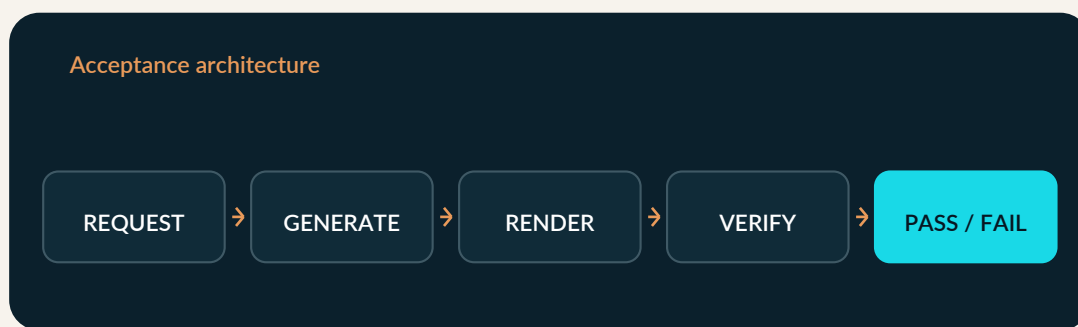
If the text leaves the frame, the result fails.

The same pattern appeared in much smaller changes. I wanted ten additional pixels of white space at the top of the page. The modification disturbed other elements, moved parts of the composition and clipped a logo. In another case I explicitly required one graphic to remain completely outside

the area occupied by another, received a corrected version and then rendered it only to find that the requirement was still violated.

Changing a parameter does not prove that the requested state was achieved. Producing another implementation is not the same operation as verifying that implementation. Once those two things are confused, a system can keep producing plausible versions while the user becomes the actual acceptance layer.

That is an engineering problem, not an intelligence problem.



One country

It would be easy to dismiss the website example as a consequence of visual work. So I have a completely different one.

I started building a small application for calculations used in my European energy architecture work. This is numerical work: defined inputs, defined rules, defined relationships and defined outputs.

After roughly eight hours of work, I had one country calculating correctly.

Then I moved to a larger country and the failures returned. One correction exposed another dependency. Another run produced another problem. Variables changed, rules were adjusted and the application failed again.

At that point I did not have a reliable system for calculating countries. I had a system that had been made to calculate one country correctly.

That distinction is precisely what disappears when AI productivity is discussed through broad percentages. "AI can do 80 percent of the work" sounds meaningful until someone asks the next question: 80 percent of what?

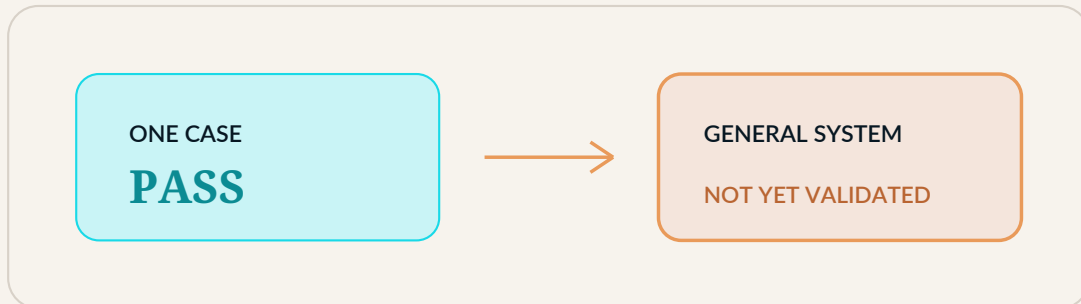
My own productivity has changed surprisingly little. My way of working has changed considerably.

I can continue some work away from the office. I can avoid learning software that I do not want to learn merely because one part of a project requires it. I can obtain visual material differently. I can approach programming differently. I can distribute execution across several tools instead of doing every operation manually.

Those are real advantages.

They do not mean that my output suddenly multiplied. In one task AI saves a large amount of time. In another, the saved execution time reappears as iteration, debugging and verification. In another, the main benefit is mobility rather than speed.

There is no single AI productivity multiplier because there is no single AI task.



We already automated most of professional production

The current discussion often treats the automation of professional work as something historically unprecedented.

It is not.

Architectural design is a very good example. The difference between classical drafting and modern CAD/BIM environments is much closer to the difference between digging with a shovel and digging with an excavator than between using two different pencils.

Both methods can produce the result.

The amount of manual work required is radically different.

A structural designer once had to know the material parameters, calculate loads, determine the necessary wall thickness, calculate reinforcement, check the load path, recalculate the effect of changes and carry every consequence through the documentation. Calculators helped with arithmetic, but the process still required repeated manual calculation.

Modern design environments automate a huge part of this work. The designer defines geometry, loading conditions, structural constraints and requirements. Software performs analysis, sizes elements, calculates reinforcement requirements, propagates changes and produces documentation. Large libraries of existing components and standard solutions are immediately available.

A designer can impose a structural condition — for example, that a large part of a floor must span without an intermediate supporting wall — and then use the computational environment to determine what material, section, reinforcement or structural arrangement is required to make that geometry work.

Work that once required pages of equations and repeated recalculation can now be performed through a small number of design decisions and automated computation.

That is not a modest productivity improvement. It is a fundamental change in the amount of labour required to produce a building.

It has not changed who designed the building.

The architect remains the architect.

The engineer remains the engineer.

Software did not acquire authorship when it began performing the calculations.

I have seen the same change at industrial scale. A team of roughly eight designers can produce the engineering documentation for a power plant consisting of 23 buildings. Producing the same volume of coordinated documentation in a comparable timeframe using historical manual methods would have been practically unrealistic for such a small team.

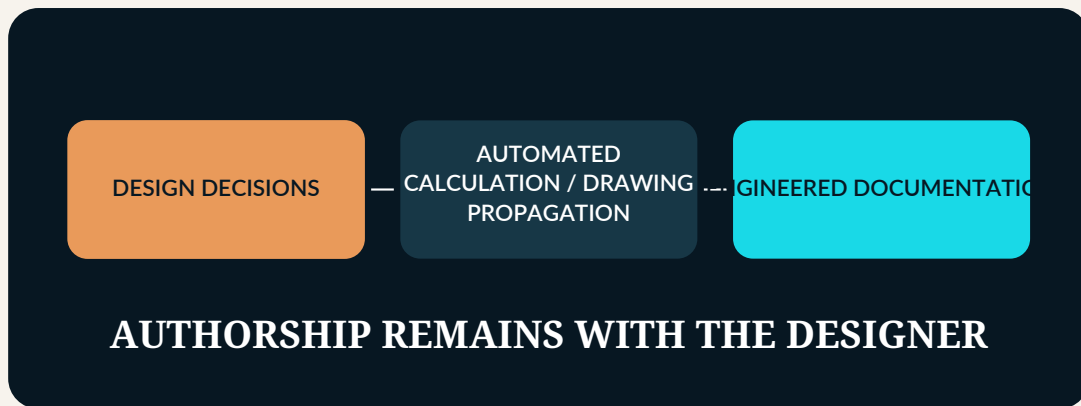
Nobody therefore concludes that a €5 million engineering project should cost €500,000 because software performed most of the repetitive production work.

That would confuse manual effort with professional value.

The designers still have to know what the plant is, how the systems interact, where the interfaces are, which constraints matter, how the disciplines affect one another and what a correct solution looks like. The software allows them to turn that knowledge into an engineered result at a speed that was once impossible.

Automation became part of their capability.

It did not become the author.



Who created the design?

The same distinction exists in visual design.

My tattoo artist can work with existing patterns and shapes, combine elements, alter them, transform geometry, change proportions, build layers and create an entirely new composition without drawing every primitive element from an empty canvas.

Who created the design?

She did.

The software may execute every transformation. It may control layers, anchors, scaling, masks, colour and geometry. Some elements may originate in existing libraries or references.

That does not remove the designer.

Design has never required creating every underlying component from raw material. An architect does not manufacture every brick. A graphic designer does not create every typeface. A product designer does not invent every screw.

The work lies in the selection, relationships, composition, constraints and decisions that determine what the whole becomes.

Using existing components does not eliminate authorship.

Automating the operations performed on those components does not eliminate it either.

I designed the installation

I have an even cleaner example from engineering.

I used to design installations manually, sometimes on paper and sometimes on a tablet. The drawing did not have to be visually elegant. It had to contain the information required to build the system: routes, sections, connection points, outlets, positions, parameters and the relationships between components.

By the time that drawing was finished, the technical solution existed.

My wife could operate AutoCAD because she is a surveyor by education and had learned CAD at university. She could take my drawing and produce clean digital documentation.

She could not have designed those installations herself.

She did not know industrial electrical design, automation, ventilation or air-conditioning systems. Give her an empty CAD file and the requirement to create the technical solution and she would not know what needed to be there.

Give her my drawing and she could turn it into DWG.

I designed the installation.

She digitised it.

AutoCAD was the tool.

I worked in much the same way on a power plant. If I needed a technical scheme quickly, I could draw exactly what was required on a tablet and send it to Jacek in the office. He could convert it into clean CAD documentation while I continued working elsewhere.

Nobody involved had difficulty understanding who had designed the solution.

Producing the final representation and creating the underlying solution were two different jobs.

That distinction did not suddenly disappear because software learned to generate more of the representation itself.



We gave programmers a calculator

Programming provides exactly the same problem in another domain.

AI can produce code extremely quickly. That is real automation. A developer can describe a required function and receive an implementation almost immediately.

But producing code is only one layer of software engineering.

Someone still has to design the software architecture. Someone must decide what components exist, what each one is responsible for, which component can depend on which other component, where state is held, what data belongs in which layer, how interfaces are defined and what happens when something fails.

A model can generate twenty functions in the time a developer once needed to write one.

That does not answer whether those twenty functions should exist.

There is a second problem that experienced developers increasingly point out. Generated code can work and still be unnecessarily complicated. A model may produce twenty or thirty lines containing several conditions, temporary values, states or dependencies. An experienced developer looks at the same requirement and sees that the function can be expressed cleanly in three or four.

The line count itself is not the issue. Three unreadable lines are not automatically better than twenty clear ones. The issue is unnecessary complexity. More states, more branches and more dependencies create more things that can fail, more things that have to be debugged and more code that must eventually be understood and maintained.

The generated function may therefore work perfectly and still be poor software engineering.

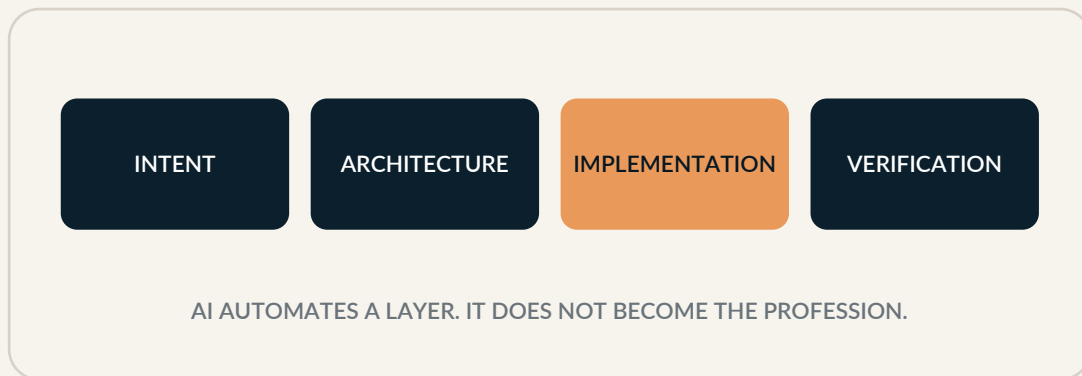
Someone has to know the difference.

We gave accountants calculators.

We gave programmers AI.

The role of the tool is different, but the principle is not. The calculator automated arithmetic. AI automates code production. Neither operation contains the entire profession around it.

A programmer still has to know what should be built.



The editor did not write the book

The same distinction is obvious outside engineering.

Give a Stephen King manuscript to an excellent editor. The editor can perform substantial work. Chapters can be moved. Weak passages can be identified. Repetition can disappear. Continuity errors can be caught. Sentences can improve. The finished book can be materially better because the editor was involved.

The editor still did not write Stephen King's book.

We have always understood that contribution and authorship are not synonyms.

Professional work is distributed across layers. Architects use engineers, draftsmen and specialist designers. Writers use editors. Directors work with cinematographers, actors, editors and entire production organisations. Designers use libraries, references and software.

We do not determine authorship by counting who produced the most lines, pixels or pages.

We look at where the intellectual work sits.

Who defined what was being created? Who determined the structure? Who selected between alternatives? Who rejected solutions that failed? Who understood what the final result was supposed to achieve? Who retained authority over the finished work?

A tool can perform an enormous amount of execution without becoming the author.

The software learned to speak

The real psychological break came when the interface became language.

A structural-analysis program can calculate reinforcement. It does not tell the engineer that it thinks the result is good.

AutoCAD can propagate changes through a model. It does not tell the architect that it understands the project.

A calculator can outperform a human at arithmetic. It does not apologise afterwards.

Conversational AI uses the same linguistic form we normally associate with another person. It says “I think”, “I understand”, “I recommend”, “I found” and “I made a mistake”. It explains itself. It maintains conversational context. It responds socially.

That changes how people perceive the system.

Once the tool begins to sound like another actor, the language around it changes as well. It becomes a “partner”. It “writes”. It “designs”. It “does the work”. Eventually it becomes “PhD-level”.

The interface changed.

The category did not.

A tool does not stop being a tool because its interface is conversation.

“PhD-level” is a category error

If a model performs comparably to doctoral researchers on a particular benchmark, say exactly that.

Name the benchmark. Define the task. Define the comparison group. Define the conditions.

That is useful information.

“PhD-level intelligence” is a different statement. It takes a result measured inside a defined evaluation and converts it into a human status label.

A PhD student is not a benchmark score.

A doctoral researcher is a person with education, memory, experience, a research environment, social context, long-term goals and responsibility for their own work. The person can decide that a question is worth pursuing, spend months investigating it, recognise that the original question was wrong and change direction.

A model does not become that person because one class of measured performance resembles the performance of a group of doctoral researchers.

The weakness of the label becomes obvious as soon as the task changes. The same system can participate in a sophisticated discussion about relativity and then repeatedly fail to position text inside a known vector frame. It can provide an excellent technical explanation and then fail to reproduce the structural character of a simple reference image. It can generate a functioning piece of software and still fill it with unnecessary complexity.

There is no contradiction.

The contradiction appears only if we insist on placing the entire capability of the system on one human scale between stupid and intelligent.

That scale does not describe the object.

The object has a performance profile.

I want that profile.



A benchmark is not authority

This distinction matters because the language used to describe a system changes the way people trust it.

If someone tells me that a system performed at 92 percent on a defined benchmark, I know what kind of statement I am hearing. I can ask what was measured, what the sample was, what the benchmark actually tests and whether those conditions resemble my task.

If someone tells me that the system is “PhD-level”, a different cognitive shortcut is activated. The benchmark has been converted into the status of a person I would normally regard as highly educated in a domain.

That status carries authority.

The problem becomes much larger when people use AI precisely because they cannot independently verify the answer.

If I use a system inside a domain I understand, I provide another validation layer. I can recognise when a structural assumption makes no sense, when the code architecture is bad, when the geometry is wrong or when the conclusion conflicts with physical reality.

If I ask because I do not know, that layer is gone.

Now the language used to frame the system becomes materially important.

“AI can make mistakes”

The familiar disclaimer is not:

Do not assign epistemic authority to this system.

It is:

AI can make mistakes.

Those are completely different statements.

Imagine a product advertised like this:



THE BEST SAFETY SYSTEM IN THE WORLD

Warning: it may occasionally fail.

Nobody responsible for a serious safety installation would consider the second sentence sufficient to neutralise the first.

The large statement establishes the class of system in the reader's mind. The smaller disclaimer merely says that this supposedly extraordinary system is not perfect.

That is exactly how the two messages work together.

If a system is described using a human expertise label and then accompanied by “AI can make mistakes”, the disclaimer does not erase the expertise label. It simply tells the user that the supposed expert is fallible.

Human experts are fallible too.

A PhD researcher can make mistakes.

A surgeon can make mistakes.

An architect can make mistakes.

So the disclaimer is entirely compatible with the authority created by the first statement.

That is why it does not solve the problem.



The ethical problem

There is an ethical distinction here that should not be hidden behind whether a marketing sentence is technically defensible.

You do not have to state a literal falsehood to mislead somebody.

Sometimes it is enough to take a narrow statement that can be defended under specific conditions and translate it into language that predictably creates a much broader interpretation.

A benchmark result can become “PhD-level”.

A measured capability can become “smarter than most people”.

A narrow comparison can become human status.

The relevant ethical question is not merely whether a lawyer could defend every word individually. The relevant question is what a reasonable user is expected to believe after seeing the communication.

If the user is shown a human expertise label because that language creates greater excitement, confidence and trust than the underlying technical specification would create, then a small disclaimer saying that AI can make mistakes does not restore the missing precision.

The marketing effect has already happened.

This matters more with AI than with many ordinary consumer products because the product is increasingly used to answer questions that the user cannot independently verify.

If a shoe advertisement exaggerates comfort, the customer eventually discovers that their feet hurt.

If an AI system produces a plausible but wrong answer in a domain the user does not understand, the user may have no independent signal that anything went wrong.

The product therefore operates in an environment where language can transfer epistemic authority directly to a system whose reliability varies dramatically by task.

That creates a higher standard for how capability should be communicated, not a lower one.

Technically defensible wording is not automatically honest communication.



Cheap inference can actually change the system

None of this means the reliability problem is unsolvable.

One of the most important developments is simply that inference is becoming cheaper. Once computation becomes sufficiently inexpensive, it becomes economically reasonable to use more of it to obtain one accepted result.

Instead of one model producing one answer, a system can use a planner, several specialist agents, an executor and a reviewer. Independent solutions can be compared. Failed paths can be retried. External tools can execute code. Deterministic validators can test conditions that should never have been left to probabilistic judgement in the first place.

That can produce a real step change in usefulness.

But notice what happened.

We built a better system.

If reliable performance requires a planner, several agents, external tools, an execution environment, deterministic validation and a human acceptance layer, the final capability belongs to that architecture.

It is not evidence that one language model has somehow become a doctoral researcher.

This distinction will become increasingly important as token prices fall. A multi-agent architecture that would once have been economically absurd may become cheap enough to use routinely. That can materially improve reliability and allow much more complex workflows to be automated.

Good.

That is engineering progress.

It still does not make “PhD-level” a useful specification.



Verification is a system function

The recurring failure in many AI workflows is not simply that a model occasionally produces a bad answer. Every serious technical system has failure modes.

The problem is that generation and verification are too often treated as if they were one capability.

They are not.

If text must remain inside a known boundary, check the geometry.

If a numerical result must fall within a tolerance, calculate the tolerance independently.

If code must compile, build it.

If a function must produce defined output for defined input, test it.

If a statement depends on current information, require a current source.

If a visual requirement cannot be inspected by the system, the system cannot truthfully declare that the visual requirement has passed.

Many of these checks do not require another model.

Where a deterministic test is possible, another probabilistic opinion is often the wrong architecture.

A correctly designed system must also have a valid FAIL state.

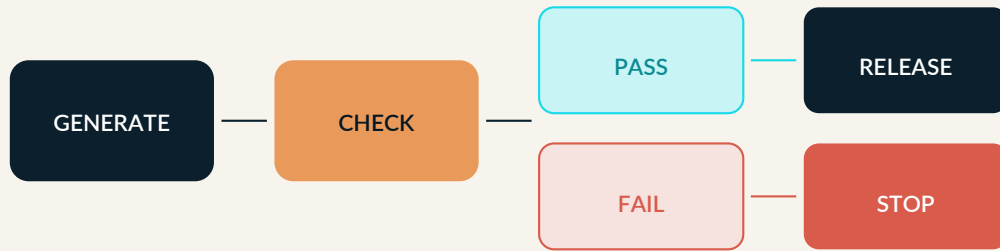
Sometimes the technically correct answer is:

I cannot verify this.

There is nothing unintelligent about that answer.

It is good engineering.

DETERMINISTIC WHERE POSSIBLE



Give me the specification

This is the information I actually want from an AI product.

Tell me what the system can do.

Tell me what it can verify.

Tell me the conditions under which it performs reliably and where the performance falls apart.

Tell me whether it can execute the code it produces.

Tell me whether it can inspect the rendered result.

Tell me whether a task requires external tools, several agents, deterministic validation or human acceptance.

Tell me which failure modes are known.

Tell me how much iteration a class of tasks normally requires.

Tell me where the operating envelope ends.

That information allows me to design a workflow around the system.

“PhD-level” does not.

The label gives me a human analogy precisely where I need engineering information.

Stop making it human

AI can be extraordinarily capable. It can automate enormous amounts of professional execution. It can calculate, generate, transform, search, code, draft and analyse at speeds that were impossible with earlier tools.

None of that makes it human.

We already automated calculation without turning software into an engineer.

We automated drawing without turning CAD into an architect.

We allowed editors to materially transform manuscripts without transferring authorship.

We gave accountants calculators without deciding that the calculator had become the accountant.

Now we have given programmers AI.

The principle has not changed.

The new tool happens to communicate in natural language, and that is enough to make people instinctively treat it as a social actor. It is not enough to make it one.

We do not need to make AI smaller to describe it accurately.

We need to stop making it human.

A hammer can outperform a hand. CAD can outperform a drawing board. AI can outperform a person in a defined task by an extraordinary margin.

None of those facts create authorship, intention or human status.

**Do not turn AutoCAD into
an architect.**

**Do not sell me a PhD
student. Give me the
specification.**

