◆ DWX 2026

Designing AI-Enabled .NET Applications with Microsoft Agent Framework

Wie viel Entscheidungsfreiheit geben wir der KI, und wann ist deterministischer Code die bessere Wahl?

Dr. Matthias Liebeck

30.06.2026, 11:45 Uhr · m:con Congress Center Rosengarten, Mannheim

Agenda

1 Einstieg
2 Was kann mit dem Microsoft Agent Framework (MAF) gebaut werden?
3 Wie viel soll die KI entscheiden?
4 Die Bausteine von MAF
5 Orchestrierung
6 Abwägungen und Grenzen
7 Beispiele aus der Praxis
8 Fazit

Einstieg

Kurz zu mir

- Senior Solution Architect in Düsseldorf, Schwerpunkt .NET, C# und Azure
- Software-Entwickler seit 2009, C# seit 2006
- 2009 – 2015: Studium Informatik & Mathematik
- 2018: Promotion in Informatik über Natural Language Processing / KI
- Organisator des Azure Düsseldorf Meetups, Speaker über Azure & agentic coding
- GitHub Copilot Newsletter: ghcp.liebeck.io
- Dieser Vortrag aus Praktiker-Sicht, kein Framework-Marketing
- **Ehrlich vorab:** MAF ist neu. Manches habe ich evaluiert, aber nicht unter Produktionslast getestet

Azure Meetup Düsseldorf

Microsoft Certified: Azure Solutions Architect Expert

Microsoft Certified: DevOps Engineer Expert

Microsoft Certified: Azure Developer Associate

Microsoft Certified: Azure Administrator Associate

Microsoft Certified: Azure Data Scientist Associate

GitHub Copilot Certification Program

GitHub Copilot Newsletter

Das Problem: ein LLM aufrufen kann jeder

```
var client = new AzureOpenAIClient(endpoint, credential);
var chat = client.GetChatClient("gpt-4o-mini");

var res = await chat.CompleteChatAsync(
    "Fasse diesen Text zusammen: ...");

Console.WriteLine(res.Value.Content[0].Text);
```

- Ein paar Zeilen mit dem Azure-OpenAI-Client, Prompt rein, Antwort raus
- Für einmalige Aufgaben wie Zusammenfassen oder Klassifizieren reicht das
- Die Einstiegshürde ist nicht mehr die Technik, sondern die Architektur
- Spannend wird es, wenn KI Teil der laufenden Anwendungslogik wird

Was an einem einzelnen Call fehlt

- **Kein State:** jeder Aufruf ist isoliert, der Kontext geht verloren
- **Keine Tool-Auswahl:** das Modell kann nicht selbst APIs oder Datenbanken aufrufen
- **Keine Orchestrierung:** mehrstufige Abläufe musst du selbst verdrahten
- **Keine Nachvollziehbarkeit:** kein Tracing, was das Modell warum getan hat
- **Keine Kontrolle über Fehlverhalten:** keine Guardrails, keine Freigaben, keine Stopp-Bedingungen
- Ein einzelner Call ist kein System. Diese Lücke füllt ein Agent-Framework

Von Semantic Kernel zu MAF

- MAF ist der Nachfolger von Semantic Kernel, kein konkurrierendes Produkt
- Es führt Semantic Kernel und AutoGen zusammen
- Version 1.0 seit April 2026, Semantic Kernel im Maintenance-Modus
- Die Konzepte bauen aufeinander auf, für Neues nimmt man MAF

Was kann mit MAF gebaut werden?

◆ ORIENTIERUNG

Zwei Bausteine: Agent und Workflow

MAF hat zwei Grundbausteine. Den Unterschied zu kennen, macht alles danach leichter.

Agent

das LLM bestimmt den Weg

- Wird vom LLM gesteuert und nutzt Tools
- Entscheidet dynamisch, je nach Gespräch
- Kein fester Ablauf vorgegeben

Workflow

du bestimmst den Weg

- Ein fest definierter Ablauf aus Schritten
- Du legst die Reihenfolge fest, garantiert
- Kann Agenten als einzelne Schritte enthalten

Die fünf Orchestrierungen (sequential, concurrent, handoff, group chat, magentic) sind fertige Workflows, um mehrere Agenten zu koordinieren.

Quelle: Microsoft Learn, Agent Framework, Workflows

Was kann man mit MAF bauen?

Smarter Assistent

Nutzt eigene APIs und Daten. Etwa ein Support- oder Beratungs-Assistent.

KI-Schritt in Automatisierung

Eine sonst deterministische Pipeline mit einer KI-Entscheidung darin.

Mehrere Agenten

Zusammenarbeit, etwa Entwurf, Prüfung und Freigabe. Dazu gibt es mehrere Orchestrierungs-Muster

Die Bausteine dahinter: Agenten, Tools, Sessions, Orchestrierung und Observability.

◆ ORIENTIERUNG

Konkrete Szenarien

Interner Support-Bot

schlägt Tickets in eurer Datenbank nach

Angebots-Assistent

bereitet aus Stammdaten ein Angebot vor

Dokumente klassifizieren

nächtliche Einordnung eingehender Belege

Onboarding-Helfer

beantwortet Fragen aus internen Dokumenten

Recherche-Assistent

fasst mehrere Quellen zusammen

Freigabe-Workflow

schlägt vor, Mensch gibt frei

◆ ORIENTIERUNG

Womit fängt man an?

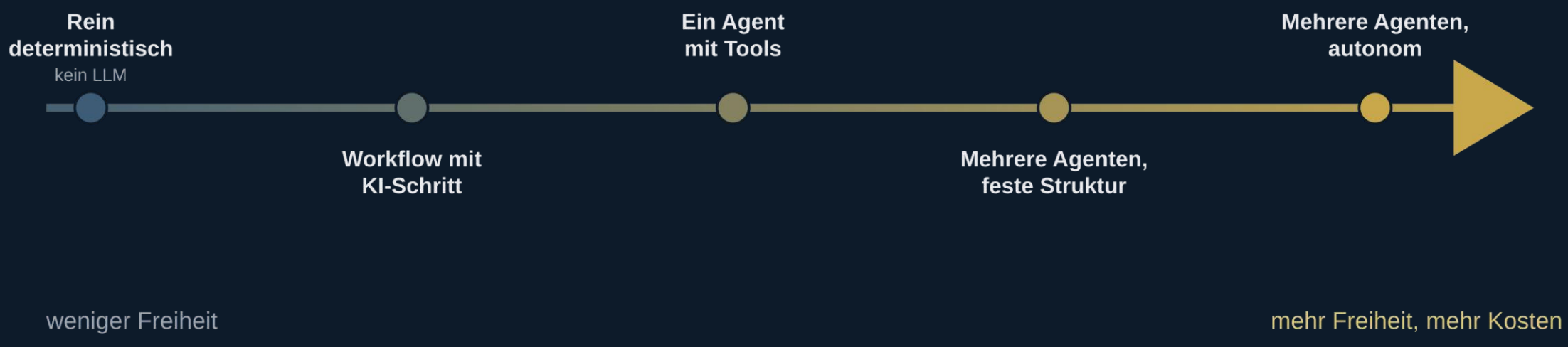
Ein Agent mit ein, zwei Tools löst die meisten Fälle. Mehr braucht es erst, wenn ein Agent nicht mehr reicht.

- Mit einem klaren, abgegrenzten Use Case starten
- Ein Tool nach dem anderen ergänzen, nicht alles auf einmal
- Orchestrierung erst, wenn ein einzelner Agent an Grenzen stößt
- **Und immer fragen: braucht dieser Schritt überhaupt KI?**

Wie viel soll die KI entscheiden?

Die Frage ist nicht ob, sondern wie viel KI

Sondern: Wie viel Entscheidungsfreiheit gebe ich der KI? Das ist ein Spektrum, nicht binär.



Je weiter rechts, desto höher Kosten und Unvorhersehbarkeit. Im Zweifel weiter links anfangen, als man denkt.

◆ WO ZIEHE ICH DIE GRENZE?

Drei Fragen vor jeder Entscheidung

1

Vorhersagbarkeit

Ist der Ausführungspfad fest oder hängt er von Zwischenergebnissen ab?

2

Kosten

Wie oft und wie teuer rufe ich das Modell auf?

3

Testbarkeit

Wie stelle ich sicher, dass das Ergebnis stimmt?

Kriterium 1: Vorhersagbarkeit

- Deterministisch bei
  - klarem, gleichbleibendem Prozess
  - wenn Compliance Reproduzierbarkeit verlangt
  - wenn die Kosten von Fehlverhalten hoch sind
- KI, wenn der nächste Schritt von Zwischenergebnissen abhängt

💡 **Faustregel:** Was du als festes Flussdiagramm zeichnen kannst, braucht keinen Agent.

Kriterium 2: Kosten

80–90%

der laufenden GenAI-Kosten ist Inferenz, nicht einmaliges Training (Branchenwert)

10–20 Calls

pro Aufgabe bei einem iterativen Agent statt eines einzigen Calls

Quellen: FinOps Foundation; Azure Architecture Center (Orchestrierungskosten); Azure OpenAI Pricing

- Abgerechnet wird pro Token, nicht pro Call
- Eingabe und Ausgabe getrennt, Ausgabe rund 4x teurer
- Jeder Call kostet anders, mehr Kontext heißt mehr Tokens
- Hebel: was ans LLM geht, wie oft und welches Modell, kleine Modelle für einfache Schritte
- Deterministische Schritte dort lassen, wo sie hingehören

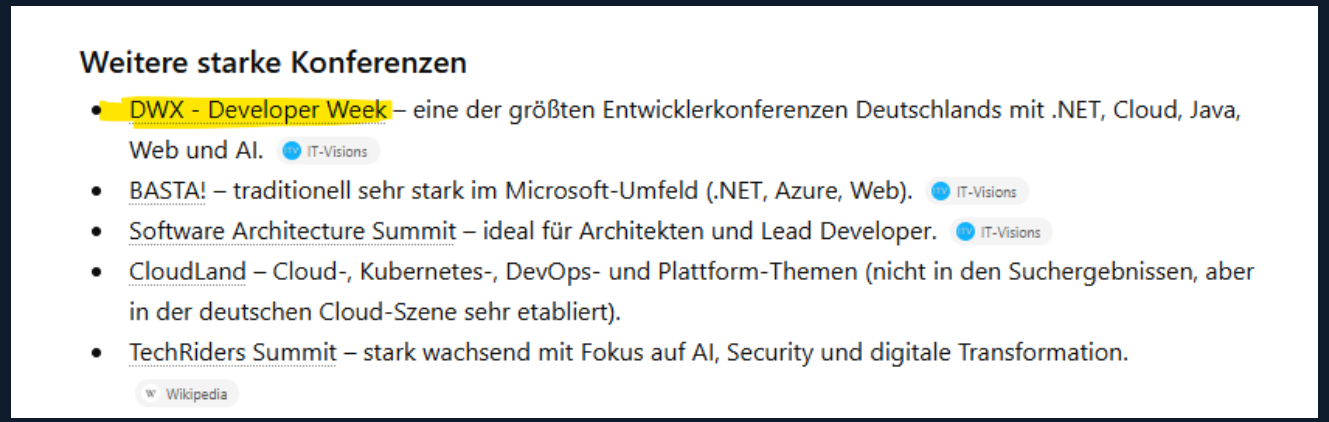
Kriterium 3: Testbarkeit

- Klassisches Testing setzt Determinismus voraus, 2 + 3 ergibt immer 5
- Ein LLM zweimal aufgerufen ergibt oft zwei verschiedene Antworten, beide können korrekt sein

Was ist eine gute Entwickler-Konferenz in Deutschland?

DWX nicht unter ersten 3 Treffern, nur unter „weitere“

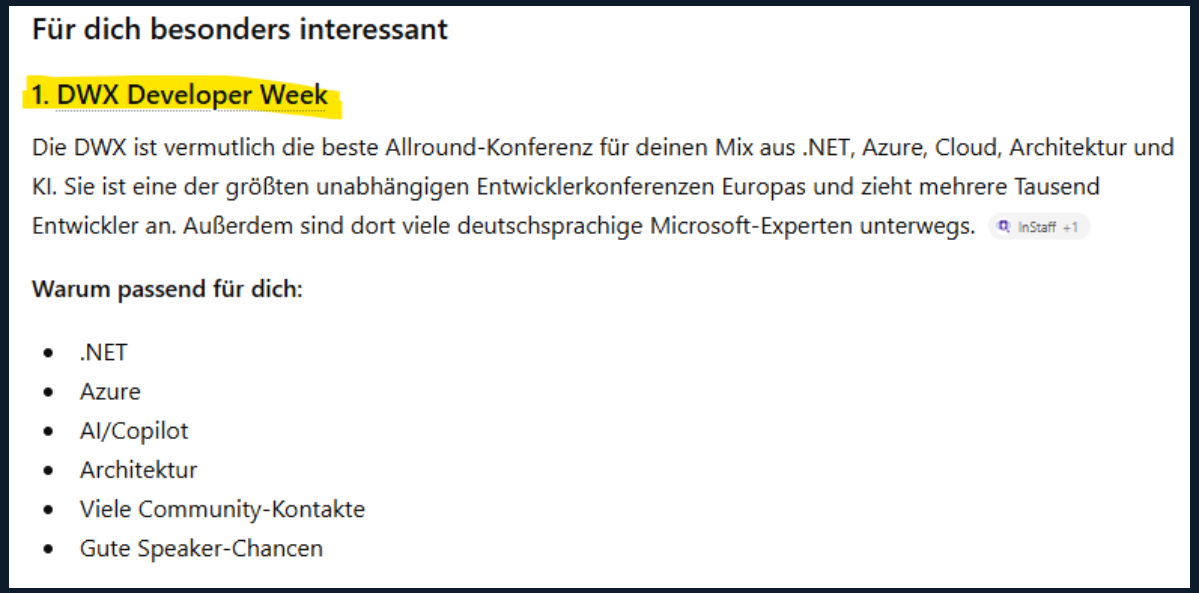
DWX ist erster Treffer



Weitere starke Konferenzen

- DWX - Developer Week – eine der größten Entwicklerkonferenzen Deutschlands mit .NET, Cloud, Java, Web und AI. IT-Visions
- BASTA! – traditionell sehr stark im Microsoft-Umfeld (.NET, Azure, Web). IT-Visions
- Software Architecture Summit – ideal für Architekten und Lead Developer. IT-Visions
- CloudLand – Cloud-, Kubernetes-, DevOps- und Plattform-Themen (nicht in den Suchergebnissen, aber in der deutschen Cloud-Szene sehr etabliert).
- TechRiders Summit – stark wachsend mit Fokus auf AI, Security und digitale Transformation.

Wikipedia



Für dich besonders interessant

1. DWX Developer Week

Die DWX ist vermutlich die beste Allround-Konferenz für deinen Mix aus .NET, Azure, Cloud, Architektur und KI. Sie ist eine der größten unabhängigen Entwicklerkonferenzen Europas und zieht mehrere Tausend Entwickler an. Außerdem sind dort viele deutschsprachige Microsoft-Experten unterwegs. InStaff +1

Warum passend für dich:

- .NET
- Azure
- AI/Copilot
- Architektur
- Viele Community-Kontakte
- Gute Speaker-Chancen

Kriterium 3: Testbarkeit

- Echten Determinismus, identischer Output bei identischem Input, gibt es mit LLMs nicht
- Varianz einschränkbar durch structured outputs, temperature null, feste Tool-Call-Sequenzen
- Eliminieren lässt sie sich nicht, das muss man bei Architektur und Tests einplanen

◆ WO ZIEHE ICH DIE GRENZE?

Direkter LLM-Call oder MAF?

Bei einem einzelnen, zustandslosen Prompt ist ein Framework Overhead. Es lohnt sich, sobald mehrere Punkte zutreffen:

- Brauche ich State über mehrere Turns hinweg?
- Habe ich mehrere Tools, deren Auswahl das Modell treffen soll?
- Muss ich mehrere Schritte orchestrieren?
- Brauche ich Observability und Human-in-the-Loop als Querschnitt?

Je mehr Ja, desto klarer zahlt sich MAF aus, denn die Infrastruktur selbst zu bauen wäre teurer.

◆ PRODUKTIONSREIFE

Vier MAF-Features für Produktion

Middleware-Pipeline

Guardrails, Logging, Validierung auf Microsoft.Extensions.AI. Funktioniert wie ASP.NET Core Middleware.

OpenTelemetry

Jeder Tool-Call und Orchestrierungsschritt wird getract. Die Antwort auf: wie debugge ich einen Agenten?

Human-in-the-Loop

Tools als freigabepflichtig markieren, der Run pausiert bis zur menschlichen Freigabe. Mehr dazu gleich.

MCP-Support

Externe Tools dynamisch entdecken (Model Context Protocol, ein offener Standard für Tool-Anbindung).

◆ PRODUKTIONSREIFE

Human-in-the-Loop: Freigabe vor der Aktion

1 Tool als freigabepflichtig markieren (C#: ApprovalRequiredAIFunction)

2 Will der Agent es aufrufen, pausiert der Run statt auszuführen

3 Statt finaler Antwort kommt eine Freigabe-Anfrage mit Tool-Name und Argumenten

4 Mensch entscheidet: Freigabe führt aus, Ablehnung geht als Feedback zurück ans Modell

Wer genehmigt? Eine Architekturentscheidung

- **Endnutzer selbst:** direkt im Ablauf, z. B. „500 Euro überweisen, ok?“
- **Anderer Mensch:** im Hintergrund, oft zeitversetzt über Liste oder Dashboard
- **Niemand:** bei unkritischen Schritten

Das Framework liefert

- Pause und Resume des Laufs
- Gatekeeper vor der Ausführung
- an Checkpoints gekoppelt

Dein Job

- was der Genehmiger sieht
- wer genehmigen darf
- Authentifizierung

Das Framework liefert nur Pausieren und Fortsetzen, das Übrige baust du.

Quelle: Microsoft Learn, Agent Framework, Tool Approval und Human-in-the-Loop

Wann KEIN Agent?

> *“If you can write a function to handle the task, do that instead of using an AI agent.”*

Offizielle Microsoft-Empfehlung

- Aufgabe deterministisch lösbar? Dann schreib die Funktion, kein LLM
- Reproduzierbarkeit oder Compliance zwingend? Dann deterministischer Pfad
- Einfache Klassifikation, die ein Regelwerk oder ein kleineres Modell auch schafft
- Kosten von Fehlverhalten hoch und kein menschlicher Review möglich? Dann kein autonomer Agent

👉 Wichtig ist der bewusste Einsatz von KI

Quelle: Microsoft Learn, Agent Framework Overview – learn.microsoft.com/en-us/agent-framework/overview

Die Bausteine von MAF

Ein Agent in wenigen Zeilen

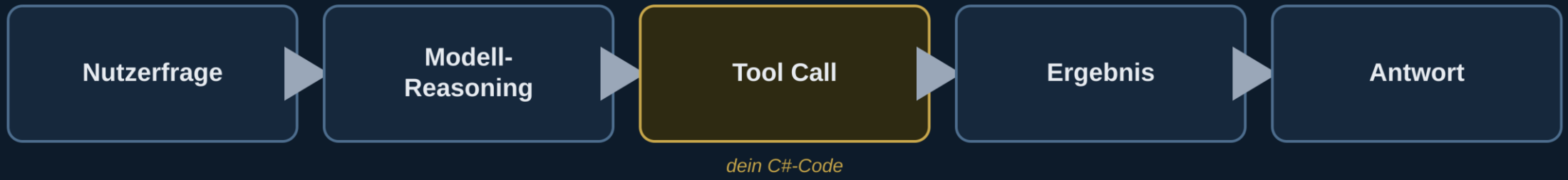
```
AIAgent agent =
  new AzureOpenAIClient(endpoint, cred)
    .GetChatClient(deployment)
    .AsAIAgent(instructions:
        "Du bist ein Beratungs-Assistent ...");

var reply = await agent.RunAsync(
    "Welche Infos brauchst du von mir?");
```

- Der Einstieg ist bewusst niedrigschwellig
- Eine Frage rein, eine Antwort raus, wie der direkte Call, aber mit Abstraktion
- Der Unterschied zeigt sich erst mit Tools und State
- Setup braucht etwas Azure-Vorarbeit, der Agent-Code selbst bleibt schlank

Tools: das LLM entscheidet

Das LLM entscheidet selbst, wann dein Tool aufgerufen wird




```
[Description("Sucht in der Produktdoku.")]
string SearchDocs(string query) => _docs.Find(query);

agent = chatClient.AsAIAgent(
    tools: [AIFunctionFactory.Create(SearchDocs)]);
```

- Description ist die Bedienungsanleitung fürs Modell: wann das Tool zu nutzen ist
- **Kein if/else:** das LLM wählt selbst, ob und wann es aufruft
- Hier verläuft die Grenze zwischen deinem Code und der KI-Entscheidung

Sessions: Kontext bleibt erhalten

```
var thread = agent.GetNewThread();

await agent.RunAsync(
    "Wir haben 3 Räume.", thread);

await agent.RunAsync(
    "Und wie viele Schalter?", thread);
// der Agent kennt den Kontext
```

- Eine Session hält den Gesprächsverlauf über mehrere Turns
- Folgefragen funktionieren ohne manuelles Mitschicken des Kontexts
- Die Basis für den Beratungs-Use-Case später
- Längerer Kontext heißt mehr Tokens. State ist Feature und Kostenfaktor zugleich

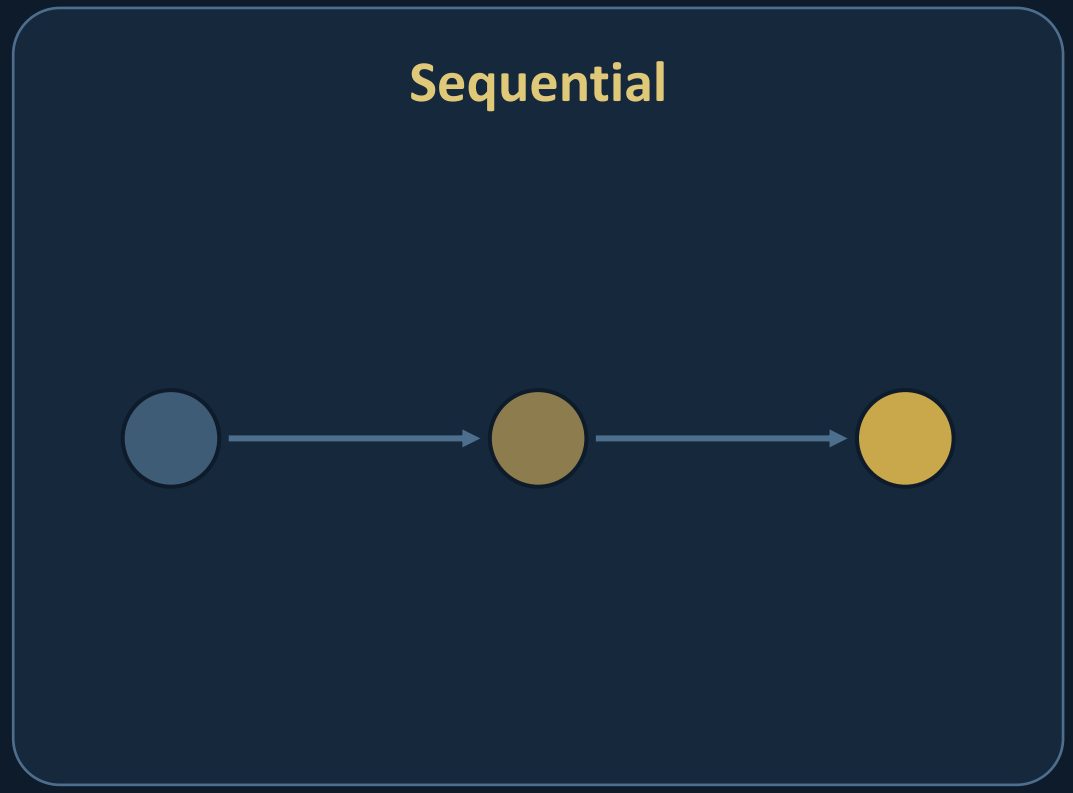
Orchestrierung

Die fünf MAF-Orchestrierungen

Erst hier kommen die benannten Muster ins Spiel, vom einfachsten zum flexibelsten:

Quelle: Microsoft Learn, Agent Framework, Workflow Orchestrations – learn.microsoft.com/en-us/agent-framework/workflows/orchestrations

Sequential



Was es ist

Agenten in fester Reihenfolge, jeder baut auf dem Ergebnis des vorherigen auf.

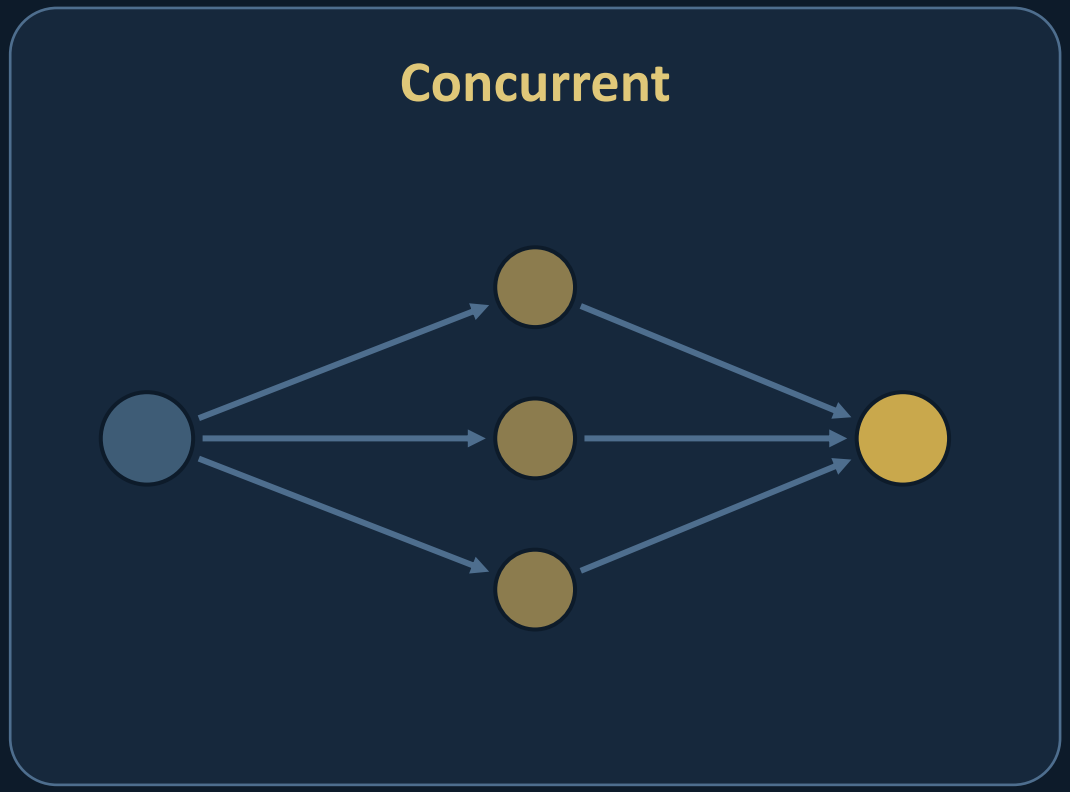
Wann nutzen

Klar definierte Schritte, etwa Entwurf, dann Review, dann Freigabe.

Kosten / Hinweis

Kosten summieren sich pro Schritt, dafür voll nachvollziehbar.

Concurrent



Was es ist

Mehrere Agenten bearbeiten dieselbe Aufgabe parallel, Ergebnisse werden zusammengeführt.

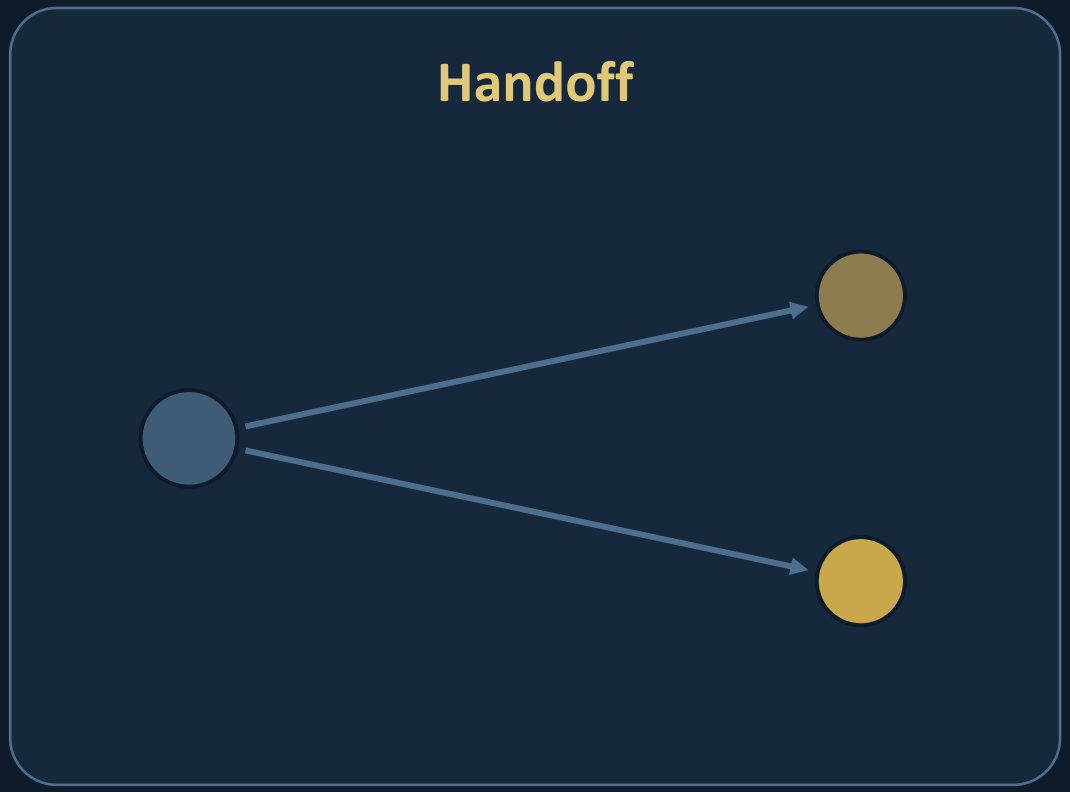
Wann nutzen

Mehrere Perspektiven oder Abstimmung, etwa Brainstorming oder Voting.

Kosten / Hinweis

Schneller, kann aber Ressourcen und Kosten gleichzeitig hochtreiben.

Handoff



Was es ist

Ein Agent übergibt die Kontrolle kontextabhängig an den passenden Spezialisten.

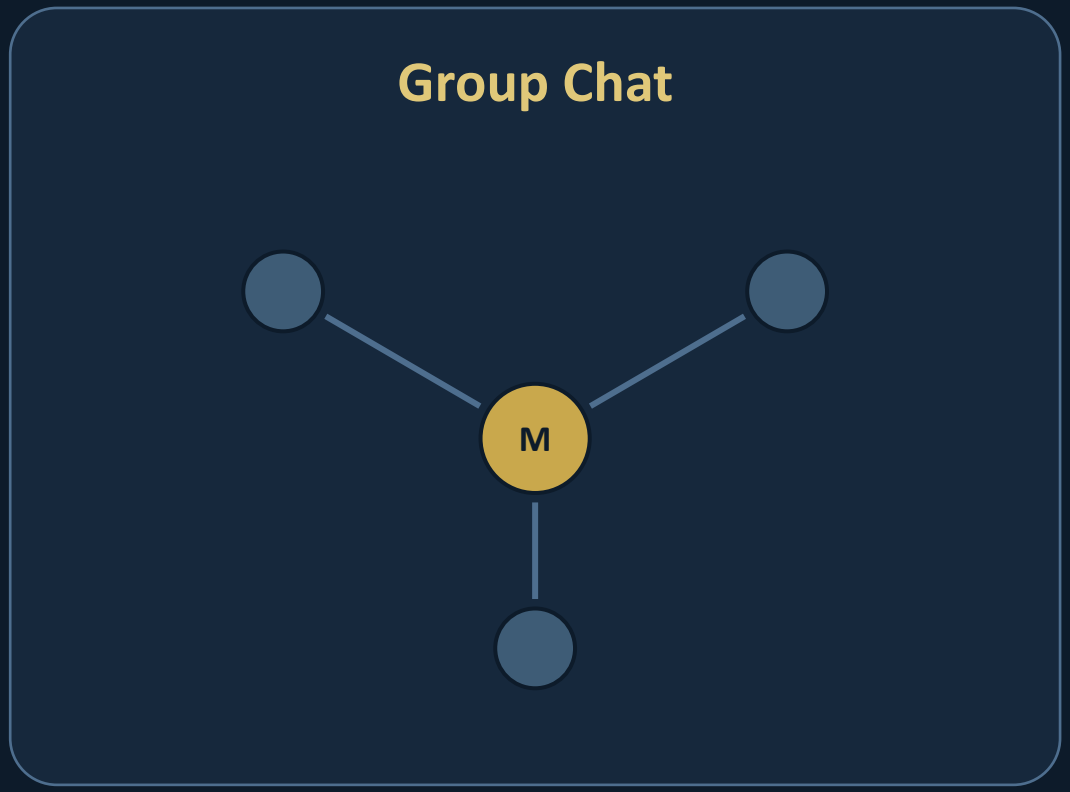
Wann nutzen

Dynamische Weiterleitung, etwa Support an den Fachbereich.

Kosten / Hinweis

Flexibel, aber Übergaben brauchen klare Zuständigkeiten.

Group Chat



Was es ist

Ein Orchestrator koordiniert, welcher Agent als Nächstes spricht, über mehrere Runden.

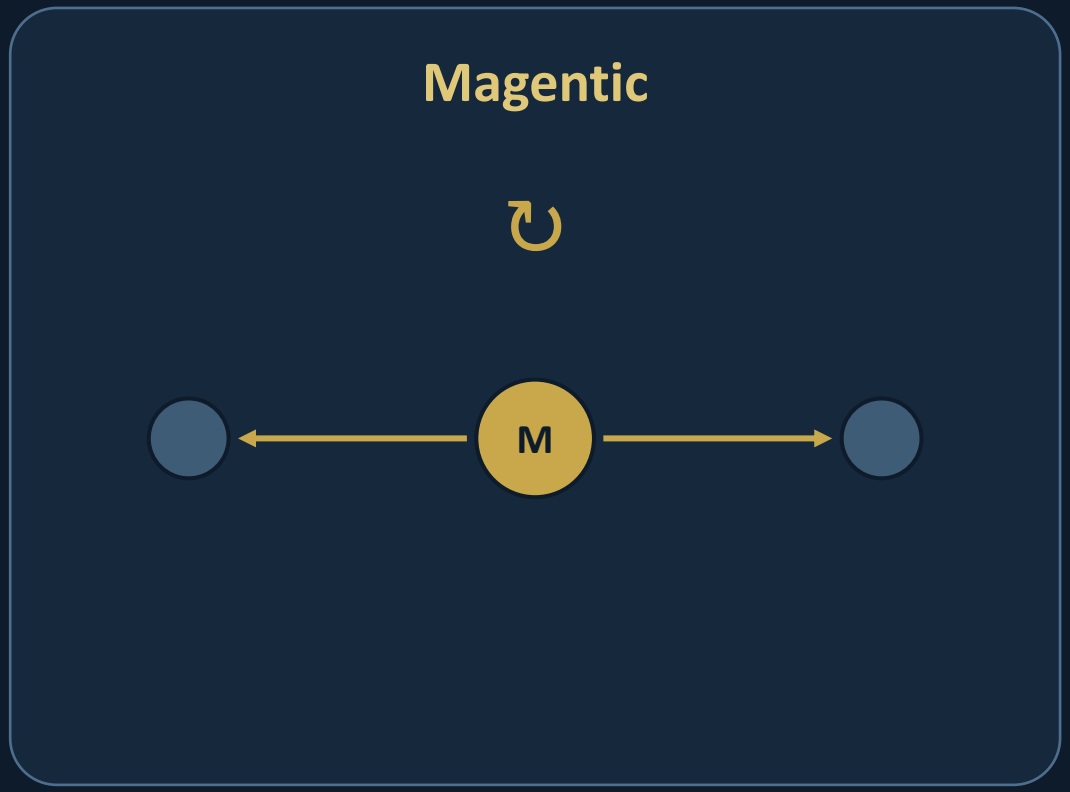
Wann nutzen

Iterative Verfeinerung im Team mit gemeinsamem Kontext.

Kosten / Hinweis

Zentrale Steuerung, mehr Runden bedeuten mehr Calls.

Magentic



Was es ist

Ein Manager plant open-ended Aufgaben dynamisch und wählt den nächsten Agenten je nach Fortschritt.

Wann nutzen

Komplexe, offene Aufgaben, die dynamische Zusammenarbeit erfordern.

Kosten / Hinweis

Genau diese open-ended, dynamische Zusammenarbeit über viele Runden macht die Kosten am schwersten vorhersagbar. Sparsam einsetzen.

Welche Orchestrierung wann?

- Sequential und Handoff akkumulieren Kosten pro Schritt, jeder Agent ist ein LLM-Call
- Concurrent erhöht den Durchsatz, kann Kosten gleichzeitig hochtreiben
- Magentic ist am flexibelsten und am schwersten zu budgetieren
- **Allen gemeinsam:** alle bieten Streaming, Checkpointing, Human-in-the-Loop und Pause/Resume

💡 ***Merksatz: im Zweifel die einfachste Orchestrierung, die den Fall löst.***

Abwägungen und Grenzen

Was in der Praxis schiefgeht

- **Halluzinationen:** erfundene Fakten oder Tool-Argumente, die echte Aktionen auslösen
- **Tool Failures:** externe API nicht erreichbar, der Agent muss den unhappy path kennen (unerreichbar, fehlerhaft, timeouts, …)
- **Timeouts und Latenz:** Agent-Ketten summieren Wartezeiten über mehrere Calls
- **Endlosschleifen:** ohne Stopp-Bedingungen verbrennt ein Agent Tokens und Geld
- **Inkonsistenter State:** über viele Turns kann der Kontext widersprüchlich werden
- **Prompt Injection:** versteckte Anweisungen in Eingaben, Dokumenten oder Tool-Ausgaben kapern Ziel oder Tool-Calls
- **Kein Rate Limiting:** Angriffsvektor, der Kosten verursacht
- **Stille Fehler:** anders als klassische Software fallen Fehler oft erst auf, wenn sie geschäftliche Folgen haben

👉 das sind alles Szenarien, die bei der Frage “wie viel KI” berücksichtigt werden müssen

Quellen: MAST-Taxonomie (Cemri et al., NeurIPS 2025); OWASP Agentic 2026; Microsoft Security Research

Testen trotz Nicht-Determinismus

Deterministisch unit-testen

State-Handling, Tool-Implementierungen und API-Bindings. Das ist klassischer Code.

Eval-basiert testen

Den KI-Output mit Scoring-Funktionen bewerten, der Test besteht ab einem Schwellwert.

- **Methoden:** LLM-as-a-Judge, regelbasierte Output-Checks (deterministisch, CI/CD-tauglich, z.B. valides JSON), Trajektorien prüfen
- **Auditierbarkeit:** beim Agent zusätzlich Reasoning, Memory und Tool-Interaktionen erklären

Wie teste ich einen Agenten?

Ein Agent antwortet nicht jedes Mal gleich. Ein einzelner Testlauf sagt also wenig aus.

Wie oft kommt ein gutes Ergebnis?

Denselben Test viele Male laufen lassen und auf die Trefferquote schauen, nicht auf den einen Glückslauf.

Hat sich der Agent richtig verhalten?

Hat er die richtigen Werkzeuge benutzt, in der richtigen Reihenfolge?

Womit?

- Microsoft bringt etwas Eingebautes mit
- Aus der Community: AgentEval von Jose Luis Latorre, .NET-nativ, noch preview

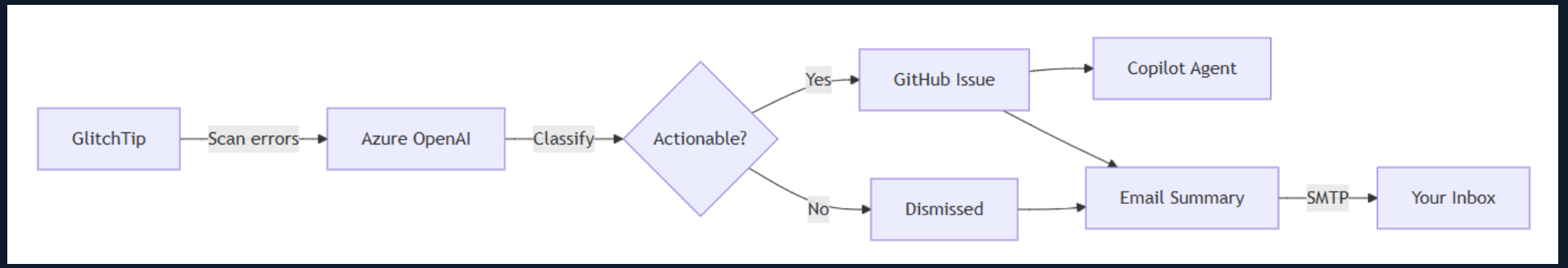
💡 ***Auch nicht-deterministisches Verhalten lässt sich testen, man misst die Quote statt eines einzelnen Ergebnisses.***

Quellen: Microsoft Learn, Agent Framework Evaluation; github.com/AgentEvalHQ/AgentEval

Beispiele aus der Praxis

GlitchPilot: wann ein einfacher Call reicht

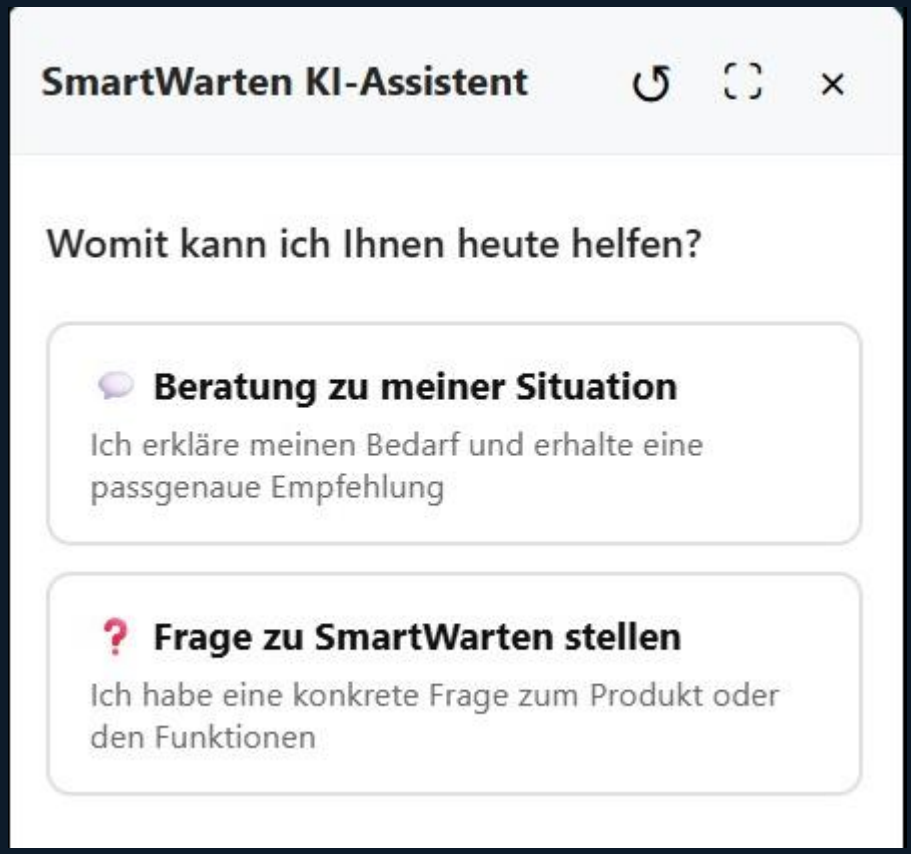
Fester Workflow, genau eine KI-Entscheidung



- Reales Projekt: nur ein Azure-OpenAI-Call mit einer Klassifikation, kein MAF, kein Agent
- **Bewusst kein autonomer Agent:** Ich will Kontrolle über nächtliche Repo-Änderungen
- Die Lektion: eine einzige Entscheidung braucht kein Framework, der einfache Weg war richtig

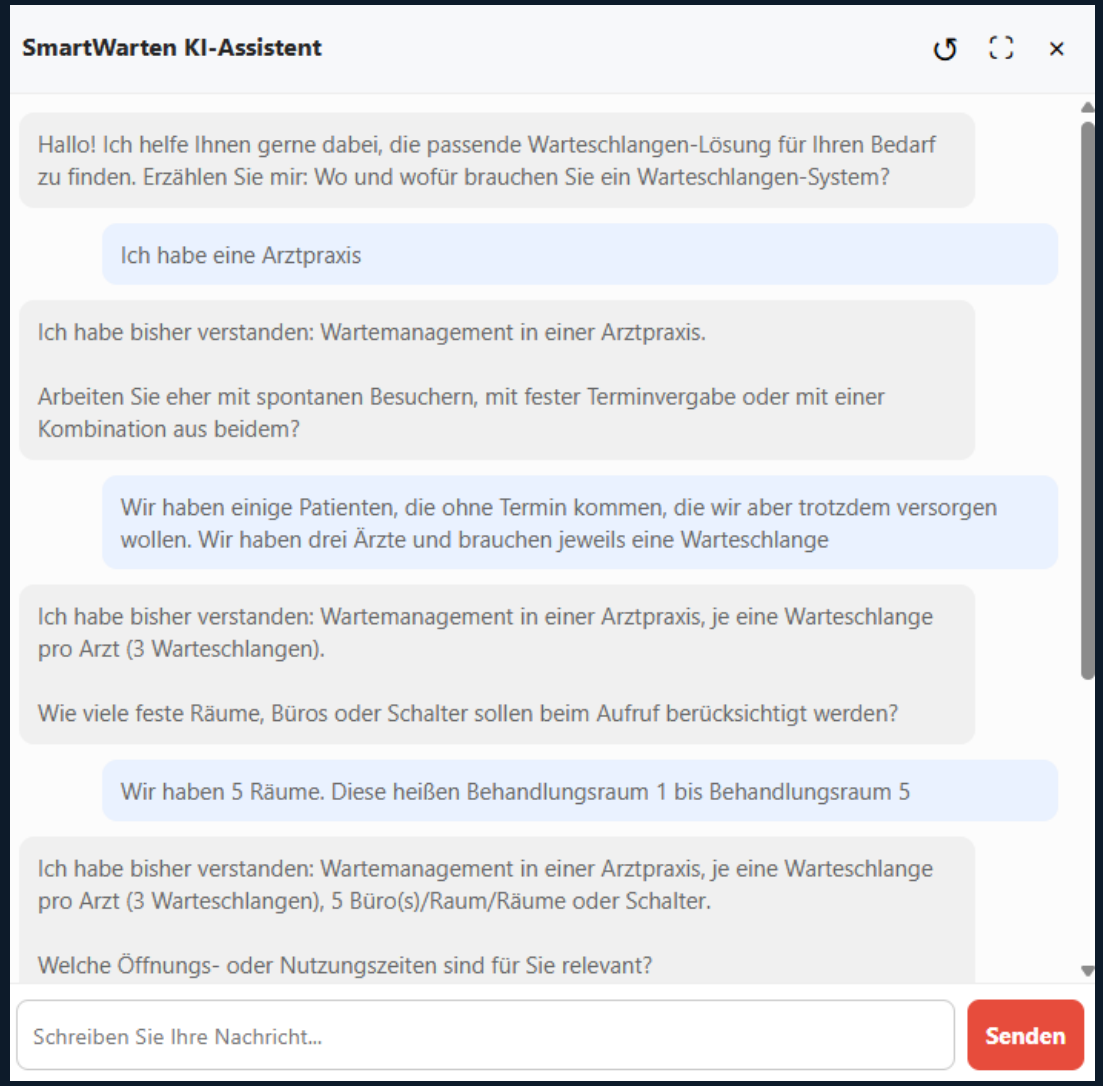
SmartWarten: Digitale Warteschlange

- Digitale Warteschlange als SaaS-Produkt
- **Chatbot mit Microsoft Agent Framework für Erstgespräch und strukturierte Datenextraktion zur Beratung vor Demo-Self-Signup**



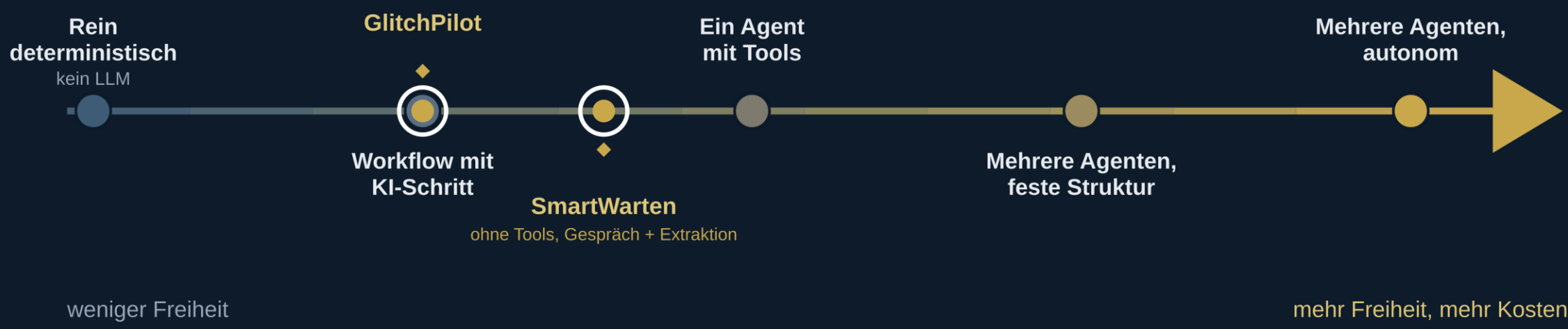
◆ BEISPIELE

SmartWarten: Beratungschatbot mit MAF



- Beratungs-Chatbot für mein SaaS, erfragt strukturiert nach der Wartesituation, d.h. Warteschlangen, Räume, Standorte
- **Der Verlauf ist nicht fest:** der Agent entscheidet je nach Antwort, was noch fehlt
- **Dokumentensuche ohne RAG:** rund 30 Seiten passen in den Context
- Validierung und Preisberechnung macht nicht die KI, sondern deterministischer, unit-testbarer Code
- **Status:** in Evaluierung, ob Eigenbau vs. third-party-Chatbot

Die Beispiele auf der Achse



Beide liegen bewusst links, nicht am autonomen Ende. GlitchPilot ist ein Workflow mit KI-Schritt, der SmartWarten-ChatBot ein Agent für Gespräch und Extraktion, ohne Tools.

Fazit

◆ ABSCHLUSS

💡 Drei Dinge zum Mitnehmen

1 Deterministisch oder KI ist eine Entscheidung pro Schritt, nicht global

2 MAF ist ein gutes Werkzeug für die KI-Teile, wenn man sie überhaupt braucht

3 Etablierte Software-Engineering-Prinzipien gelten weiter, KI ersetzt sie nicht

◆ ABSCHLUSS

Realismus

30–50%

der GenAI-Projekte werden laut Gartner nach dem Proof of Concept aufgegeben

70%

der GenAI-Projekte erreichen laut Deloitte production nicht

- Die meisten KI-Projekte scheitern nicht an der Technik
- **Gründe:** unklarer Wert, Datenqualität, eskalierende Kosten, fehlende Risikokontrollen
- **Empfehlung:** klein anfangen, weit links auf der Achse

◆ DIE KERNBOTSCHAFT

Die spannende Frage ist nicht, ob KI Entscheidungen treffen kann.

Sondern welche Entscheidungen wir ihr bewusst erlauben sollten.

Nicht das Maximum an Autonomie gewinnt, sondern der klar abgegrenzte, wartbare Einsatz.

Vielen Dank

Kontakt LinkedIn: Dr. Matthias Liebeck

Newsletter ghcp.liebeck.io

LinkedIn

GitHub Copilot Newsletter