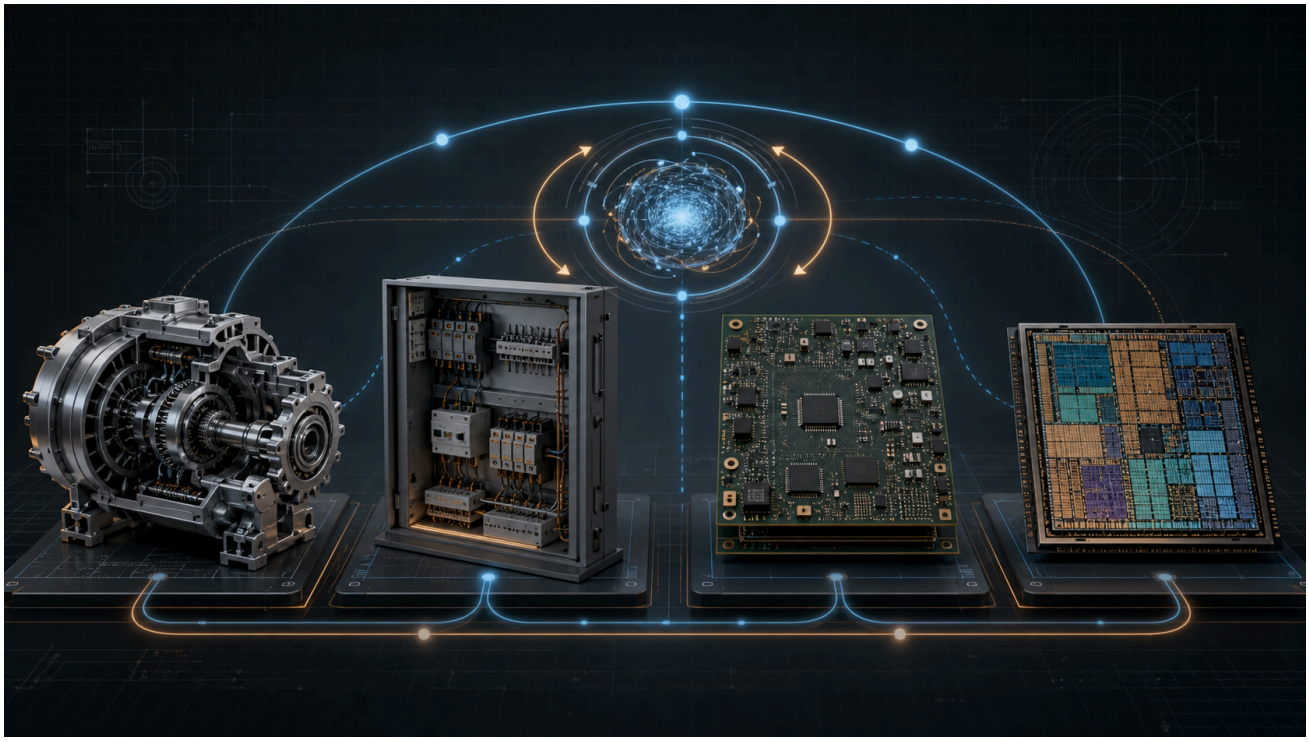
The Future of Engineering Design Is Agentic

Dr. Axel Richter
Engineering-Consult Dr. Richter, Aschaffenburg
Created: August 11, 2026 · Modified: August 11, 2026



Header graphic: Original AI-generated illustration.

The next CAD revolution is not a better text box. It begins when an AI system understands a design goal, independently selects the right tools, applies changes to the native model, verifies the result, and uses failures to determine its next step. This closed loop is what turns a copilot into a design agent.

The field is still in its earliest phase. Even so, it is already clear that it will fundamentally change computer-aided design: mechanical engineering, electrical schematics and cabinet planning, PCB development, and—with much higher barriers—chip design. The decisive change is not that engineers will need less knowledge. It is that knowledge, intent, and computation will be connected differently.

Today, people translate technical intent into a long sequence of menu commands, modeling operations, library searches, checks, and corrections. Agentic CAD moves that translation work into an executable loop. The human states requirements and boundaries; the agent plans and operates tools; CAD kernels, rule checks, and simulation produce dependable evidence; the human retains responsibility and release authority.

That may sound like a minor difference. In reality, it is a new operating model for engineering software.

Level 1: What actually makes CAD “agentic”

The term is currently applied rather generously. Anthropic offers a useful distinction: a *workflow* follows predefined code paths, while an *agent* dynamically directs its own process and tool use.[1] Applied to CAD, this gives us a clear ladder:

1. **Automation** executes a fixed sequence: run a macro, generate a bill of materials, export a drawing.
2. **Assistance** answers questions or suggests commands, but does not change the model—or does so only after one explicit, scoped request.
3. **Generation** creates geometry, code, or a draft from text or images, often in a single open-loop pass.
4. **Agentic CAD** pursues a goal over multiple steps, observes model state, calls CAD and verification tools, evaluates feedback, and revises its plan.

A generative model might turn “create a 100 × 60 mm mounting bracket with four M5 holes” into a solid. An agent also asks for ambiguous boundary conditions, creates parameters, builds a robust feature history, notices a missing bend radius, runs a collision or manufacturability check, corrects the design, and presents the final drawing package for approval.

The difference, then, is not primarily the quality of the language model. It is the loop of **perceiving, planning, acting, checking, and correcting**. Such a process needs standardized tool contracts, which MCP supports through tools and resources,[2] and layered guardrails rather than a single protective measure.[3] The local reference *The Hitchhiker’s Guide to Agentic AI* describes the agent harness as the runtime layer for state, tool calls, error handling, permissions, and observability.[4] In engineering, this layer is at least as important as the model itself.

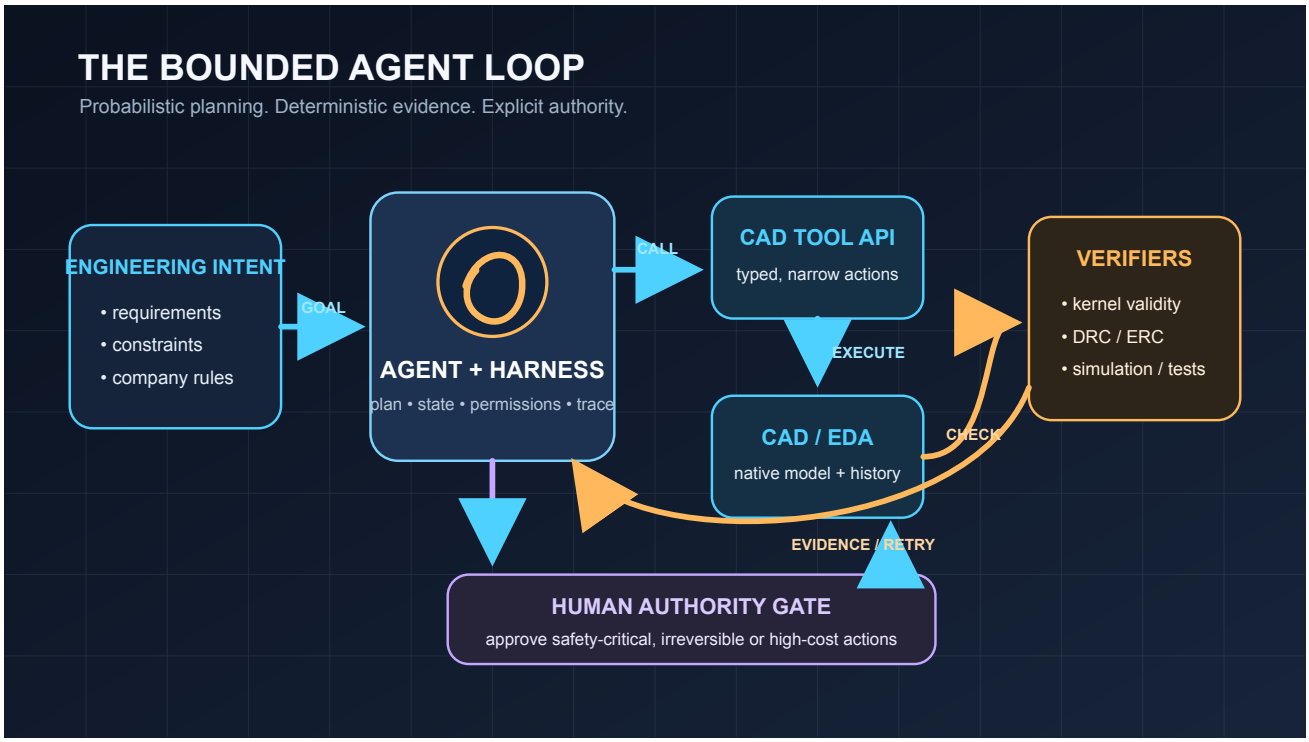


Figure 1: The bounded agent loop. Original diagram.

Formally, a CAD agent can be described as a stateful process. Given requirements, current model state, and observations, its policy selects the next action. The CAD system transforms that action into a new state; an independent verifier then evaluates the result:

$$a_t = \pi_\theta(g, s_t, o_t), \qquad s_{t+1} = \mathrm{CAD}(s_t, a_t), \qquad r_t = \mathrm{verify}(s_{t+1})$$

Formula graphic 1: State transition and verification in the agentic CAD loop.

The agent stops not when its answer sounds plausible, but when defined verification criteria pass, the budget is exhausted, or a human decision is required. This is the central mistake in many demos: attractive geometry or a compiling Verilog module is not yet a trustworthy engineering result.

What agentic CAD is not

Agentic CAD is not the same as generative design. Classical generative design searches a defined solution space under load, geometry, and manufacturing constraints, then returns alternatives. An agent can launch that optimization, compare results, alter constraints, and subsequently create drawings, documentation, or follow-on analyses. Optimization is one tool in the agent process; it is not the agent itself.

Nor is every chat panel an agent. An assistant that searches documentation is useful—especially inside a complex CAD system. It becomes agentic only when it performs authorized actions, reads their results, and derives further actions from them. Many current products sit precisely on this boundary.

Level 2: The shared architecture behind MCAD, ECAD, PCB, and IC

The four domains look very different. MCAD works with sketches, B-Rep geometry, features, and assemblies. Electrical design works with symbols, potentials, cables, terminals, and control cabinets. PCB design connects logical nets to footprints, copper, stackups, and fabrication rules. Chip design moves from specification and RTL through verification and synthesis to timing, power, and area closure.
The underlying agent architecture is nevertheless the same:

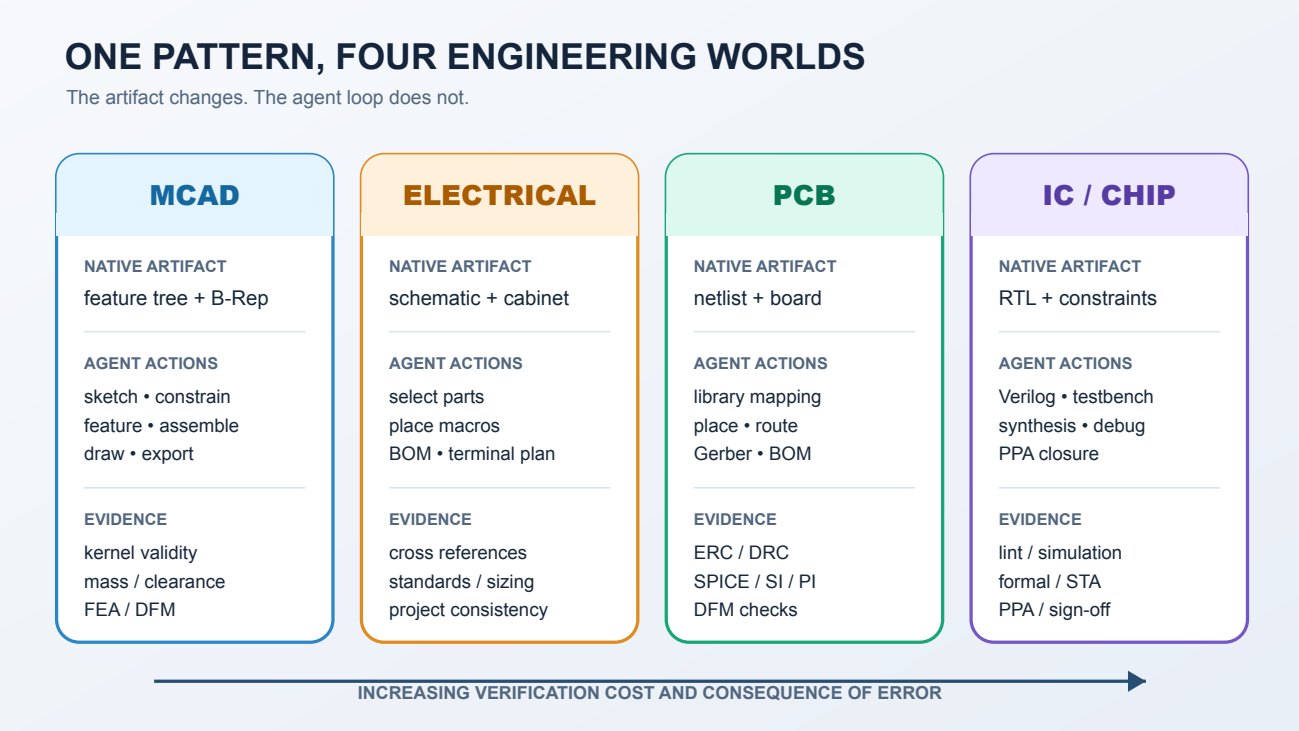


Figure 2: One pattern, four engineering worlds. Original diagram.

1. A native, editable artifact

A PNG of a component, a triangulated mesh, or a PDF of a schematic is not enough. Professional engineering needs the native object model: parametric features and B-Rep in MCAD; logically connected devices in an electrical schematic; nets and footprints on a PCB; RTL and constraints in the IC flow. Only then do design intent, editability, variants, and downstream processes remain intact.
This explains why "text-to-3D" does not automatically mean "text-to-CAD." A visually convincing surface can be topologically wrong, non-watertight, impossible to dimension, or unable to regenerate. Current research confirms the gap: Text2CAD-Bench contains 600 curated tasks across four levels of complexity and reports that current models perform reasonably on basic geometry but degrade substantially on complex topology and advanced features.[5]

2. A narrow, typed tool surface

An agent should not "click somehow." It needs tools with unambiguous names, input schemas, preconditions, and return values: `create_sketch`, `place_component`, `run_drc`, `compile_rtl`, `compare_mass_properties`. The Model Context Protocol standardizes this kind of connection between an AI application and tool servers through a client-server architecture with tools, resources, prompts, and notifications.[2]

MCP is not a quality certificate. A poorly designed or excessively powerful tool remains poor or dangerous even when invoked through a standard protocol. Its value is decoupling: CAD vendors and integrators can expose capabilities as versioned contracts while agent models remain replaceable.

3. A harness that turns model calls into a controlled process

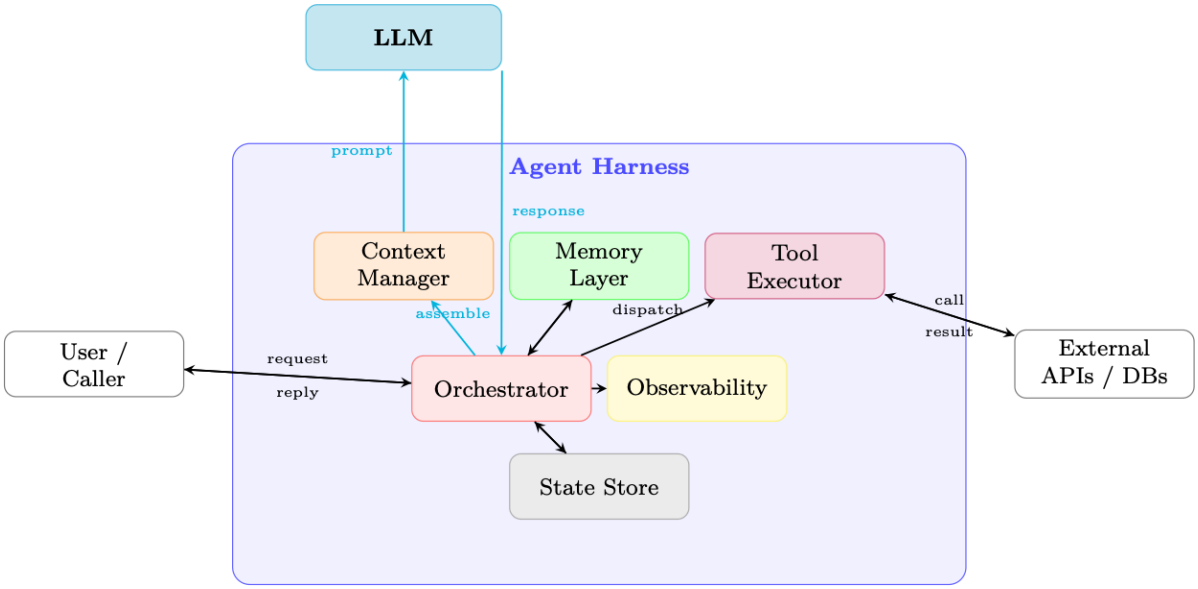


Figure 3: Agent harness architecture from The Hitchhiker's Guide to Agentic AI. *Source and context: [4].*

The harness retains working state, limits steps and cost, manages permissions, logs tool calls, and detects loops. It also decides what information the language model sees. In a real project, BOMs, datasheets, standards extracts, material tables, previous variants, and test reports can overflow the context window. A robust agent therefore does not blindly load "the entire project"; it constructs a goal-directed view for each step.

OpenAI recommends defense in depth for agent systems and notes that one guardrail is unlikely to be sufficient. [3] In CAD, that means combining schema validation of tool arguments, parameter limits, project-scoped access rights, transactional edits, undo and restore points, deterministic checks, and approval gates.

4. Verification outside the language model

The language model may propose actions and interpret error reports. It should not be the sole judge of whether its own design is correct. The decisive evidence comes from specialized systems:

MCAD	Feature rebuild, valid solid, dimensions	Collision, tolerance chain, FEA, DFM	Drawing, material, standards, review
Electrical	Open connections, device data	Cross-references, cable/terminal logic, sizing	Standards, customer rules, cabinet review
PCB	ERC, DRC, library consistency	SPICE, signal/power integrity, thermal	DFM/DFA, manufacturing files, review
IC	Lint, compile, unit testbench	Simulation, formal, CDC/RDC, synthesis, STA	Coverage, PPA, sign-off, tape-out gates

The right division of labor is therefore: **the language model plans probabilistically; the engineering stack verifies deterministically.**

Level 3: Building a dependable CAD agent

Use a state model, not a chat history

The most important technical step is an explicit state model. A conversation log is not project state. A CAD agent needs at least:

- an immutable requirements baseline with sources and versions,
- the current native document and revision state,
- a goal and constraint model,
- a list of planned, executed, and reverted actions,
- verification results with tool version, configuration, and timestamp,
- unresolved assumptions, risks, and approvals.

An action should be represented as a typed, auditable record, for example:

```
{
  "tool": "mcad.create_hole_pattern",
  "document_revision": "bracket@r17",
  "arguments": {
    "support_face": "face:mounting_plate/top",
    "count": 4,
    "diameter_mm": 5.5,
    "pitch_circle_mm": 72.0
  },
  "preconditions": ["support_face.planar", "diameter_mm > 0"],
  "approval": "auto_within_sandbox",
  "expected_checks": ["solid_valid", "min_edge_distance >= 2d"]
}
```

The semantic reference `face:mounting_plate/top` is intentionally more important than a transient internal face ID. In parametric CAD, topology and internal identifiers can change after a model edit—the familiar topological naming problem. An agent that stores only raw IDs may modify the wrong face on its next step. Robust integrations need stable semantics, geometric signatures, or controlled reselection.

Transactions, diffs, and reversibility

Every agent step should run as a transaction:

1. Read the state and document revision.
2. Validate preconditions.
3. Apply the change in a branch, copy, or undo scope.
4. Regenerate the model and run local checks.
5. Produce a structured diff: geometry, parameters, nets, components, or RTL.
6. Accept, repair, or fully roll back the change.

Undo is not merely a convenience; it is a safety primitive. Partially successful actions are particularly dangerous: three components were placed correctly, the fourth failed, yet the agent believes the whole operation was atomic. Tool results must therefore distinguish among `success`, `partial`, `failed`, and `unknown`, and identify affected objects exactly.

Let the verifier drive the loop

A practical objective combines hard constraints with optimization goals. For a mechanical design, one might define:

$$J(x) = w_m m(x) + w_c C(x) + w_t T(x) + \lambda_g P_{\text{geometry}}(x) + \lambda_d P_{\text{DFM}}(x)$$

Formula graphic 2: Example weighted objective function for a mechanical design.

Here, m represents mass, C estimated cost, and T a thermal metric. Geometry and manufacturing violations are represented by large lambda penalty weights. In practice, hard rules should not be modeled only as soft penalties: a self-intersection, short circuit, or negative timing slack must disqualify the candidate.
The agent should then receive structured evidence, not a vague sentence such as “that is wrong”:

```
check: min_edge_distance
status: failed
objects: hole[2], outer_edge[west]
measured_mm: 7.2
required_mm: 11.0
severity: blocking
suggested_scope: move_pattern_or_reduce_diameter
```

The more machine-readable the feedback, the less the language model has to guess.

Permissions must be domain-specific

Not every action has the same risk. A useful policy distinguishes:

- **Read:** inspect the model, properties, rules, and reports.
- **Draft:** create proposals, variants, or branches.
- **Modify:** change an active document within defined limits.
- **Release:** approve a revision, change PLM state, or create manufacturing deliverables.
- **Externalize:** transmit data to cloud services, suppliers, or manufacturers.

The last two classes should require human approval by default. The same applies to changes in safety-critical requirements, unqualified component substitutions, protection circuits, isolation distances, and sign-off constraints. The useful question is not “How autonomous is our agent?” but “Within what verifiable operating envelope may it act autonomously?”

Observability: every step must be reconstructable

For a production agent, teams must later be able to answer:

- Which model and prompt or skill version made the decision?
- Which documents and datasheets were in context?
- Which tools were called with which arguments?
- What changed in the native artifact?
- Which checks ran, with which version and configuration?
- Why was a warning accepted or escalated?

A complete trace therefore contains not just chat text, but model revisions, tool calls, CAD diffs, check artifacts, runtimes, cost, and approval identities. This is the foundation of debugging, auditability, and reproducibility.

Test it like software—plus engineering evidence

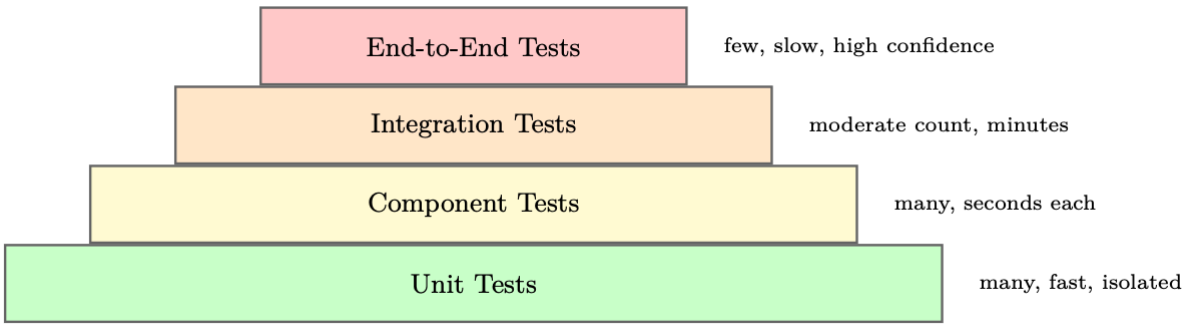


Figure 4: Testing pyramid from The Hitchhiker’s Guide to Agentic AI. *Source and context: [4].*

An agent needs unit tests for tool adapters, schema validators, and state transitions; component tests with simulated CAD responses; integration runs against real CAD versions; and a small number of expensive end-to-end tasks with expert acceptance. It also needs a domain evaluation set drawn from representative design work. The primary metric cannot be "a language model liked it." More useful measures include:

- **Task Success Rate:** share of tasks fully completed under deterministic criteria.
- **First-Pass Yield:** share accepted without human repair.
- **Constraint Violation Rate:** violations per task, weighted by severity.
- **Human Edit Distance:** amount of change required before release.
- **Tool Efficiency:** successful results per tool call, time, and compute budget.
- **Recovery Rate:** share of failure states repaired correctly.
- **Unsafe Action Rate:** unauthorized or impermissible action attempts.

A simple operational value metric can balance saved engineering time against rework and risk:

$$V = \Delta t_{\text{engineering}} - \alpha t_{\text{rework}} - \beta N_{\text{critical violations}} - \gamma C_{\text{compute}}$$

Formula graphic 3: Operational value after accounting for rework, risk, and compute cost.

An agent that produces many drafts quickly but overloads senior reviewers with rework has negative value—even if its demo is impressive.

Common failure modes

Visually plausible, technically wrong	Image feedback without native model understanding	Kernel, rule, and simulation checks as gates
Wrong object changed	Unstable IDs or ambiguous selection	Semantic references, preview, selection confirmation
Endless repair loop	No progress metric, vague error message	Iteration budget, state hash, structured verifier output
Hallucinated standard or datasheet	Ungrounded model knowledge	Approved sources, version/page citation, conflict detection
Local check passes, system fails	Verification context is too narrow	Hierarchical checks from feature to system
IP leaves the organization	Uncontrolled cloud or tool access	Data classification, local execution, egress policy, audit
Agent "optimizes away" the test	Poor reward definition	Immutable tests, separate verifiers, human gate

Current landscape: where the four domains stand in August 2026

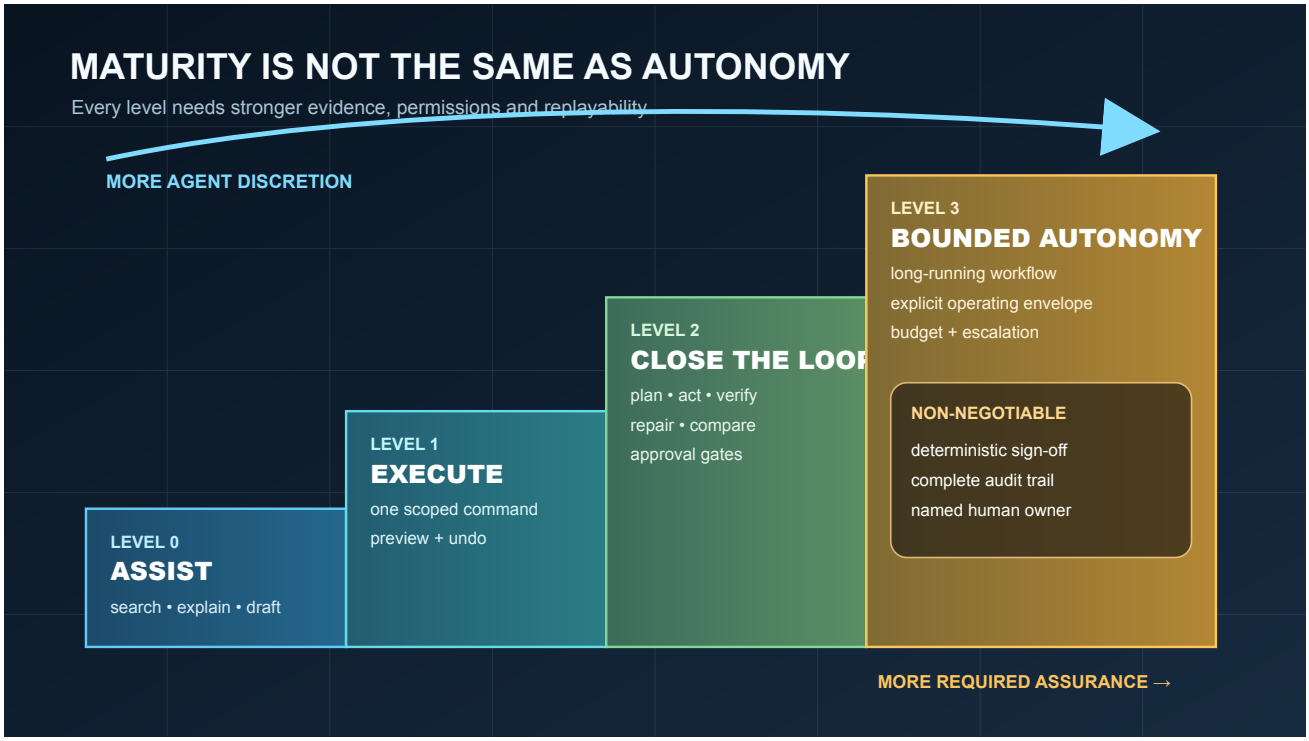


Figure 5: Maturity is not the same as autonomy. Original diagram.

MCAD: From help to native model action

In mechanical CAD, progress is visible but uneven. Autodesk is already moving Fusion toward native model action: Autodesk Assistant is described as a conversational experience that finds information, explains workflows, and completes tasks through natural-language prompts. Autodesk explicitly labels capabilities such as text-to-command and native geometry generation as development work or Tech Preview, however, and it has also described MCP-enabled Fusion workflows since 2026.[6] That qualification matters: a preview signals direction, not a guaranteed production feature.

Siemens introduced Design Copilot NX in 2025 as a natural-language learning and operating aid; its initial product description focused mainly on documentation-grounded guidance and launching commands mentioned in responses.[7] PTC brought Creo AI Assistant into the design environment with Creo 13 in June 2026.[8]

The open-source ecosystem makes the architecture especially visible. FreeCAD exposes an extensive C++ and Python API,[9] and several community projects expose subsets of it through MCP. One current example describes 32 tools for parametric modeling, inspection, CAM, and export, while also documenting platform limitations and the broad access granted through FreeCAD's Python environment.[10] Such projects are valuable laboratories for tool design, permissions, and evaluations—but they do not demonstrate that arbitrary industrial parts can already be designed autonomously.

The MCAD bottleneck is not creating a block with holes. The difficult problems are robust feature histories, stable references after edits, complex freeform geometry, assembly relationships, tolerances, material and manufacturing knowledge, and the transition from "looks right" to "ready for release." That is precisely why the field is still at the beginning.

Electrical CAD: WSCAD shows how quickly routine work can become agentic

WSCAD provides one of the most tangible examples in electrical design. The company introduced ELECTRIX AI in 2024 with an integrated AI Copilot. According to WSCAD, it can review designs, generate bills of materials, and place macros and components, among other tasks.[11] Release notes for ELECTRIX AI 2026 list concrete actions such as importing parts from wscaduniverse.com by command and automatically positioning inserted path macros in sequence.[12]

This goes beyond search assistance: natural language triggers structured actions inside the project. At the same time, advertised productivity figures—such as “up to 99 percent faster”—should be read as vendor claims, not as an independent cross-domain benchmark.[11]

Why is this domain so suitable? Many tasks are repetitive and strongly structured: selecting devices, inserting macros, maintaining cross-references, producing lists, translating text, and deriving terminal and cabinet data. Many rules are also machine-checkable. That creates a favorable agent environment: substantial routine work, a clear project state, abundant domain knowledge, and measurable outputs.

The hard boundary remains professional responsibility. Conductor sizing, protection concepts, selectivity, short-circuit withstand, thermal behavior, and standards compliance cannot rest on a linguistically convincing answer. A sound WSCAD or ECAD agent must load versioned company standards and project rules with traceable citations, call calculation tools, and escalate uncertainty.

Schematic and PCB design: KiCad as an open experimental platform

KiCad is particularly interesting for agentic PCB development because native file formats, command-line tools, rule checks, and a growing API come together. KiCad 10 was released on March 20, 2026 and added variants, PCB design blocks, and a graphical design-rule editor, among other capabilities.[13] Its official IPC API connects external processes to a running KiCad instance using Protocol Buffers and local IPC. The documentation also states clear limitations: in KiCad 9 and 10 it communicates only with the GUI; coverage is expanding incrementally; and clients must account for timeouts and synchronous processing.[14]

These sober interface details determine agent quality. An agent must recognize that KiCad is busy, avoid opening competing connections, and handle export or schematic manipulation differently depending on the version. “There is an MCP server” is no substitute for understanding the underlying CAD lifecycle.

Research also shows how quickly the closed loop is advancing. pcbGPT produces editable KiCad schematics from natural language, grounds component selection in libraries and datasheets, and combines structural and semantic validation. On 20 embedded-design tasks, the best model achieved strong first-pass results but degraded on difficult tasks; the authors explicitly state that expert review remains necessary.[15]

PCBWorld takes a similar step for routing: agents act through native KiCad operations and consume DRC feedback instead of merely producing coordinates or images. The benchmark environment includes synthetic generators and 679 real open-source boards, and scores completed designs using eight engine-checked metrics.[16] Methodologically, this matters more than a spectacular screenshot: the CAD/EDA engine becomes part of the evaluation.

For real teams, the best starting point is still not “generate the entire board.” Better bounded tasks include library checks, BOM enrichment, placement alternatives for non-critical regions, DRC triage, generation of manufacturing artifacts in a pipeline, or a review agent that prioritizes violations and attaches direct object links.

Chip design: Verilog is only the beginning

In chip design, the potential is larger—and the consequences of errors are much more severe. First, a naming correction often needed in these discussions: the EDA company is **Synopsys**, not “Synopsis.” **Mentor Graphics** has belonged to Siemens since 2017 and forms a major part of today’s Siemens EDA portfolio.[17]

LLMs can write Verilog, create testbenches, and explain tool scripts. Yet syntactically valid RTL says little about functional correctness, synthesizability, timing, area, power, or verification coverage. An AutoChip study coupled language models iteratively to compiler and simulation feedback. Tool feedback did not improve all evaluated models consistently; in the best case, successful designs increased by 5.8 percent, while a model-mixing strategy reduced cost.[18] The lesson is not “Verilog is solved,” but: **EDA feedback can help when model, loop, and cost strategy are evaluated together.**

At physical implementation, AlphaChip demonstrates another form of AI-assisted design. Google DeepMind models floorplanning as a sequential placement task using reinforcement learning and reports deployment on blocks across several TPU generations.[19] This is high-impact specialized optimization—not a general language agent autonomously developing a chip from specification to tape-out.

The clearest current move toward agentic EDA comes from established toolchains. Synopsys describes its copilots for knowledge access, RTL and formal testbench creation, and EDA workflows.[20] In July 2026, the company

announced two autonomous debug-closure and implementation-closure workflows with AMD and Microsoft for evaluation. Early vendor-reported results cite 25 to 40 percent shorter debug cycles; access is positioned as an evaluation, not as general proof of fully autonomous chip design.[21]

Siemens EDA—with its Mentor heritage—also positions Questa One around AI-assisted verification across creation, analysis, debug, and regression.[22] Agent architectures in this domain are likely to mature first in closure tasks: clustering failures, investigating root causes, proposing assertions or tests, varying tool parameters, and planning the next run. Tape-out approvals, safety targets, and sign-off remain deterministic, tightly controlled processes.

What will fundamentally change

The interface will move from command space to intent space

Menus will not disappear. Experts will continue to edit geometry directly, set constraints, and inspect reports. But the primary unit of interaction shifts from a command to a goal: “Reduce mass by eight percent without violating natural frequency, casting radii, or existing interfaces.” The agent translates that goal into a verifiable plan and shows what it intends to change and why.

CAD models will become executable knowledge objects

A good model will contain not just geometry or nets, but machine-readable intent, sources, assumptions, verification rules, and approvals. Design intent moves from informal expert knowledge into an operational layer. This sharply increases the value of good naming conventions, clean parameters, modular macros, maintained libraries, and versioned requirements.

Design and verification will converge

Today, “modeling” and “checking” are often separate phases. An agent needs feedback after nearly every meaningful step. Simulation, DRC, ERC, lint, formal verification, DFM, and cost models become tools in a continuous search loop. An organization that has not automated its verification pipeline cannot reliably automate its agent either.

Engineering organizations will operate agent platforms

An enterprise agent cannot consist only of a model subscription. It needs tool servers, approved knowledge sources, identity and access management, observability, evaluation data, sandbox infrastructure, and release processes. This creates a new interface among design engineering, CAE/EDA, PLM, IT security, and AI engineering.

Practical takeaways: Start without autonomy theater

1. **Choose a narrow, frequent task.** Good candidates include bills of materials, documentation, DRC triage, variants from known templates, parameter changes, and test-report summaries. Poor first candidates are safety-critical new designs with ambiguous requirements.
2. **Define success before choosing the model.** Assemble 30 to 100 representative tasks, hard constraints, permitted deviations, and review criteria. Without an evaluation set, you are optimizing for demos.
3. **Build read and verify tools first.** An agent that can reliably inspect and validate model state is already useful. Add write permissions later, initially only in branches or copies.
4. **Keep tools small and typed.** Prefer `set_parameter(name, value, unit)` and `run_drc(rule_set)` over a universal `execute_python(code)`. Powerful escape hatches multiply risk and make evaluations harder.
5. **Make every step visible and reversible.** Preview, diff, undo, checkpoint, and object links are not UI details; they are trust infrastructure.
6. **Separate generation from verification.** Use CAD kernels, solvers, rule checkers, and independent verifiers. A second LLM call is not a substitute for physics or formal methods.

7. **Measure rework, not just speed.** What matters is time to an approved result. Track first-pass yield, human edit distance, critical violations, and recovery rate.
8. **Increase autonomy only with evidence.** The maturity ladder runs from assistance to single actions, then to closed-loop work, and only then to bounded autonomy. Every step requires better tests, finer permissions, and stronger observability.

Conclusion

Agentic CAD will not transform engineering because a chatbot suddenly "designs." It will transform engineering because technical intent, software actions, and verification are joined into a continuous, machine-readable loop.

MCAD now shows the transition from help to native model actions. WSCAD demonstrates how rapidly structured electrical routine work can become executable through a copilot. KiCad's open formats, API, and rule checks reveal how agentic schematic and PCB flows can be built and evaluated. In chip design, Synopsys, Siemens EDA, and specialized approaches such as AlphaChip are moving toward longer, more autonomous optimization and closure loops.

But the standard remains engineering, not eloquence: Is the artifact native and editable? Are assumptions and sources visible? Does it pass deterministic checks? Is every change traceable and reversible? Does the agent know when to stop and ask a human?

Organizations that answer these questions early will not merely design faster. They will convert engineering knowledge into a new, executable form. That is the transformative potential—and that is exactly why we are only at the beginning.

References

[1] "Building effective agents." Anthropic. https://www.anthropic.com/engineering/building-effective-agents. Published/updated: 2024-12-19. Accessed: 2026-08-11. Used for: distinction between workflows and agents.

[2] "Architecture overview — Model Context Protocol." Model Context Protocol. https://modelcontextprotocol.io/docs/2026-07-28/learn/architecture. Published/updated: 2026-07-28. Accessed: 2026-08-11. Used for: MCP architecture, roles, and protocol primitives.

[3] "A practical guide to building agents." OpenAI. https://openai.com/business/guides-and-resources/a-practical-guide-to-building-ai-agents/. Published/updated: not listed. Accessed: 2026-08-11. Used for: guardrails and orchestration patterns.

[4] "The Hitchhiker's Guide to Agentic AI: From Foundations to Systems." Haggai Roitman. arXiv:2606.24937 [cs.AI]. https://arxiv.org/abs/2606.24937. Submitted: 2026-06-22; last revised: 2026-07-27 (v2). Accessed: 2026-08-11. Used for: agent harness, loop, observability, and testing-pyramid framing, plus Figures 3 and 4.

[5] "Text2CAD-Bench: A Benchmark for LLM-based Text-to-Parametric CAD Generation." Liang Wang et al. arXiv. https://arxiv.org/abs/2605.18430. Published/updated: 2026-05-18. Accessed: 2026-08-11. Used for: state and limits of complex text-to-CAD generation.

[6] "Autodesk Assistant: Shaping the Future of Design and Make." Autodesk Fusion Blog. https://www.autodesk.com/products/fusion-360/blog/autodesk-assistant-ai/. Published/updated: 2026-07 (updated; day not listed). Accessed: 2026-08-11. Used for: Autodesk Assistant, text-to-command, native geometry, and the MCP direction in Fusion.

[7] "Siemens brings AI copilot, immersive design and integrated fluid and thermal simulation to NX." Siemens. https://news.siemens.com/de-de/siemens-designcenter-nx-summer-2025/. Published/updated: 2025-07-01. Accessed: 2026-08-11. Used for: Design Copilot NX and its initial positioning.

[8] "PTC Brings AI-Powered Guidance to the Design Environment with Creo 13." PTC. https://www.ptc.com/en/news/2026/ptc-brings-ai-powered-guidance-to-the-design-environment-with-creo-13. Published/updated: 2026-06-10. Accessed: 2026-08-11. Used for: Creo AI Assistant and 2026 product status.

[9] "FreeCAD source documentation." FreeCAD. https://www.freecad.org/api/. Published/updated: not listed. Accessed: 2026-08-11. Used for: availability of C++ and Python programming interfaces.

[10] "freecad-mcp: MCP server for FreeCAD — 32 tools for AI-assisted 3D CAD modeling." blwfish. GitHub. https://github.com/blwfish/freecad-mcp. Published/updated: not listed. Accessed: 2026-08-11. Used for: current open-source example and its security and platform boundaries.

[11] "ELECTRIX AI – WSCAD Launches the First AI-Powered Electrical CAD." WSCAD GmbH. https://www.wscad.com/en/worlds-first-ai-powered-electrical-cad/. Published/updated: 2024-09-18; updated 2025-08-18. Accessed: 2026-08-11. Used for: launch, functionality, and vendor-reported productivity claims.

[12] "Release Notes | WSCAD ELECTRIX." WSCAD GmbH. https://www.wscad.com/en/electrix/release-notes/. Published/updated: continuously updated; ELECTRIX AI 2026 release 2026-02-12. Accessed: 2026-08-11. Used for: specific AI Copilot actions in ELECTRIX AI 2026.

[13] "Version 10.0.0 Released." The KiCad Development Team. https://www.kicad.org/blog/2026/03/Version-10.0.0-Released/. Published/updated: 2026-03-20. Accessed: 2026-08-11. Used for: KiCad 10 features and current product status.

[14] "For Add-on Developers — KiCad IPC API." KiCad Developer Documentation. https://dev-docs.kicad.org/en/apis-and-binding/ipc-api/for-addon-developers/index.html. Published/updated: 2026-04-04. Accessed: 2026-08-11. Used for: IPC API architecture, status, limitations, and runtime behavior.

[15] "pcbGPT: Automatic PCB Schematic Synthesis from Natural Language Requirements." Tobias King et al. arXiv. https://arxiv.org/abs/2606.01188. Published/updated: 2026-05-31. Accessed: 2026-08-11. Used for: grounded KiCad schematic synthesis, evaluation, and limitations.

[16] "PCBWorld: A Benchmark Environment for Engine-Grounded PCB Design Automation." Hyungseok Song et al. arXiv. https://arxiv.org/abs/2607.05915. Published/updated: 2026-08-03 (v2). Accessed: 2026-08-11. Used for: engine-grounded routing, DRC feedback, and benchmark design.

[17] "Siemens closes Mentor Graphics acquisition." Siemens. https://press.siemens.com/global/en/pressrelease/siemens-closes-mentor-graphics-acquisition. Published/updated: 2017-03-30. Accessed: 2026-08-11. Used for: positioning Mentor Graphics within Siemens EDA.
[18] "Automatically Improving LLM-based Verilog Generation using EDA Tool Feedback." Jason Blocklove et al. arXiv. https://arxiv.org/abs/2411.11856. Published/updated: 2025-03-04 (v3). Accessed: 2026-08-11. Used for: AutoChip, tool feedback, success, and cost findings.
[19] "How AlphaChip transformed computer chip design." Anna Goldie and Azalia Mirhoseini, Google DeepMind. https://deepmind.google/blog/how-alphachip-transformed-computer-chip-design/. Published/updated: 2024-09-26. Accessed: 2026-08-11. Used for: RL-based floorplanning and TPU deployment.
[20] "Generative AI for Chip Design." Synopsys. https://www.synopsys.com/ai/generative-ai.html. Published/updated: not listed. Accessed: 2026-08-11. Used for: Synopsys.ai Copilot capabilities for guidance, RTL, and formal testbenches.
[21] "Synopsys Advances Agentic AI Chip Design with AMD and Microsoft." Synopsys. https://investor.synopsys.com/news/news-details/2026/Synopsys-Advances-Agentic-AI-Chip-Design-with-AMD-and-Microsoft/default.aspx. Published/updated: 2026-07-27. Accessed: 2026-08-11. Used for: AgentEngineer workflows, evaluation status, and vendor-reported results.
[22] "Questa One Smart Verification Solution." Siemens EDA. https://eda.sw.siemens.com/en-US/ic/questa/. Published/updated: not listed. Accessed: 2026-08-11. Used for: current AI-assisted verification positioning from Siemens EDA.