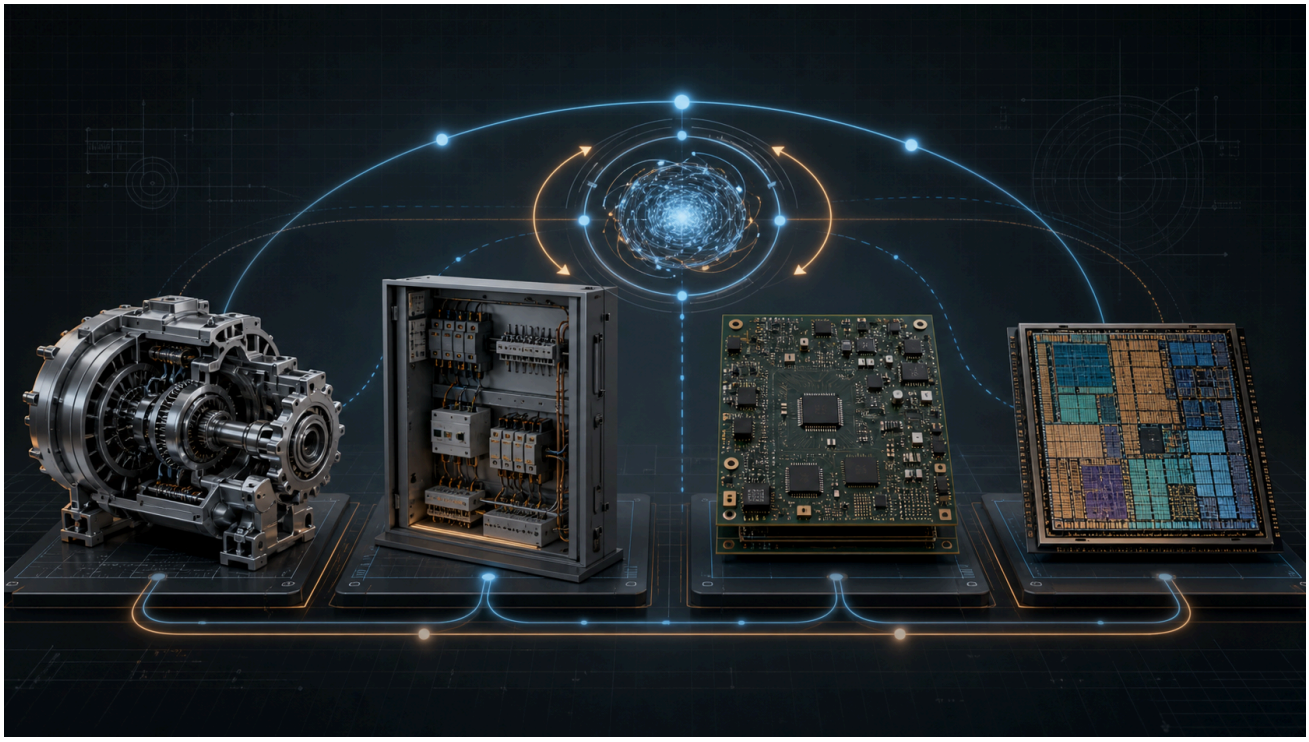
Die Konstruktion der Zukunft ist agentisch

Dr. Axel Richter
Engineering-Consult Dr. Richter, Aschaffenburg
Erstellt: 11. August 2026 · Geändert: 11. August 2026



Headergrafik: Eigene, KI-generierte Illustration.

Die nächste CAD-Revolution ist keine bessere Texteingabe. Sie beginnt, wenn ein KI-System ein Konstruktionsziel versteht, selbstständig die passenden Werkzeuge auswählt, Änderungen am nativen Modell ausführt, das Ergebnis prüft und aus Fehlern den nächsten Arbeitsschritt ableitet. Genau dieser geschlossene Regelkreis macht aus einem Copiloten einen Konstruktionsagenten.

Die Entwicklung steht noch ganz am Anfang. Trotzdem ist bereits erkennbar, dass sie Computer-Aided Design grundlegend verändern wird: mechanische Konstruktion ebenso wie Stromlauf- und Schaltschrankplanung, Leiterplattenentwicklung und – mit noch höheren Hürden – das Chipdesign. Der entscheidende Wandel liegt nicht darin, dass Ingenieurinnen und Ingenieure künftig weniger wissen müssen. Er liegt darin, dass Wissen, Absicht und Berechnung anders miteinander gekoppelt werden.

Heute übersetzen Menschen eine technische Absicht in eine lange Folge von Menübefehlen, Modellierungsoperationen, Bibliotheksrecherchen, Prüfungen und Korrekturen. Agentisches CAD verschiebt diese Übersetzungsarbeit in eine ausführbare Schleife. Der Mensch formuliert Anforderungen und Grenzen; der Agent plant und bedient Werkzeuge; CAD-Kernel, Regelprüfungen und Simulation liefern belastbare Evidenz; der Mensch behält Verantwortung und Freigabe.

Das klingt nach einem kleinen Unterschied. Tatsächlich ist es ein neues Betriebsmodell für Engineering-Software.

Was „agentisch“ an CAD wirklich bedeutet

Der Begriff wird derzeit großzügig verwendet. Eine nützliche Abgrenzung stammt von Anthropic: Ein *Workflow* folgt vorgegebenen Codepfaden, während ein *Agent* den eigenen Prozess und die Verwendung seiner Werkzeuge dynamisch steuert.[1] Übertragen auf CAD ergibt sich eine klare Leiter:

1. **Automatisierung** führt eine fest programmierte Folge aus: Makro starten, Stückliste erzeugen, Zeichnung exportieren.

2. **Assistenz** beantwortet Fragen oder schlägt Befehle vor, verändert das Modell aber nicht oder nur nach einem einzelnen, expliziten Auftrag.
3. **Generierung** erzeugt Geometrie, Code oder einen Entwurf aus Text beziehungsweise Bildern – häufig in einem offenen Durchlauf.
4. **Agentisches CAD** verfolgt ein Ziel über mehrere Schritte, beobachtet den Modellzustand, ruft CAD- und Prüfwerkzeuge auf, wertet Rückmeldungen aus und revidiert seinen Plan.

Ein generatives Modell kann beispielsweise aus „Baue einen 100 × 60 mm großen Montagewinkel mit vier M5-Bohrungen“ einen Körper erzeugen. Ein Agent fragt zusätzlich nach unklaren Randbedingungen, legt Parameter an, modelliert eine belastbare Feature-Historie, erkennt einen fehlenden Biegeradius, führt eine Kollisions- oder Herstellbarkeitsprüfung aus, korrigiert den Entwurf und legt die finale Zeichnungsableitung zur Freigabe vor.

Der Unterschied ist also nicht primär die Qualität des Sprachmodells. Es ist die Schleife aus **Wahrnehmen, Planen, Handeln, Prüfen und Korrigieren**. Ein solcher Prozess braucht standardisierte Werkzeugverträge, wie sie MCP mit Tools und Resources unterstützt,[2] sowie mehrschichtige Guardrails statt einer einzelnen Schutzmaßnahme.[3] Das Referenzwerk *The Hitchhiker's Guide to Agentic AI* beschreibt dafür den Agent Harness als Laufzeitschicht für Zustand, Werkzeugaufrufe, Fehlerbehandlung, Berechtigungen und Beobachtbarkeit.[4] Diese Schicht ist im Engineering mindestens so wichtig wie das Modell selbst.

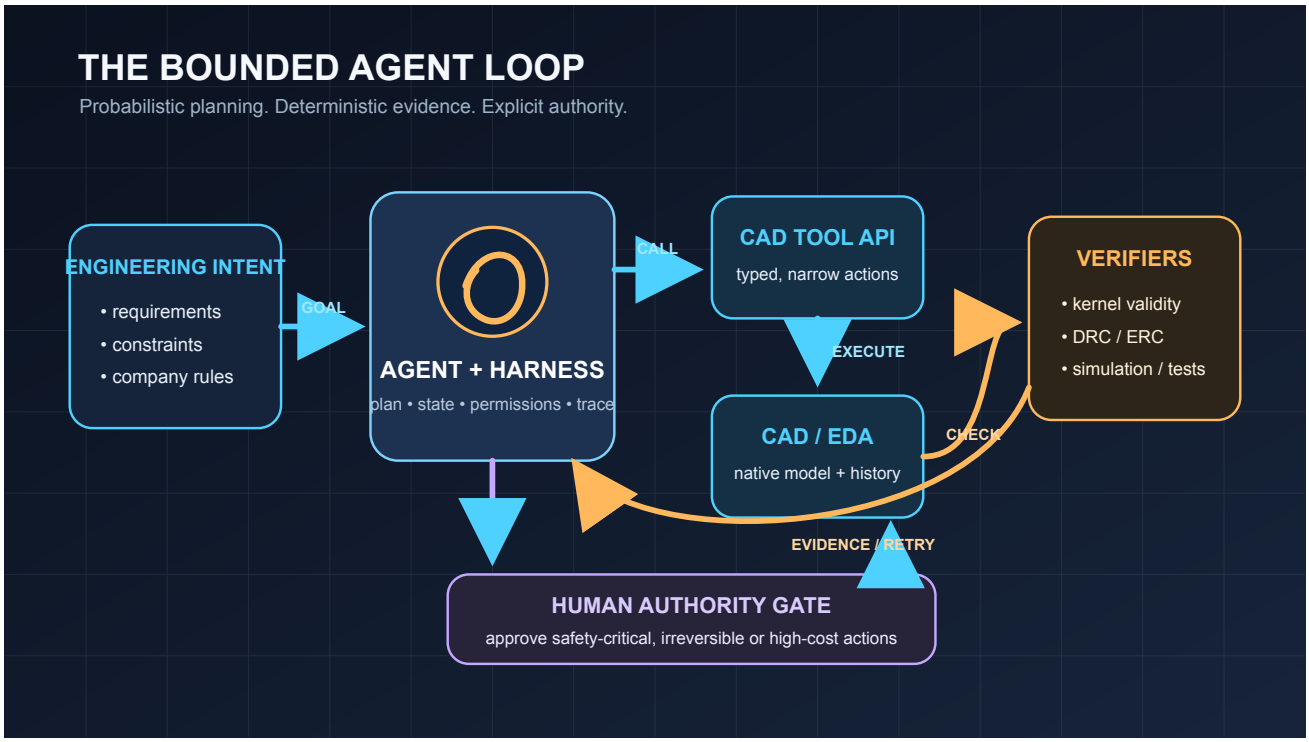


Abbildung 1: Der begrenzte Agentenregelkreis. Eigene Darstellung.

Formal lässt sich ein CAD-Agent als zustandsbehafteter Prozess ausdrücken. Aus Anforderungen, aktuellem Modellzustand und Beobachtungen wählt seine Policy die nächste Aktion. Das CAD-System überführt diese Aktion in einen neuen Zustand; ein unabhängiger Verifier bewertet anschließend das Ergebnis:

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

Formelgrafik 1: Zustandsübergang und Verifikation im agentischen CAD-Regelkreis.

Der Agent beendet die Arbeit nicht dann, wenn die Antwort plausibel klingt, sondern wenn definierte Prüfkriterien erfüllt sind, das Budget ausgeschöpft ist oder eine menschliche Entscheidung nötig wird. Das ist der zentrale

Denkfehler vieler Demos: Eine hübsche Geometrie oder ein kompilierendes Verilog-Modul ist noch kein belastbares Engineering-Ergebnis.

Was agentisches CAD nicht ist

Agentisches CAD ist nicht identisch mit Generative Design. Klassisches Generative Design durchsucht einen definierten Lösungsraum unter Last-, Geometrie- und Fertigungsbedingungen und liefert Varianten. Ein Agent kann eine solche Optimierung anstoßen, ihre Ergebnisse vergleichen, Randbedingungen verändern und anschließend Zeichnung, Dokumentation oder Folgeanalysen erstellen. Die Optimierung ist ein Werkzeug im Agentenprozess, nicht der Agent selbst.

Ebenso wenig ist jede Chatleiste ein Agent. Ein Assistent, der die Dokumentation durchsucht, ist nützlich – insbesondere in komplexen CAD-Systemen. Agentisch wird er erst, wenn er autorisierte Aktionen ausführt, deren Resultate liest und daraus weitere Handlungen ableitet. Genau an dieser Grenze bewegen sich viele heutige Produkte.

Die gemeinsame Architektur hinter MCAD, ECAD, PCB und IC

Die vier Domänen wirken auf den ersten Blick sehr verschieden. MCAD arbeitet mit Skizzen, B-Rep-Geometrie, Features und Baugruppen. Elektrische Planung arbeitet mit Symbolen, Potenzialen, Kabeln, Klemmen und Schaltschränken. PCB-Design verbindet logische Netze mit Footprints, Kupfer, Lagenaufbauten und Fertigungsregeln. Chipdesign bewegt sich von Spezifikation und RTL über Verifikation und Synthese bis zu Timing-, Leistungs- und Flächenabschluss.

Die agentische Grundarchitektur ist trotzdem dieselbe:

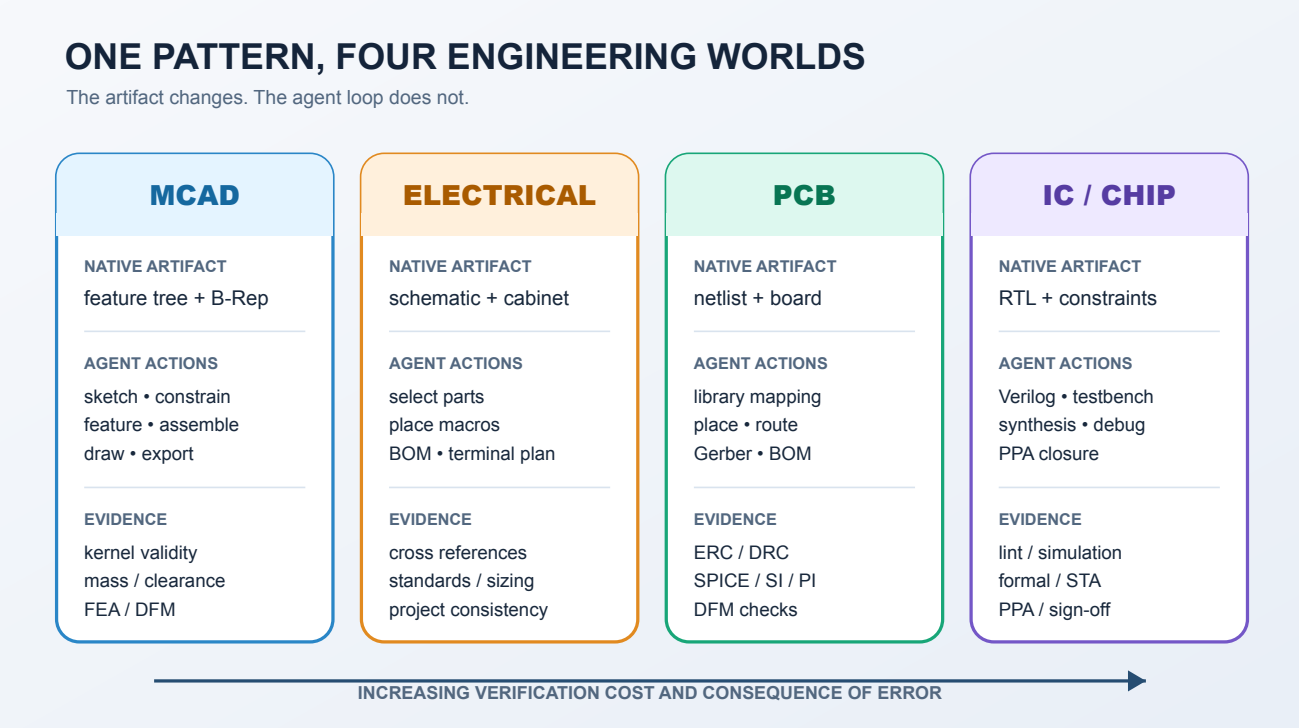


Abbildung 2: Ein Muster, vier Engineering-Welten. Eigene Darstellung.

1. Ein natives, editierbares Artefakt

Ein PNG eines Bauteils, ein trianguliertes Mesh oder ein PDF eines Schaltplans reicht nicht. Professionelles Engineering braucht das native Objektmodell: parametrische Features und B-Rep im MCAD, logisch verbundene Betriebsmittel im Stromlaufplan, Netze und Footprints auf dem PCB, RTL und Constraints im IC-Flow. Nur dann bleiben Design Intent, Änderbarkeit, Variantenfähigkeit und nachgelagerte Prozesse erhalten.

Das erklärt, warum „Text-to-3D“ nicht automatisch „Text-to-CAD“ ist. Eine visuell passende Oberfläche kann topologisch falsch, nicht wasserdicht, nicht bemaßbar oder nicht regenerierbar sein. Aktuelle Forschung bestätigt

die Lücke: Text2CAD-Bench umfasst 600 kuratierte Aufgaben über vier Komplexitätsstufen und berichtet, dass heutige Modelle bei einfachen Geometrien vernünftig arbeiten, bei komplexer Topologie und fortgeschrittenen Features aber deutlich abbauen.[5]

2. Eine schmale, typisierte Werkzeugoberfläche

Ein Agent sollte nicht „irgendwie klicken". Er braucht Werkzeuge mit eindeutigen Namen, Eingabeschemata, Vorbedingungen und Rückgabewerten: `create_sketch`, `place_component`, `run_drc`, `compile_rtl`, `compare_mass_properties`. Der Model Context Protocol (MCP) standardisiert genau diese Art der Verbindung zwischen KI-Anwendung und Werkzeugservern über eine Client-Server-Architektur mit Tools, Resources, Prompts und Benachrichtigungen.[2]

MCP ist dabei kein Qualitätszertifikat. Ein schlecht entworfenes oder zu mächtiges Werkzeug bleibt schlecht oder gefährlich, auch wenn es standardisiert aufgerufen wird. Der Wert liegt in der Entkopplung: CAD-Anbieter oder Integratoren können Fähigkeiten als versionierte Verträge bereitstellen, während Agentenmodelle austauschbar bleiben.

3. Ein Harness, der aus Modellaufrufen einen kontrollierten Prozess macht

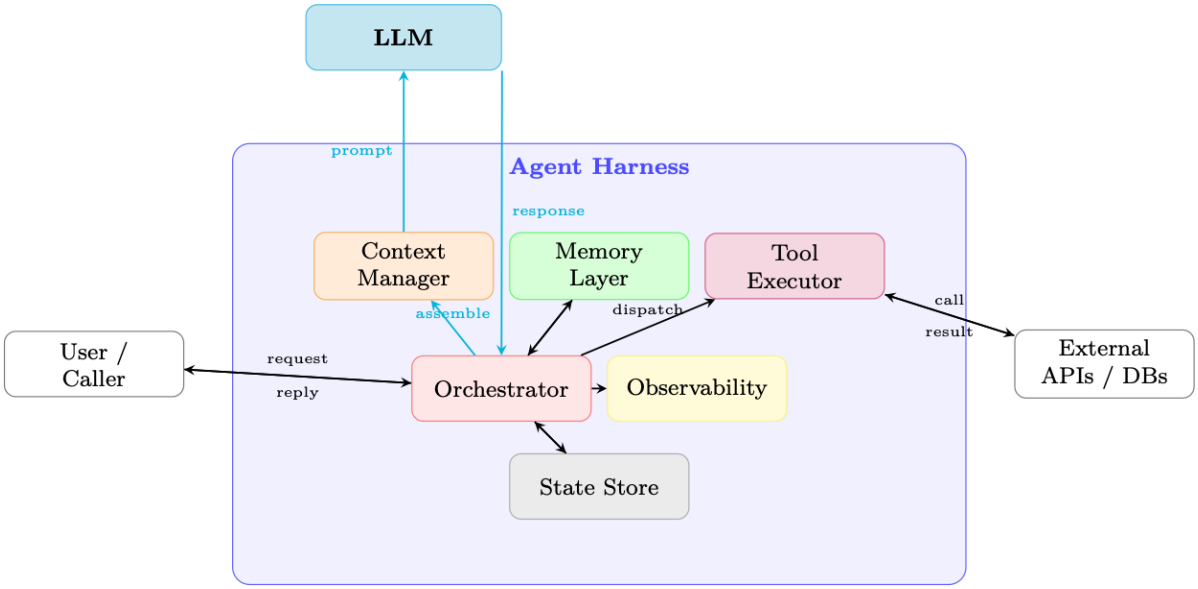


Abbildung 3: Agent-Harness-Architektur aus The Hitchhiker's Guide to Agentic AI. *Quelle und Einordnung: [4].*

Der Harness hält den aktuellen Arbeitsstand, begrenzt Schritte und Kosten, verwaltet Berechtigungen, protokolliert Werkzeugaufrufe und erkennt Schleifen. Er entscheidet außerdem, welche Informationen das Sprachmodell sieht. Bei einem realen Projekt können Stücklisten, Datenblätter, Normenauszüge, Materialtabellen, frühere Varianten und Prüfberichte den Kontext sprengen. Ein robuster Agent lädt daher nicht blind „das ganze Projekt", sondern stellt pro Schritt eine zielgerichtete Sicht zusammen.

OpenAI empfiehlt für Agentensysteme eine mehrschichtige Absicherung und weist darauf hin, dass einzelne Guardrails nicht genügen.[3] Im CAD-Kontext bedeutet das: Schema-Prüfung der Tool-Argumente, Grenzwerte für Parameter, projektbezogene Zugriffsrechte, transaktionale Änderungen, Undo/Restore-Punkte, deterministische Prüfungen und Freigaben müssen zusammenwirken.

4. Verifikation außerhalb des Sprachmodells

Das Sprachmodell darf Vorschläge machen und Fehlerberichte interpretieren. Es sollte aber nicht allein beurteilen, ob sein eigener Entwurf korrekt ist. Die entscheidende Evidenz kommt aus spezialisierten Systemen:

MCAD	Feature-Rebuild, gültiger Solid, Maße	Kollision, Toleranzkette, FEA, DFM	Zeichnung, Material, Normen, Review
ECAD	offene Verbindungen, Betriebsmitteldaten	Querverweise, Kabel-/Klemmenlogik, Dimensionierung	Normen, Kundenvorgaben, Schaltschrankprüfung
PCB	ERC, DRC, Bibliothekskonsistenz	SPICE, Signal-/Power-Integrity, Thermik	DFM/DFA, Fertigungsdaten, Review
IC	Lint, Compile, Unit-Testbench	Simulation, Formal, CDC/RDC, Synthese, STA	Coverage, PPA, Sign-off, Tape-out-Gates

Die richtige Arbeitsteilung lautet deshalb: **Das Sprachmodell plant probabilistisch; der Engineering-Stack prüft deterministisch.**

Wie man einen belastbaren CAD-Agenten baut

Zustandsmodell statt Chatverlauf

Der wichtigste technische Schritt ist ein explizites Zustandsmodell. Ein Chatprotokoll ist kein Project State. Ein CAD-Agent braucht mindestens:

- eine unveränderliche Anforderungsbasis mit Quellen und Versionen,
- den aktuellen nativen Dokument- und Revisionsstand,
- ein Ziel- und Constraint-Modell,
- eine Liste geplanter, ausgeführter und zurückgenommener Aktionen,
- Prüfresultate mit Tool-Version, Konfiguration und Zeitstempel,
- offene Annahmen, Risiken und Freigaben.

Eine Aktion sollte als typisierter, auditierbarer Datensatz formuliert sein, beispielsweise:

```
{
  "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"]
}
```

Die semantische Referenz `face:mounting_plate/top` ist absichtlich wichtiger als eine flüchtige interne Face-ID. In parametrischem CAD ändern sich Topologie und interne Kennungen nach Modelländerungen – das bekannte Topological-Naming-Problem. Ein Agent, der nur rohe IDs speichert, kann beim nächsten Schritt die falsche Fläche bearbeiten. Robuste Integrationen benötigen stabile Semantik, geometrische Signaturen oder eine kontrollierte Neuselektion.

Transaktionen, Diffs und Reversibilität

Jeder Agentenschritt sollte in einer Transaktion laufen:

1. Zustand und Dokumentrevision lesen.
2. Vorbedingungen prüfen.
3. Änderung in Branch, Kopie oder Undo-Scope ausführen.
4. Modell regenerieren und lokale Checks starten.
5. Strukturierten Diff erzeugen: Geometrie, Parameter, Netze, Bauteile oder RTL.
6. Änderung annehmen, reparieren oder vollständig zurückrollen.

„Undo“ ist dabei kein Komfortmerkmal, sondern ein Sicherheitsprimitive. Besonders gefährlich sind teilweise erfolgreiche Aktionen: drei Komponenten wurden korrekt platziert, die vierte schlug fehl, der Agent glaubt aber,

der gesamte Schritt sei atomar gewesen. Deshalb müssen Tool-Antworten zwischen `success`, `partial`, `failed` und `unknown` unterscheiden und betroffene Objekte exakt benennen.

Der Verifier steuert die Schleife

Eine praktische Zielfunktion kombiniert harte Constraints mit Optimierungszielen. Für einen mechanischen Entwurf könnte gelten:

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

Formelgrafik 2: Beispiel einer gewichteten Zielfunktion für einen mechanischen Entwurf.

Dabei steht m für die Masse, C für die Kostenabschätzung und T für eine thermische Kennzahl. Verletzungen der Geometrie- und Fertigungsbedingungen werden über große Lambda-Strafgewichte abgebildet. In der Praxis sollten harte Regeln nicht nur als weiche Strafe modelliert werden: Ein Selbstschnitt, ein Kurzschluss oder eine negative Timing-Slack muss den Kandidaten disqualifizieren.

Der Agent erhält anschließend keinen unspezifischen Satz wie „das ist falsch“, sondern strukturierte Evidenz:

```
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
```

Je maschinenlesbarer diese Rückmeldung, desto weniger muss das Sprachmodell raten.

Berechtigungen sind domänenspezifisch

Nicht jede Aktion hat dasselbe Risiko. Eine hilfreiche Policy unterscheidet:

- **Read:** Modell, Eigenschaften, Regeln und Prüfberichte lesen.
- **Draft:** Vorschläge, Varianten oder Branches erzeugen.
- **Modify:** ein aktives Dokument innerhalb definierter Grenzen ändern.
- **Release:** Revision freigeben, PLM-Status ändern oder Fertigungsdaten erzeugen.
- **Externalize:** Daten an Cloud-Dienste, Lieferanten oder Fertiger übertragen.

Die letzten beiden Klassen sollten standardmäßig menschliche Freigabe verlangen. Dasselbe gilt für Änderungen an sicherheitskritischen Anforderungen, Bauteilsubstitutionen ohne qualifizierte Alternative, Schutzschaltungen, Isolationsabstände oder Sign-off-Constraints. Die Frage lautet nicht „Wie autonom ist unser Agent?“, sondern „Innerhalb welches überprüfbaren Betriebsbereichs darf er autonom sein?“

Beobachtbarkeit: Jeder Schritt muss rekonstruierbar sein

Für einen Produktionsagenten müssen Teams später beantworten können:

- Welches Modell und welche Prompt-/Skill-Version haben entschieden?
- Welche Dokumente und Datenblätter lagen im Kontext?
- Welche Tools wurden mit welchen Argumenten aufgerufen?
- Was änderte sich am nativen Artefakt?
- Welche Prüfungen liefen, mit welcher Version und Konfiguration?
- Warum wurde eine Warnung akzeptiert oder eskaliert?

Ein vollständiger Trace enthält deshalb nicht nur Chattext, sondern Modellrevisionen, Tool-Calls, CAD-Diffs, Check-Artefakte, Laufzeiten, Kosten und Freigabeidentitäten. Das ist die Voraussetzung für Fehleranalyse, Audit und Reproduzierbarkeit.

Testen wie Software – plus Engineering-Evidenz

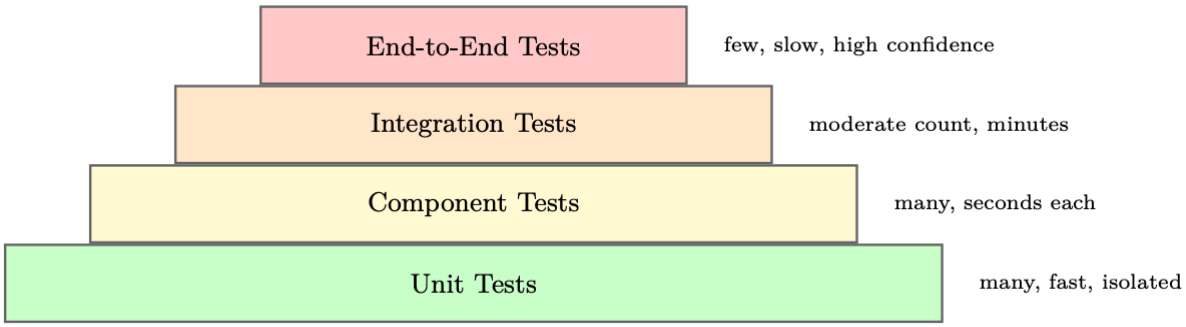


Abbildung 4: Testpyramide aus The Hitchhiker's Guide to Agentic AI. *Quelle und Einordnung: [4].*

Ein Agent benötigt Unit-Tests für Tooladapter, Schema-Validatoren und Zustandsübergänge; Komponententests mit simulierten CAD-Antworten; Integrationsläufe gegen echte CAD-Versionen; und wenige, teure End-to-End-Aufgaben mit fachlicher Abnahme. Zusätzlich braucht er ein Domänen-Evalset aus repräsentativen Konstruktionsaufgaben.

Die zentrale Kennzahl darf nicht „gefällt einem Sprachmodell" sein. Sinnvoller sind:

- **Task Success Rate:** Anteil vollständig erfüllter Aufgaben nach deterministischen Kriterien.
- **First-Pass Yield:** Anteil ohne menschliche Reparatur.
- **Constraint Violation Rate:** Verletzungen pro Aufgabe, nach Schweregrad.
- **Human Edit Distance:** Umfang der Änderungen bis zur Freigabe.
- **Tool Efficiency:** erfolgreiche Ergebnisse pro Toolaufruf, Zeit und Rechenbudget.
- **Recovery Rate:** Anteil korrekt reparierter Fehlerzustände.
- **Unsafe Action Rate:** unzulässige oder nicht genehmigte Aktionsversuche.

Eine einfache betriebliche Nutzenmetrik kann die Zeitersparnis gegen Nacharbeit und Risiko abwägen:

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

Formelgrafik 3: Betrieblicher Nutzen unter Berücksichtigung von Nacharbeit, Risiko und Rechenkosten.

Ein Agent, der schnell viele Entwürfe produziert, aber erfahrene Prüfer mit Nacharbeit überlastet, hat negativen Wert – selbst wenn seine Demo beeindruckt.

Typische Fehlerbilder

Visuell plausibel, technisch falsch	Bildfeedback ohne natives Modellverständnis	Kernel-, Regel- und Simulationschecks als Gates
Falsches Objekt verändert	instabile IDs oder mehrdeutige Auswahl	semantische Referenzen, Preview, Selektion bestätigen
Endlosschleife bei Reparaturen	kein Fortschrittsmaß, diffuse Fehlermeldung	Iterationsbudget, Zustands-Hash, strukturierte Verifier-Ausgabe
Norm oder Datenblatt halluziniert	ungrounded Modellwissen	freigegebene Quellen, Versions-/Seitenzitat, Konflikterkennung
Lokaler Check bestanden, System versagt	zu enger Prüfkontext	hierarchische Checks vom Feature bis zum System
IP verlässt die Organisation	unkontrollierter Cloud- oder Toolzugriff	Datenklassifizierung, lokale Ausführung, Egress-Policy, Audit
Agent „optimiert“ den Test weg	schlecht definierte Belohnung	unveränderliche Tests, getrennte Verifier, Human Gate

Aktuelle Landschaft: Wo die vier Domänen im August 2026 stehen

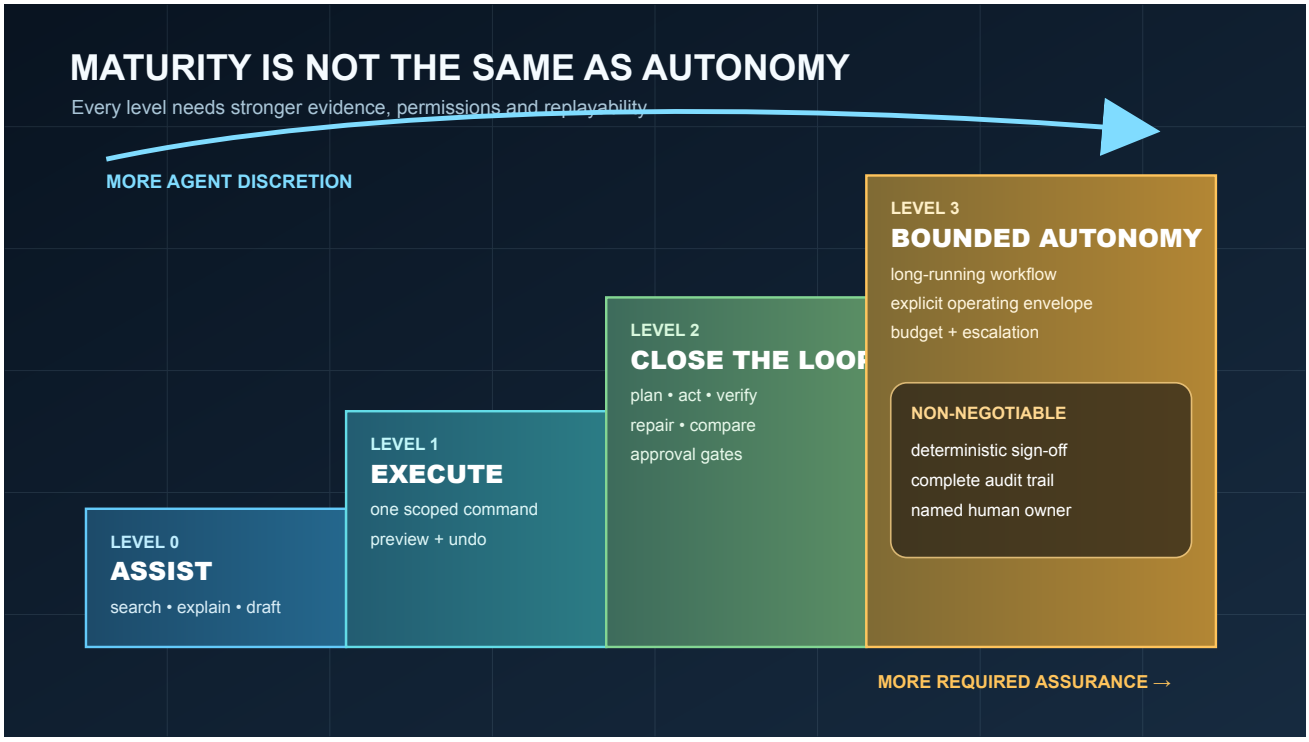


Abbildung 5: Reife ist nicht dasselbe wie Autonomie. Eigene Darstellung.

MCAD: Von der Hilfe zur nativen Modellaktion

Im mechanischen CAD ist die Entwicklung sichtbar, aber heterogen. Autodesk geht in Fusion bereits in Richtung nativer Modellaktion: Der Autodesk Assistant wird als konversationelle Erfahrung beschrieben, die Informationen findet, Workflows erklärt und Aufgaben per natürlicher Sprache ausführt. Autodesk kennzeichnet Teile wie Text-to-Command und native Geometrieerzeugung allerdings ausdrücklich als Entwicklung beziehungsweise Tech Preview; seit 2026 werden außerdem MCP-gestützte Fusion-Workflows beschrieben.[6] Diese Formulierung ist wichtig: Eine Vorschau ist ein Richtungssignal, keine garantierte Serieneigenschaft.

Siemens führte 2025 den Design Copilot NX als natürlichsprachliche Lern- und Bedienhilfe ein; die damalige Produktbeschreibung betonte vor allem dokumentationsgestützte Anleitung und das Starten genannter Befehle.[7] PTC brachte im Juni 2026 mit Creo 13 einen Creo AI Assistant in die Konstruktionsumgebung.[8]

Im Open-Source-Umfeld wird die Architektur besonders transparent. FreeCAD bietet eine umfangreiche C++- und Python-API,[9] und mehrere Community-Projekte exponieren Teilmengen davon über MCP. Ein aktuelles Beispiel beschreibt 32 Werkzeuge für parametrisches Modellieren, Prüfung, CAM und Export, weist aber zugleich auf Plattformgrenzen und den weitreichenden Zugriff auf FreeCADs Python-Umgebung hin.[10] Solche Projekte sind wertvolle Laboratorien für Tooldesign, Berechtigungen und Evals – aber kein Beleg dafür, dass beliebige Industriebauteile bereits autonom konstruiert werden können.

Der Engpass im MCAD ist nicht das Erzeugen eines Blocks mit Bohrungen. Schwierig sind robuste Feature-Historien, stabile Referenzen nach Änderungen, komplexe Freiformgeometrie, Baugruppenbeziehungen, Toleranzen, Material- und Fertigungswissen sowie die Überführung von „sieht richtig aus“ in „ist freigabefähig“. Genau deshalb ist die Entwicklung erst am Anfang.

Elektrische CAD-Planung: WSCAD zeigt, wie schnell Routine agentisch wird

In der elektrischen Planung ist WSCAD ein besonders greifbares Beispiel. Das Unternehmen stellte ELECTRIX AI 2024 mit einem integrierten AI Copilot vor. Laut Hersteller kann dieser unter anderem Designs prüfen, Stücklisten erzeugen sowie Makros und Komponenten platzieren.[11] Die Release Notes für ELECTRIX AI 2026 nennen konkrete Aktionen wie den Import von Teilen aus wscaduniverse.com per Befehl und die automatische, sequenzielle Positionierung eingefügter Pfadmakros.[12]

Das ist mehr als eine Suchhilfe: Natürliche Sprache löst strukturierte Aktionen im Projekt aus. Gleichzeitig sollte man die beworbenen Produktivitätswerte – etwa „bis zu 99 Prozent schneller“ – als Herstellerangaben und nicht als unabhängigen, domänenübergreifenden Benchmark lesen.[11]

Warum ist diese Domäne so geeignet? Viele Aufgaben sind repetitiv und stark strukturiert: Betriebsmittel auswählen, Makros einsetzen, Querverweise pflegen, Listen erzeugen, Texte übersetzen, Klemmen- und Schaltschrankdaten ableiten. Gleichzeitig existieren zahlreiche prüfbare Regeln. Das ergibt einen guten Agentenraum: hoher Routineanteil, klarer Projektzustand, viel vorhandenes Domänenwissen und messbare Resultate.

Die harte Grenze bleibt die fachliche Verantwortung. Leiterquerschnitt, Schutzkonzept, Selektivität, Kurzschlussfestigkeit, Wärmehaushalt und normative Konformität dürfen nicht allein auf einer sprachlich überzeugenden Antwort beruhen. Ein sinnvoller WSCAD- oder ECAD-Agent muss Unternehmensstandards und Projektrichtlinien mit Versionsstand zitierbar laden, Berechnungswerkzeuge aufrufen und Unsicherheit eskalieren.

Schaltplan und PCB: KiCad als offenes Experimentierfeld

KiCad ist für agentische PCB-Entwicklung besonders interessant, weil native Dateiformate, Kommandozeilenwerkzeuge, Regelprüfungen und eine wachsende API zusammenkommen. KiCad 10 erschien am 20. März 2026 und brachte unter anderem Varianten, PCB-Designblöcke und einen grafischen Editor für Designregeln.[13] Die offizielle IPC API verbindet externe Prozesse über Protocol Buffers und lokale IPC mit einer laufenden KiCad-Instanz. Die Dokumentation betont jedoch klare Grenzen: In KiCad 9 und 10 kommuniziert sie nur mit der GUI; der Abdeckungsgrad wird schrittweise erweitert, und Clients müssen Timeouts sowie synchrone Verarbeitung berücksichtigen.[14]

Genau diese nüchternen Schnittstellendetails entscheiden über Agentenqualität. Ein Agent muss erkennen, dass KiCad beschäftigt ist, darf keine konkurrierenden Verbindungen eröffnen und muss Export oder schematische Manipulation je nach Version anders behandeln. „MCP vorhanden“ ersetzt keine Kenntnis des darunterliegenden CAD-Lebenszyklus.

Die Forschung zeigt zugleich, wie schnell sich der geschlossene Regelkreis entwickelt. pcbGPT erzeugt editierbare KiCad-Schaltpläne aus natürlicher Sprache, verankert Bauteilauswahl in Bibliotheken und Datenblättern und kombiniert strukturelle sowie semantische Validierung. Auf 20 Embedded-Aufgaben erreicht das beste Modell laut Paper einen hohen First-Pass-Wert, fällt bei schweren Aufgaben aber ab; die Autoren halten Expert Review ausdrücklich weiterhin für notwendig.[15]

PCBWorld geht beim Routing einen ähnlichen Schritt: Agenten agieren über native KiCad-Operationen und verwenden DRC-Feedback, statt lediglich Koordinaten oder Bilder zu erzeugen. Das Benchmark-Environment

enthält synthetische Generatoren und 679 reale Open-Source-Boards und bewertet Ergebnisse mit acht engine-geprüften Metriken.[16] Das ist methodisch wichtiger als ein spektakulärer Screenshot: Der CAD/EDA-Kernel wird Teil der Evaluation.

Für reale Teams ist der beste Einstieg trotzdem nicht „Erzeuge das ganze Board". Sinnvoller sind begrenzte Aufgaben: Bibliotheksprüfung, BOM-Anreicherung, Platzierungsvarianten für unkritische Bereiche, DRC-Triage, Erzeugung von Fertigungsartefakten in einer Pipeline oder ein Review-Agent, der Verstöße priorisiert und mit Objektlinks versieht.

Chipdesign: Verilog ist nur der Anfang

Im Chipdesign werden die Chancen größer – und die Konsequenzen von Fehlern drastischer.

LLMs können Verilog schreiben, Testbenches erzeugen und Toolskripte erklären. Doch syntaktisch gültiges RTL sagt wenig über funktionale Korrektheit, Synthesizierbarkeit, Timing, Fläche, Leistung oder Verifikationsabdeckung. Eine Untersuchung zu AutoChip koppelte Sprachmodelle iterativ an Compiler- und Simulationsergebnisse. Toolfeedback verbesserte die Ergebnisse nicht bei allen Modellen konsistent; im besten Fall stieg die Zahl erfolgreicher Designs um 5,8 Prozent, während eine geschickte Modellmischung Kosten senkte.[18] Die Lehre ist nicht „Verilog ist gelöst", sondern: **EDA-Feedback kann helfen, wenn Modell, Schleife und Kostenstrategie gemeinsam evaluiert werden.**

Bei der physischen Implementierung zeigt AlphaChip eine andere Form von KI-gestütztem Design. Google DeepMind behandelt Floorplanning als sequenzielle Platzierungsaufgabe mit Reinforcement Learning und berichtet, dass die Methode für Blöcke mehrerer TPU-Generationen eingesetzt wurde.[19] Das ist hochwirksame spezialisierte Optimierung – aber kein allgemeiner Sprachagent, der einen Chip autonom von der Spezifikation bis zum Tape-out entwickelt.

Die aktuell deutlichste Bewegung in Richtung agentischer EDA kommt von den etablierten Toolketten. Synopsys beschreibt seinen Copilot für Wissenszugriff, RTL- und formale Testbench-Erzeugung sowie EDA-Workflows.[20] Im Juli 2026 kündigte das Unternehmen mit AMD und Microsoft zwei autonome Workflows zur Debug- und Implementierungs-Closure zur Evaluation an. Frühe, vom Anbieter berichtete Ergebnisse nennen 25 bis 40 Prozent kürzere Debug-Zyklen; der Zugang ist als Evaluation ausgewiesen, nicht als allgemeiner Nachweis vollautonomen Chipdesigns.[21]

Siemens EDA – aus der Mentor-Tradition – positioniert mit Questa One ebenfalls KI-gestützte Verifikation über Erstellung, Analyse, Debug und Regression.[22] In dieser Domäne wird die Agentenarchitektur wahrscheinlich zuerst bei Closure-Aufgaben reifen: Fehler clustern, Root Causes untersuchen, Assertions oder Tests vorschlagen, Toolparameter variieren und den nächsten Lauf planen. Tape-out-Freigaben, Sicherheitsziele und Sign-off bleiben deterministische, streng kontrollierte Prozesse.

Was sich grundsätzlich verändern wird

Die Benutzeroberfläche wird vom Befehlsraum zum Absichtsraum

Menüs verschwinden nicht. Experten werden weiterhin Geometrie direkt bearbeiten, Constraints setzen und Reports lesen. Aber die primäre Einheit der Interaktion verschiebt sich vom Befehl zum Ziel: „Reduziere die Masse um acht Prozent, ohne Eigenfrequenz, Gussradien und vorhandene Schnittstellen zu verletzen." Der Agent übersetzt dieses Ziel in einen überprüfbaren Plan und zeigt, was er warum ändern will.

CAD-Modelle werden ausführbare Wissensobjekte

Ein gutes Modell enthält künftig nicht nur Geometrie oder Netze, sondern maschinenlesbare Absichten, Quellen, Annahmen, Prüfregeln und Freigaben. Design Intent wird vom informellen Erfahrungswissen zur operativen Schicht. Das erhöht den Wert guter Namenskonventionen, sauberer Parameter, modularer Makros, gepflegter Bibliotheken und versionierter Requirements massiv.

Konstruktion und Verifikation rücken zusammen

Heute sind „modellieren“ und „prüfen“ häufig getrennte Phasen. Der Agent braucht dagegen nach fast jedem relevanten Schritt Feedback. Simulation, DRC, ERC, Lint, Formal, DFM und Kostenmodelle werden zu Werkzeugen in einer kontinuierlichen Suchschleife. Wer die Prüfpipeline nicht automatisiert hat, kann auch den Agenten nicht zuverlässig automatisieren.

Engineering-Organisationen werden Agentenplattformen betreiben

Ein Unternehmensagent kann nicht allein aus einem Modellabo bestehen. Er benötigt Toolserver, freigegebene Wissensquellen, Identitäts- und Rechtemanagement, Observability, Evaluationsdaten, Sandbox-Infrastruktur und Release-Prozesse. Das erzeugt eine neue Schnittstelle zwischen Konstruktion, CAE/EDA, PLM, IT-Sicherheit und AI Engineering.

Praktische Takeaways: So beginnt man ohne Autonomie-Theater

1. **Wählen Sie eine eng begrenzte, häufige Aufgabe.** Gute Kandidaten sind Stücklisten, Dokumentation, DRC-Triage, Varianten aus bekannten Templates, Parameteränderungen oder Prüfberichtszusammenfassungen. Schlechte erste Kandidaten sind sicherheitskritische Neuentwicklungen mit unklaren Anforderungen.
2. **Definieren Sie Erfolg vor dem Modell.** Legen Sie 30 bis 100 repräsentative Aufgaben, harte Constraints, erlaubte Abweichungen und Reviewkriterien fest. Ohne Evalset optimieren Sie auf Demos.
3. **Bauen Sie zuerst Read- und Verify-Tools.** Ein Agent, der den Modellzustand zuverlässig lesen und prüfen kann, ist bereits wertvoll. Schreibrechte kommen danach, zunächst nur in Branches oder Kopien.
4. **Halten Sie Werkzeuge klein und typisiert.** Lieber `set_parameter(name, value, unit)` und `run_drc(rule_set)` als ein universelles `execute_python(code)`. Mächtige Escape Hatches vervielfachen Risiko und erschweren Evals.
5. **Machen Sie jeden Schritt reversibel und sichtbar.** Preview, Diff, Undo, Checkpoint und Objektlinks sind keine UI-Details, sondern Trust-Infrastruktur.
6. **Trennen Sie Generierung und Prüfung.** Verwenden Sie CAD-Kernel, Solver, Regelprüfungen und unabhängige Verifier. Ein zweiter LLM-Call ist kein Ersatz für Physik oder Formalismus.
7. **Messen Sie Nacharbeit, nicht nur Geschwindigkeit.** Entscheidend ist die Zeit bis zur freigegebenen Lösung. Erfassen Sie First-Pass Yield, Human Edit Distance, kritische Verstöße und Recovery Rate.
8. **Steigern Sie Autonomie nur mit Evidenz.** Die Reifeleiter führt von Assistenz über einzelne Aktionen zum geschlossenen Regelkreis und erst dann zu begrenzter Autonomie. Jeder Schritt verlangt bessere Tests, feinere Rechte und stärkere Beobachtbarkeit.

Fazit

Agentisches CAD wird die Ingenieurarbeit nicht dadurch verändern, dass ein Chatbot plötzlich „konstruiert“. Es wird sie verändern, weil technische Absicht, Softwareaktionen und Verifikation in einer fortlaufenden, maschinenlesbaren Schleife zusammenfinden.

MCAD zeigt heute den Übergang von Hilfe zu nativen Modellaktionen. WSCAD demonstriert, wie schnell strukturierte elektrische Routinearbeit durch einen Copiloten ausführbar wird. KiCad macht durch offene Formate, API und Regelprüfungen sichtbar, wie agentische Schaltplan- und PCB-Flows aufgebaut und bewertet werden können. Im Chipdesign bewegen sich Synopsys, Siemens EDA und spezialisierte Ansätze wie AlphaChip in Richtung längerer, stärker autonomer Optimierungs- und Closure-Schleifen.

Aber der Maßstab bleibt Engineering, nicht Eloquenz: Ist das Artefakt nativ und editierbar? Sind Annahmen und Quellen sichtbar? Besteht es deterministische Checks? Ist jede Änderung nachvollziehbar und reversibel? Weiß der Agent, wann er stoppen und einen Menschen fragen muss?

Die Unternehmen, die diese Fragen früh beantworten, werden nicht nur schneller konstruieren. Sie werden Engineering-Wissen in eine neue, ausführbare Form überführen. Genau darin liegt das transformative Potenzial – und genau deshalb stehen wir erst am Anfang.

Quellen

[1] "Building effective agents." Anthropic. https://www.anthropic.com/engineering/building-effective-agents. Published/updated: 2024-12-19. Accessed: 2026-08-11. Used for: Abgrenzung zwischen Workflows und Agenten.

[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-Architektur, Rollen und Protokollprimitive.

[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 und Orchestrierungsmuster.

[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- und Testpyramiden-Framing sowie Abbildungen 3 und 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: Stand und Grenzen komplexer Text-to-CAD-Generierung.

[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 Geometrie und MCP-Richtung 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 und dessen ursprüngliche Positionierung.

[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 und Produktstand 2026.

[9] "FreeCAD source documentation." FreeCAD. https://www.freecad.org/api/. Published/updated: not listed. Accessed: 2026-08-11. Used for: Verfügbarkeit der C++- und Python-Programmierschnittstellen.

[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: aktuelles Open-Source-Beispiel und dessen Sicherheits-/Plattformgrenzen.

[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: Einführung, Funktionen und als Herstellerangaben gekennzeichnete Produktivitätsclaims.

[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: konkrete AI-Copilot-Aktionen 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-Funktionen und aktueller Produktstand.

[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-Architektur, Status, Grenzen und Betriebsverhalten.

[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 und Grenzen.

[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 und Benchmarkaufbau.

[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: Einordnung von Mentor Graphics in 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, Toolfeedback, Erfolgs- und Kostenbefunde.

[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-basiertes Floorplanning und TPU-Einsatz.

[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-Funktionen für Guidance, RTL und formale 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, Evaluationsstatus und vom Anbieter berichtete Ergebnisse.

[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: aktuelle KI-gestützte Verifikationspositionierung von Siemens EDA.