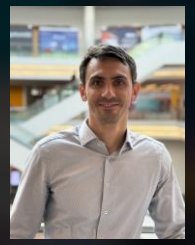
THINKWISE

empower business

Legacy neu denken: Prompt to Model



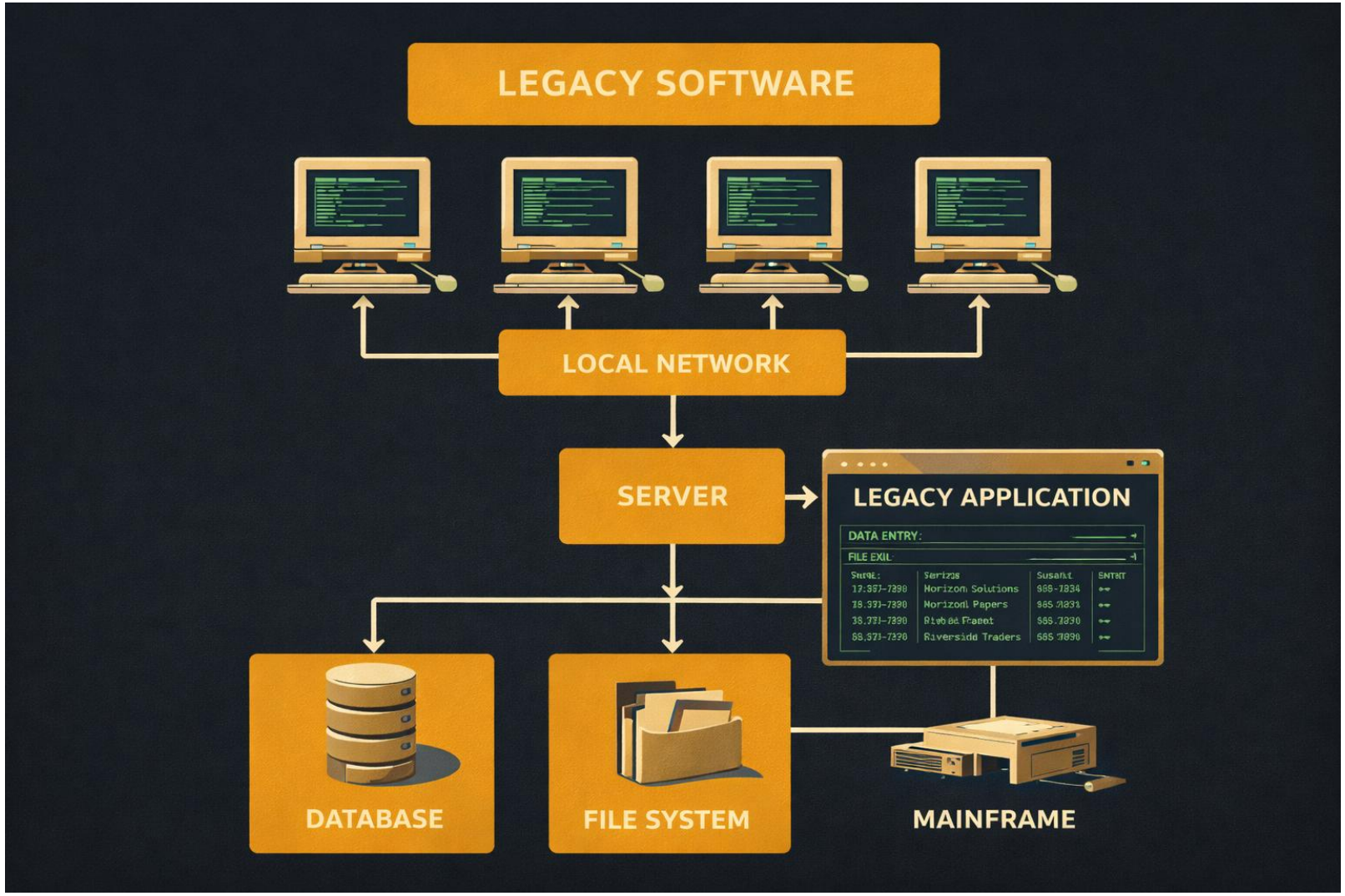
David Baier
Lead Pre-Sales Consultant DACH

Büros in den 1990er

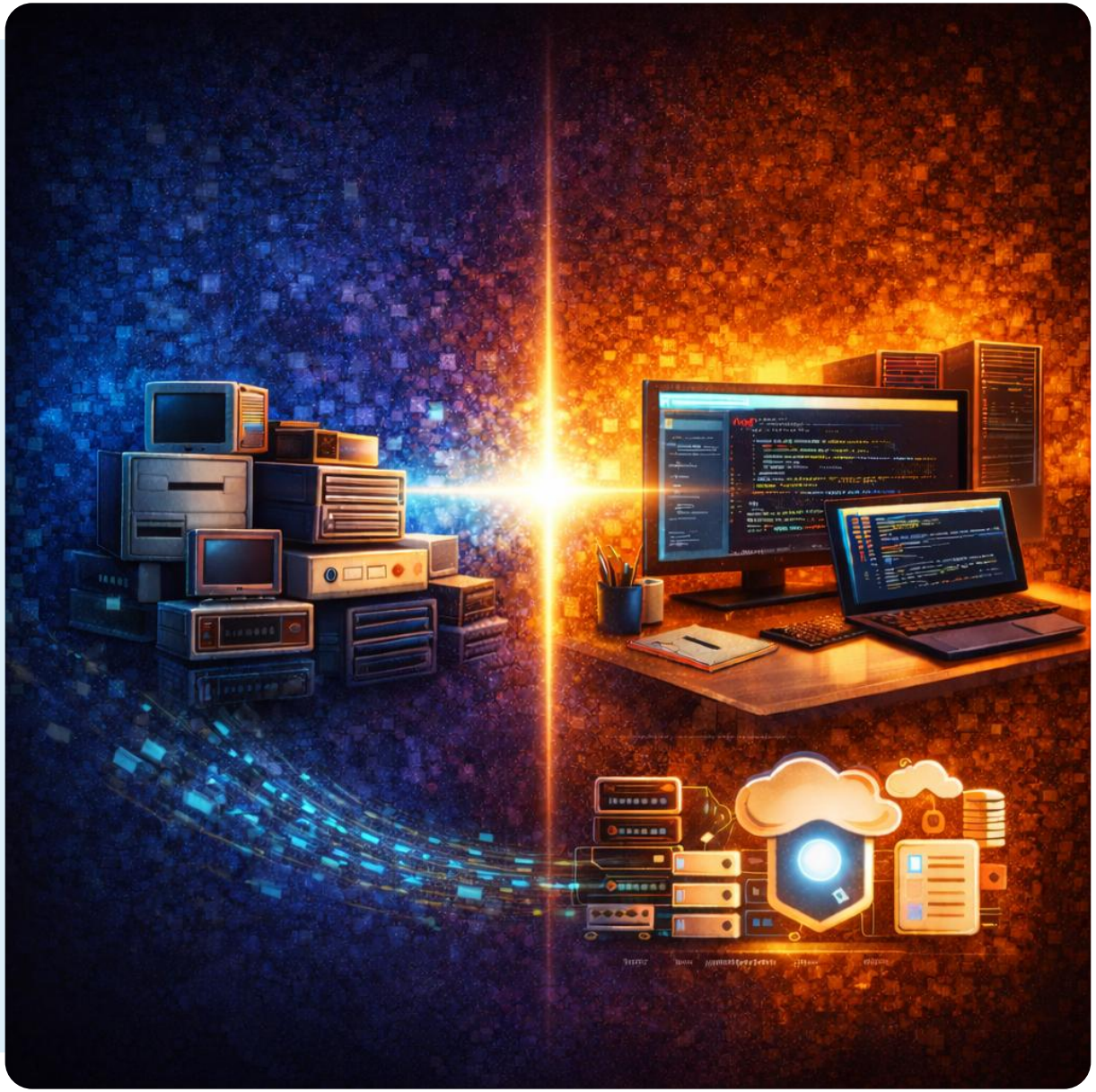
Büros in 2026

Legacy Software ist Software, die noch genutzt wird, aber technisch und organisatorisch „aus der Zeit gefallen“ ist.

Klassische Legacy-Architektur



Mission: From Legacy to Future



Mission: Warum Legacy Software modernisieren?

Man ersetzt Legacy Software, weil sie langfristig:

- ✅ **Risiko reduziert** (Security, Know-how, Ausfälle)
- ✅ **Kosten senkt** (Betrieb, Wartung, Change)
- ✅ **Innovation ermöglicht** (Speed, Integration, Cloud, UX)
- ✅ **Zukunftssicherheit schafft**

Strategien: From Legacy to Future

Rehosting

Was: Legacy-Anwendung wird nahezu unverändert auf neue Infrastruktur verschoben (z. B. Cloud, neue VMs, Container).

Ziel: Schnelle Modernisierung der Plattform ohne große Codeänderungen.

Vorteile

- schnell umsetzbar
- geringes Risiko für Fachlichkeit
- reduziert Infrastruktur-/Betriebsprobleme

Nachteile

- technische Schulden bleiben
- Architektur bleibt legacy
- wenig Mehrwert für Business

Geeignet wenn: Betrieb/Hardware veraltet, aber Anwendung stabil.

Replatforming

Was: Moderate Anpassungen, um die Anwendung besser auf moderner Infrastruktur zu betreiben (z. B. DB-Wechsel, Runtime-Update, Containerisierung).

Vorteile

- bessere Performance/Skalierung
- modernere Betriebsfähigkeit (DevOps, Monitoring)
- oft überschaubarer Aufwand

Nachteile

- Risiko bei Abhängigkeiten/Kompatibilität
- Fachlichkeit bleibt meist unverändert

Geeignet wenn: Technologie-Stack ist alt, aber noch migrierbar.

Refactoring

Was: Der Code wird schrittweise verbessert: Modularisierung, Testabdeckung, Entkopplung, APIs, Clean Architecture.

Ziel: Wartbarkeit, Stabilität und Erweiterbarkeit steigern.

Vorteile

- nachhaltige Qualitätssteigerung
- keine „Big Bang“-Ablösung
- Business kann parallel weiterarbeiten

Nachteile

- dauert länger
- braucht Disziplin, Tests, Architektur-Guardrails
- schwieriger ROI zu erklären („unsichtbare Arbeit“)

Geeignet wenn: Anwendung geschäftskritisch ist und kontinuierlich weiterentwickelt werden muss.

Rearchitecting

Was: Grundlegende Umgestaltung der Architektur – z. B. Monolith → modulare Architektur oder Microservices.

Vorteile

- Skalierbarkeit und Flexibilität
- bessere Team- und Deployment-Entkopplung
- Cloud-native möglich

Nachteile

- hoher Aufwand & Komplexität
- erfordert hohe Architektur- und DevOps-Reife
- Risiko von Overengineering

Geeignet wenn: viele Teams, hohe Änderungsfrequenz, Skalierungsprobleme.

Replacement

Was: Legacy wird durch Standardsoftware/SaaS ersetzt (ERP, CRM, DMS etc.).

Vorteile

- schnellere Modernisierung
- Wartung/Updates extern
- Fokus auf Kernkompetenzen

Nachteile

- Anpassungsdruck auf Prozesse
- Lock-in
- Integrationsaufwand

Geeignet wenn: Legacy bildet keine Differenzierung/Kernleistung.

Rebuilding

Was: Anwendung wird neu entwickelt (ggf. auf Basis neuer Technologien und neuer UX).

Vorteile

- saubere Basis ohne Altlasten
- bessere UX / Prozesse möglich
- moderne Security & Architektur by design

Nachteile

- sehr teuer und riskant
- lange Time-to-Value
- Gefahr: Fachlogik wird falsch nachgebaut

Geeignet wenn: Legacy nicht mehr wartbar ist oder Technologie/Skills fehlen.

REBUILDING

THINKWISE

Rebuilding - Varianten

A) 1:1 Rebuild (funktional identisch)

- Fokus: technische Erneuerung
- weniger Prozessänderungen
- oft schneller abnehmbar

➡️ gut, wenn Business stabil, aber Technik tot

B) Rebuild + Prozessoptimierung

- fachliche Neugestaltung
- bessere UX
- neue Workflows

➡️ hoher Business-Wert, aber mehr Risiko/Change Management

C) Rebuild als Produktplattform

- neue Plattform, modulare Architektur
- evtl. Mandantenfähigkeit, APIs, App-Ökosystem

➡️ strategisch, aber anspruchsvoll

Rebuilding - Vorgehen

Phase 1: Assessment & Scoping

- Welche Module/Prozesse sind wirklich relevant?
- Welche Teile können weg?
- Welche Quick Wins gibt es?

Phase 2: Domänenschnitt + Zielarchitektur

- Domänen definieren (DDD)
- Ownership klären
- Datenmodellstrategie festlegen

Phase 3: MVP & Pilotdomäne

- klein starten
- echte Nutzer
- früh integrieren

Phase 4: Inkrementeller Ausbau

- Domäne für Domäne ersetzen
- parallel betreiben
- Altkomponenten abschalten

Rebuilding - Datenstrategien

Option 1: Big Bang Migration

- einmalige Migration
- Cutover am Stichtag

➡️ nur wenn Daten sauber & System überschaubar

Option 2: Parallelbetrieb + Synchronisation

- Legacy und Neu laufen parallel
- Daten werden synchron gehalten

➡️ robust, aber technisch anspruchsvoll

Option 3: CDC (Change Data Capture)

- Änderungen aus Legacy-DB werden „gestreamt“
- Neu-System wird kontinuierlich aktualisiert

➡️ sehr modern, gut skalierbar

Rebuilding: Umsetzungsansätze

Rebuilding – Classic Dev

Was ist das?
Die Anwendung wird komplett neu entwickelt – typischerweise mit einem modernen Tech-Stack (z. B. Java/.NET/Node + React/Angular + Cloud).

Typische Merkmale

- volle Freiheit bei Architektur & UX
- Microservices / modulare Monolithen möglich
- API-first / Event-driven möglich
- maximale Anpassbarkeit

Geeignet wenn:
Die Software ist Kern des Geschäfts / Differenzierung und extrem individuell.

Rebuilding – Classic Dev

Vorteile

✅ höchste Flexibilität

✅ keine Plattformabhängigkeit

✅ sehr gut für komplexe Individualprozesse

✅ sehr skalierbar

Nachteile

❌ hoher Aufwand / lange Laufzeit

❌ hoher Bedarf an Experten (Architektur, DevOps, Security)

❌ Wartung bleibt vollständig beim Unternehmen

Rebuilding – Low-Code

Was ist das?
Die Anwendung wird neu gebaut, aber auf Basis einer Plattform, die Datenmodell, UI, Logik, Security und Deployment stark beschleunigt.

Typische Merkmale

- modellgetriebene Entwicklung (Domain Model, UI, Workflows)
- schneller MVP / schnelle Iteration
- integrierte Standards (Security, CRUD, Rollen, UI Patterns)
- häufig bessere Wartbarkeit durch Struktur & Konventionen

Geeignet wenn:
Viele Prozesse datengetrieben sind (ERP/Backoffice/Industrie), hoher Change-Need, begrenzte Dev-Kapazität.

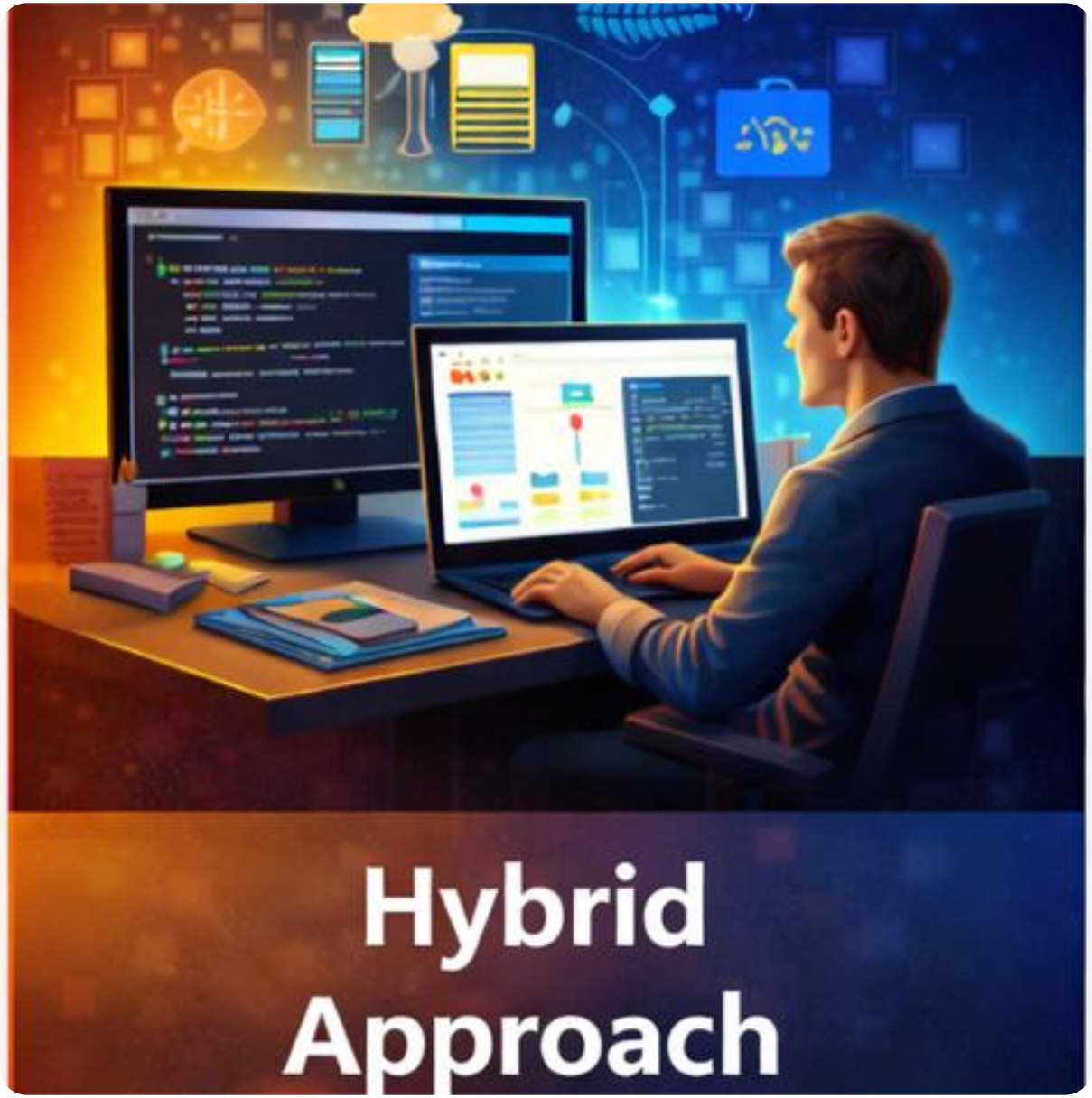
Rebuilding – Low-Code

Vorteile

- ✅ deutlich schneller als Classic Dev
- ✅ geringere Komplexität im Alltag
- ✅ weniger „Boilerplate“-Code
- ✅ schneller Business-Value
- ✅ gut für Standardprozesse + viele Datenmasken

Nachteile

- ❌ Plattform-Lock-in
- ❌ Grenzen bei extrem individuellen Anforderungen
- ❌ Spezialfälle brauchen oft Extensions/Custom Code
- ❌ Architektur wird durch Plattform geprägt



Rebuilding – Hybrid

A) Low-Code Core + Custom Extensions

- Standardfunktionen in Low-Code
- Speziallogik als Services (.NET/Java) via API
- ➡️ Best Practice für Industrie/ERP-nahe Anwendungen.

B) UI in Low-Code, Backend als Services

- Plattform liefert UI + schnelle Business-Logik
- Kernlogik in eigenem Domain Backend
- ➡️ gut, wenn Domain sehr komplex ist.

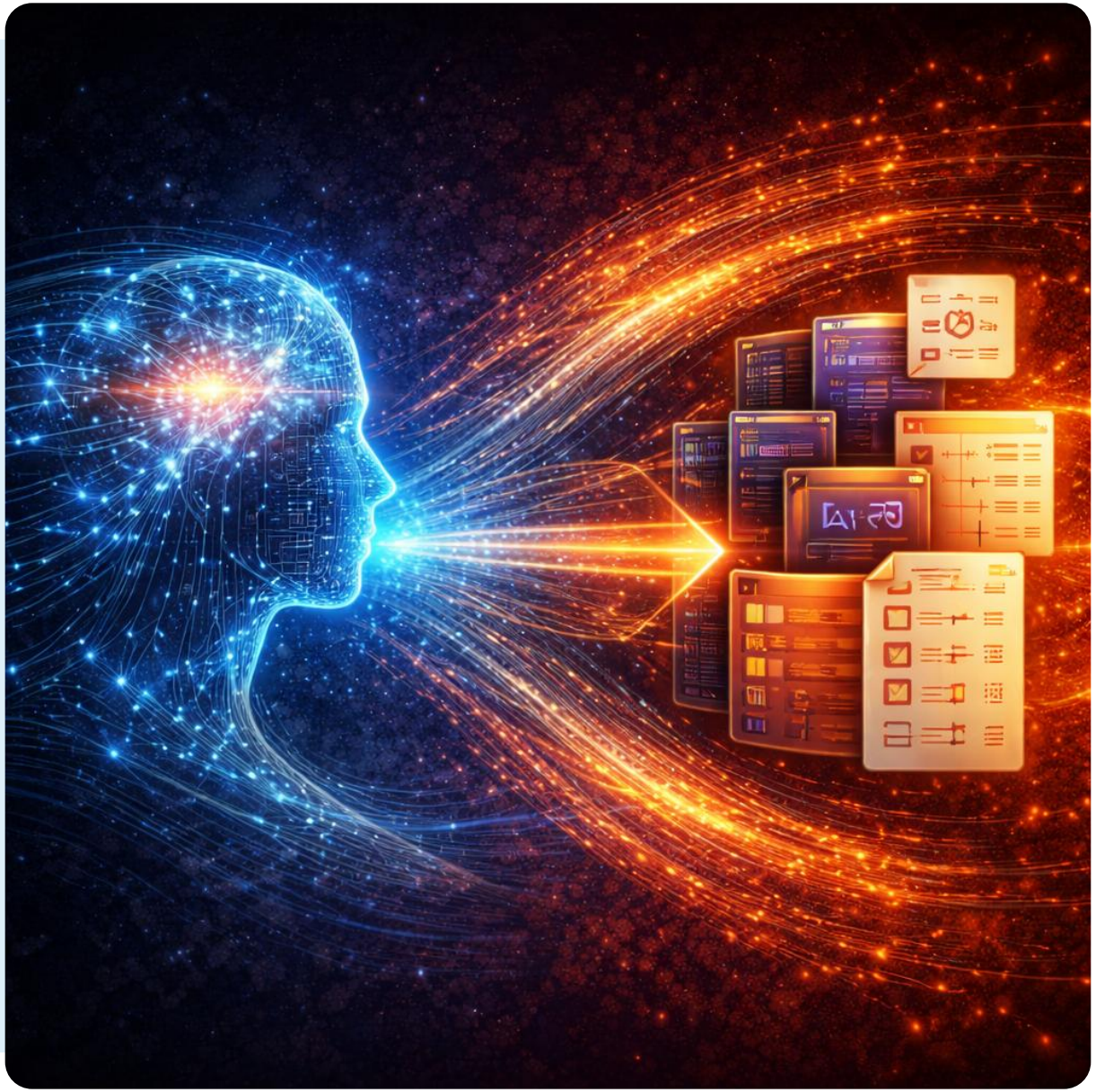
C) Rebuild als Plattform + Integration Layer

- neue Anwendung auf Low-Code
- Legacy/Standardsoftware via Integration (API/Events/ETL)
- ➡️ sehr häufig in Transformationsprojekten.

Rebuilding: Was ist der richtige Ansatz?

REBUILDING MIT KI

THINKWISE



Zielbild: Von implizitem Wissen zu expliziten Modellen

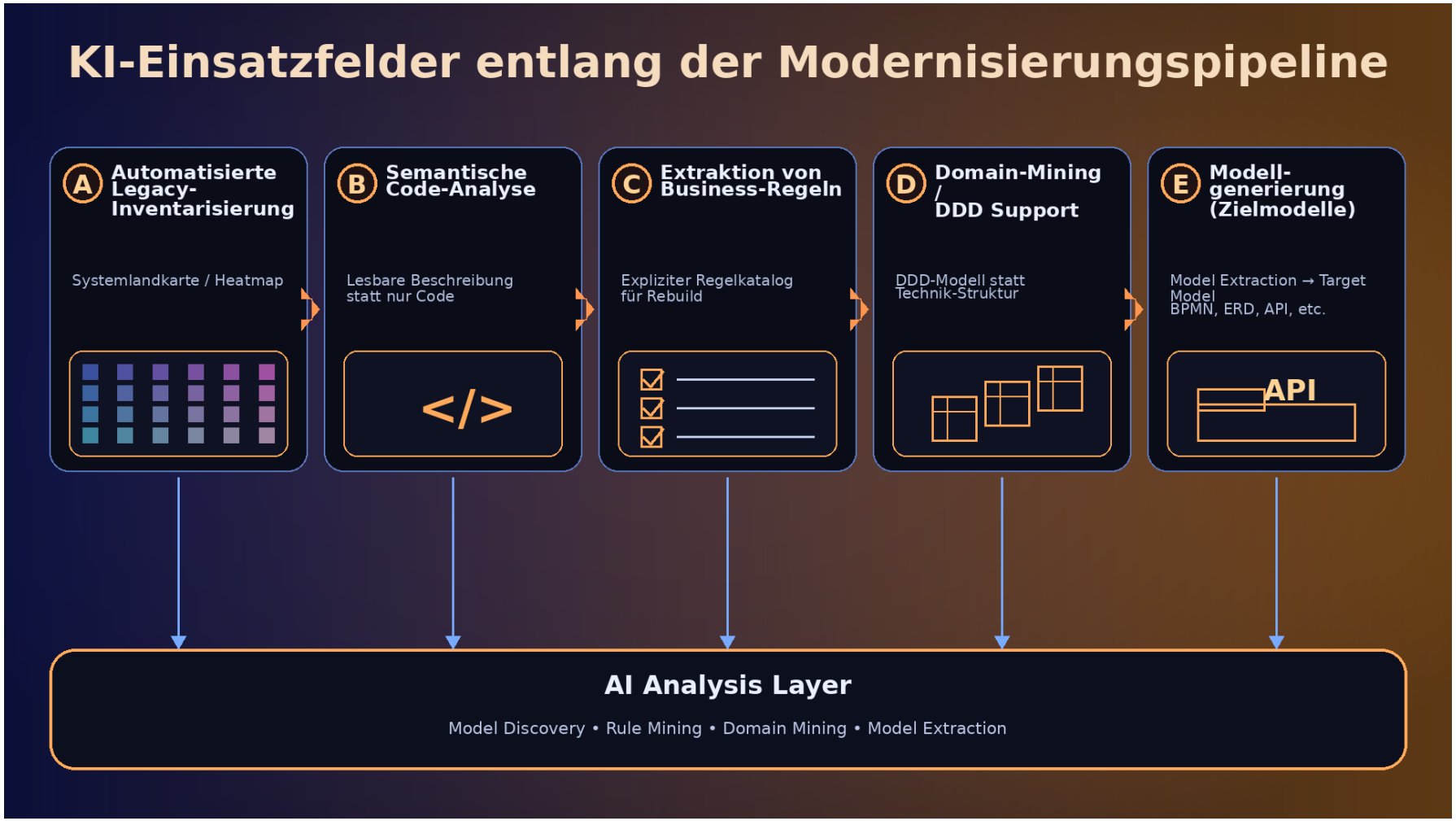
Legacy-Systeme enthalten Geschäftslogik oft in impliziter Form:

- Code (Spaghetti, Monolith, Stored Procedures)
- Datenbankstrukturen (Tabellen, Trigger)
- UI-Logik (Forms, Events)
- Konfigurationen, Batch-Jobs
- Wissen in Köpfen

KI hilft, daraus explizite Modelle zu bauen, z. B.:

- **Domänenmodell** (Entities, Value Objects, Aggregate)
- **Prozessmodell** (Workflows, States, Transitions)
- **Schnittstellenmodell** (APIs, Events, Integrationen)
- **Regelmodell** (Validierungen, Constraints, Policies)
- **Datenmodell** (normalisiert, fachlich benannt, mit Ownership)

KI-Einsatzfelder entlang der Modernisierungspipeline



A) Automatisierte Legacy-Inventarisierung (System Discovery)

KI kann technische Artefakte sammeln, klassifizieren und in Beziehung setzen:

Inputs

- Quellcode (z. B. RPG, COBOL, Java, VB6, PL/SQL)
- DB-Schema (DDL, ERDs)
- Logs / Traces
- UI-Screens (Screenshots, GUI-Definitionen)
- Tickets, Doku, Wiki, Emails

KI-Aufgaben

- Klassifikation: „Was ist UI? Was ist Businesslogik? Was ist Integration?“
- Clustering: Module, Komponenten, Subsysteme erkennen
- Dependency Graph: Abhängigkeiten zwischen Artefakten extrahieren

Ergebnis

➡️ Systemlandkarte / Heatmap („Wo steckt die Komplexität?“)

B) Semantische Code-Analyse (nicht nur Syntax)

B) Semantische Code-Analyse (nicht nur Syntax)

- Klassische Tools liefern AST/Callgraphs – KI liefert *Bedeutung*:

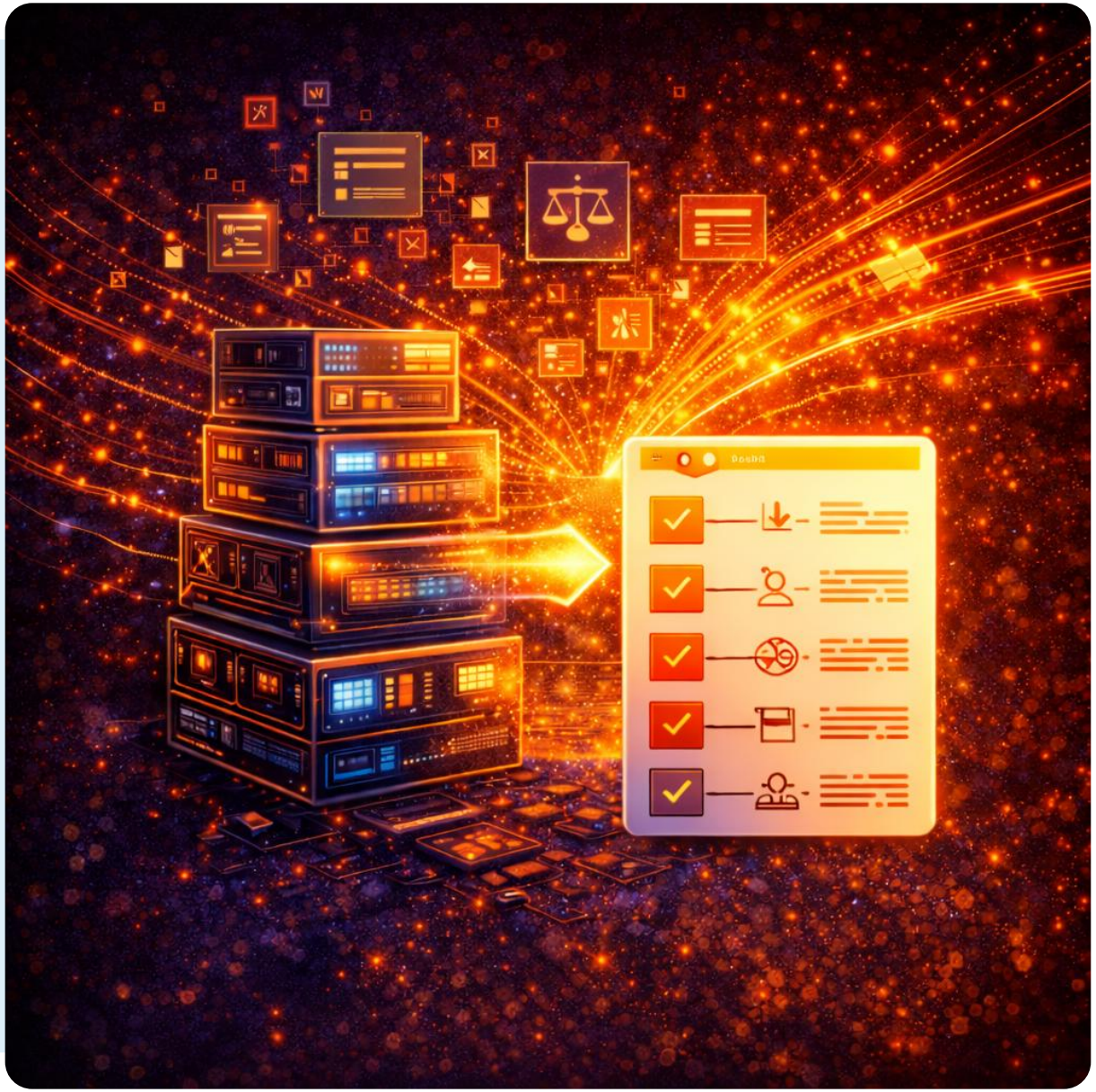
Beispiele

- „Diese 2000-Zeilen-Prozedur enthält Pricing-Logik“
- „Dieses Modul ist Kundenstammdaten“
- „Diese Tabellen gehören fachlich zusammen“

Methoden

- Embeddings + Similarity Search („ähnliche Funktionen finden“)
- LLM-gestützte Zusammenfassungen auf Funktions-/Modul-Ebene
- Pattern Detection (z. B. CRUD, Workflow, Berechnung, Validation)

Ergebnis
➡️ Lesbare fachliche Beschreibung statt nur Code



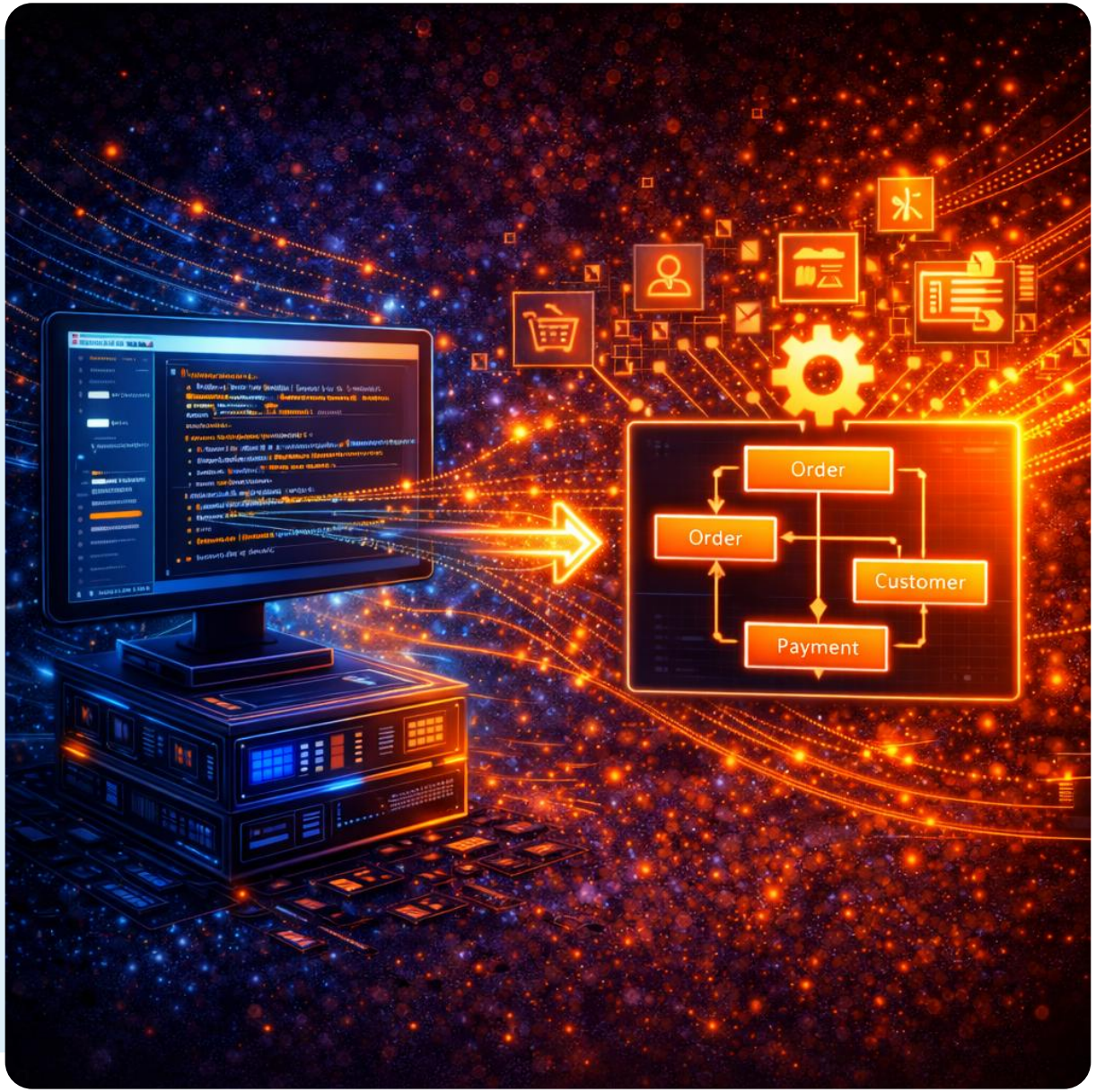
C) Extraktion von Business-Regeln (Rule Mining)

Legacy-Systeme sind oft „regelgetrieben“, aber die Regeln sind verteilt.

KI kann:

- Validierungen und Constraints erkennen
- Entscheidungslogik extrahieren („if/else“, case statements)
- Berechnungslogik in verständliche Regeln übersetzen
- Regeln in Formate überführen:
  - Decision Tables
  - DMN
  - Policies / Rule Catalog

Ergebnis
➡️ expliziter Regelkatalog als Grundlage für Rebuild



D) Domänenmodell-Ableitung (Domain Mining / DDD Support)

KI unterstützt die Ableitung von Bounded Contexts und Domänenobjekten:

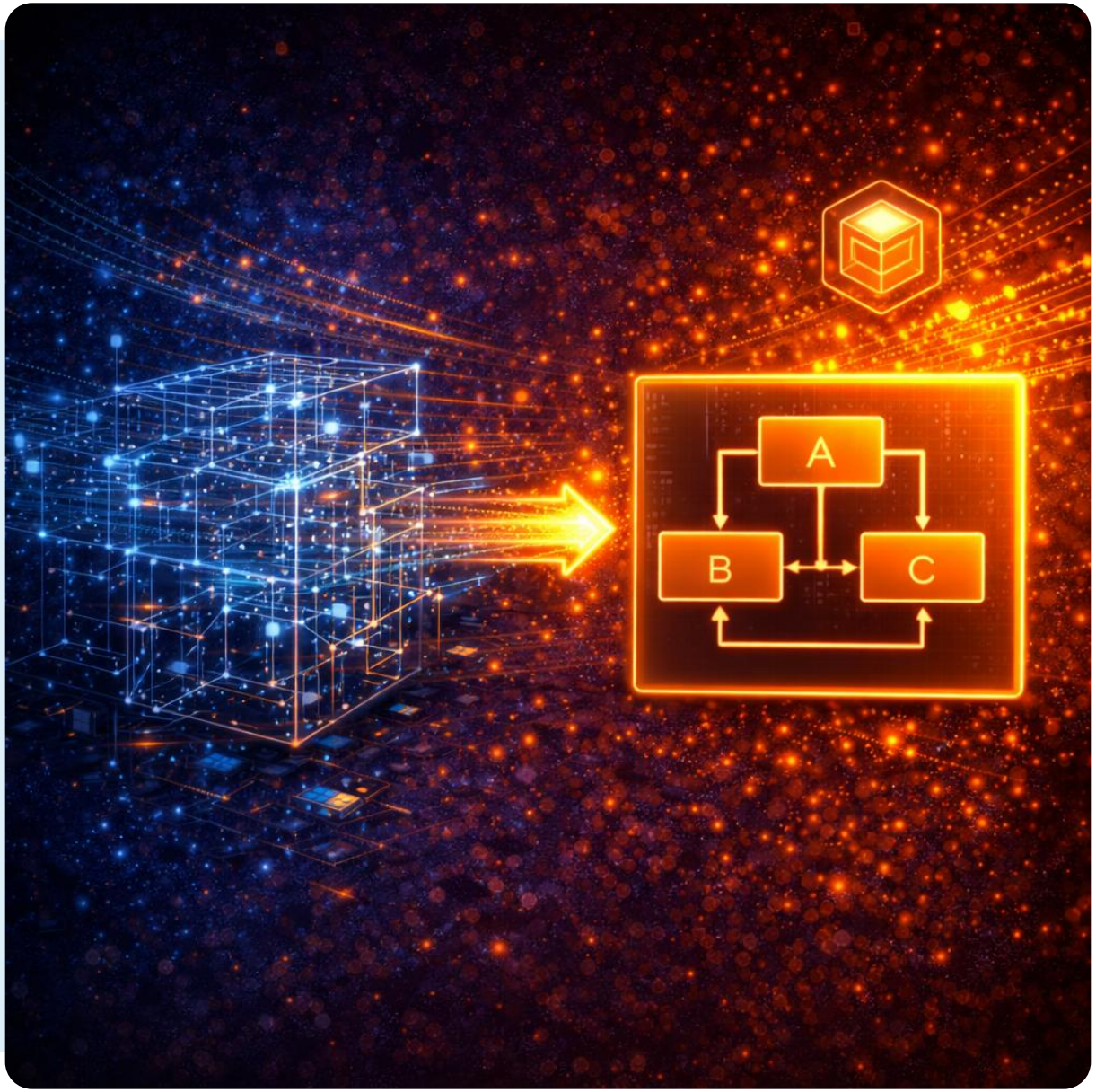
Technik

- Analyse von Namensräumen, Tabellen, Funktionen, UI-Gruppen
- Erkennung fachlicher Cluster
- Vorschläge für Domänenschnitt (z. B. Order, Inventory, Billing)

Outputs

- Entity-Kandidaten + Attribute
- Beziehungen (1:n, n:m) + fachliche Begründung
- Kontextkarten (Context Map)
- Ubiquitous Language (Begriffe + Synonyme)

Ergebnis
➡️ DDD-Modell statt „Technik-Struktur“



E) Modellgenerierung (Model Extraction → Target Model)

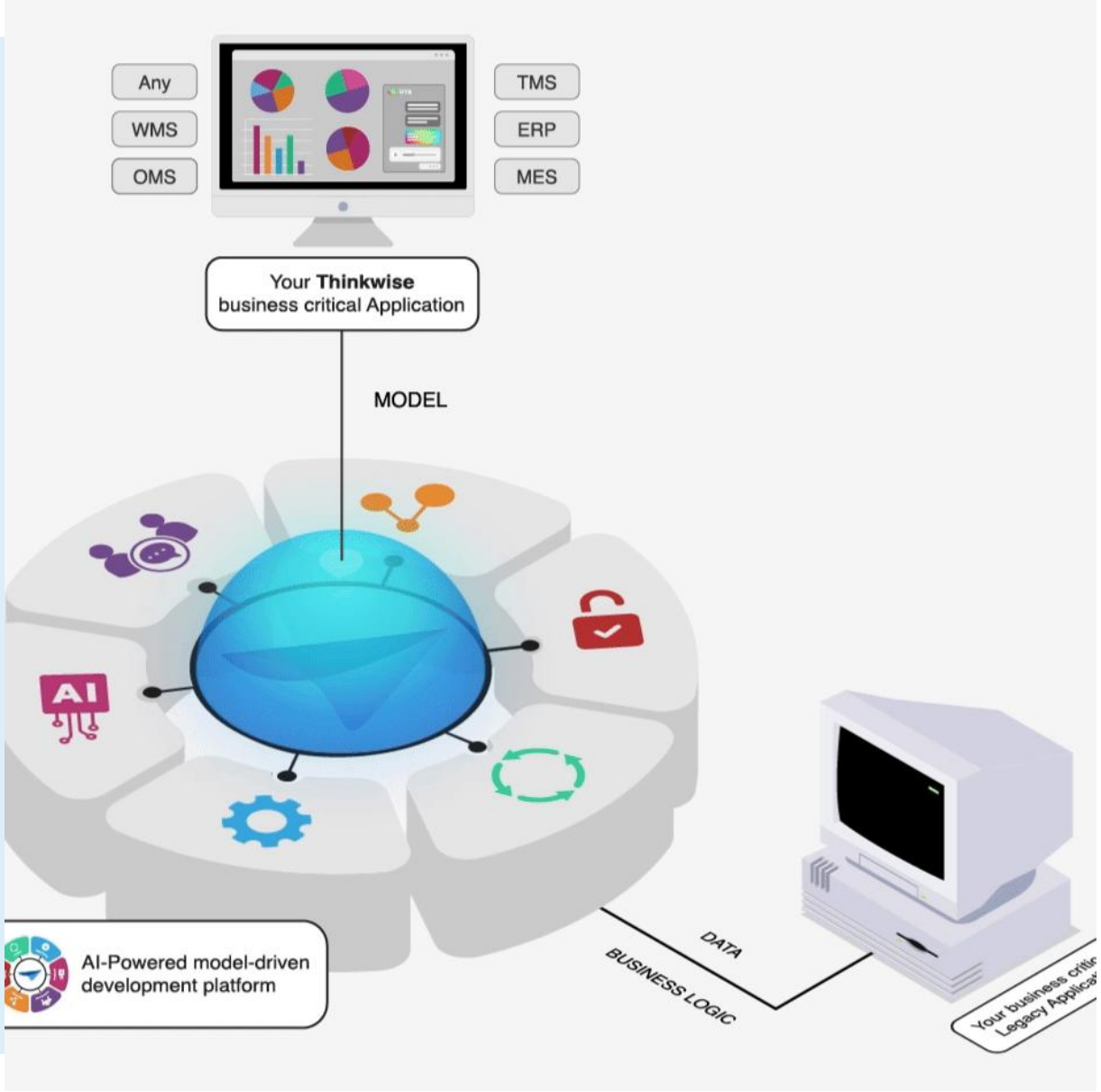
Wenn ein Zielmodell definiert ist (z. B. Metamodell einer Plattform, UML, BPMN, ERD), kann KI Artefakte direkt generieren:

- ER-Modelle / Datenmodelle
- API-Spezifikationen (OpenAPI)
- Event-Kataloge (Event Storming Output)
- BPMN / Prozessdiagramme
- UI-Strukturmodelle (Menü, Screens, Rollen)

Ergebnis
➡️ KI als „Übersetzer“ von Legacy-Artefakten in moderne Modellartefakte

REBUILDING MIT THINKWISE

THINKWISE



Rebuilding mit Thinkwise

- **Modellgetriebene Low-Code-Plattform**

- **Thinkwise Upcycler**:

Der **Thinkwise Upcycler** unterstützt dabei, bestehende Legacy-Anwendungen **automatisiert zu analysieren und in ein modernes Modell zu überführen** (z. B. aus bestehenden Datenbankstrukturen / Applikationslogik / UI-Strukturen – je nach Quelle und Projektsetup).

➡️ **Prompt to Model** für die Transformation von Legacy Code

- **Technology-as-a-Service:**

Technology as a Service bedeutet, dass Thinkwise nicht nur eine Low-Code-Plattform liefert, sondern den kompletten Technologie-Stack als **Service** bereitstellt

Rebuilding mit Thinkwise

THINKWISE

Vielen Dank!

Interesse geweckt?
www.thinkwise.de