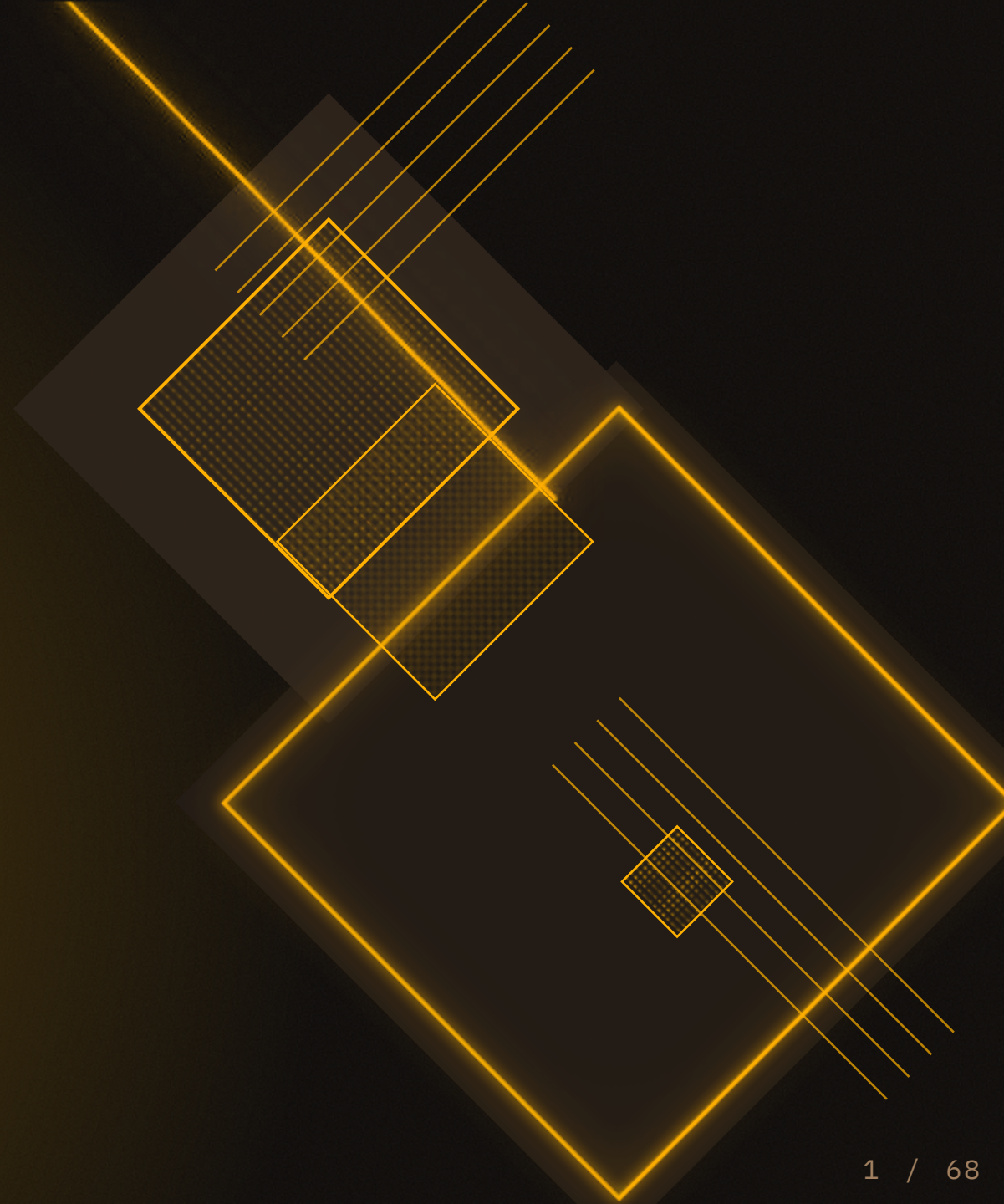
martin boßlet



# TEXT-FIRST

## MAKE AI GREAT ALREADY.

# HABEN SIE SCHON VON CHATGPT GEHÖRT?

Just kidding...

# 01

# Jeder redet über KI. Nicht jeder ist zufrieden.

Warum gehen die Erfahrungen so weit auseinander?

# ZWEI LAGER, ZWEI REALITÄTEN

**KI-PROPHETEN**

- "10× produktiver"
- "KI für alles"
- "Konkurrenz längst abgehängt"
- "Junior-Devs überflüssig"

**KI-VERWEIGERER**

- "Halluziniert ständig"
- "Macht mehr Arbeit als es spart"
- "Ist nur Blockchain reloaded"
- "Das ist nichts für mich"

Von Euphorie bis Hass - man findet die komplette Bandbreite.

# DAS 70 %-PROBLEM

> "They can get 70% of the way there surprisingly quickly, but that final 30% becomes an exercise in diminishing returns."
> -Addy Osmani

**START**
**70%**
KI bringt erstaunlich schnell einen plausiblen ersten Stand.

**REST**
**30%**
Die letzten Details entscheiden, ob daraus produktiver Code wird.

**KOSTEN**
**?**
Wenn die Nacharbeit explodiert, verschwindet der Produktivitätsgewinn.

KI wird **wirtschaftlich**, wenn die letzten 30 % günstiger werden.

Nicht das *Erzeugen* ist teuer - das *Nacharbeiten* ist es.

# DER EIGENTLICHE ENGPASS

Nicht das Modell

Nicht das "Tool du jour"

Sondern:

## Passgenauer Kontext

Verstreut in:

- Tickets, Slack, E-Mails
- PDF, PowerPoints, Word-Dokus, Visio-Diagramme
- Confluence-Seiten, die niemand pflegt
- Köpfen von Kollegen, die längst weg sind

# CONTEXT IS KING

**Garbage in→ Garbage out.**

Ohne Kontext rät die KI.
Faulheit wird bestraft.
Mit Nacharbeit.

Mit präzisem Kontext: bessere Ergebnisse, schneller, günstiger.

# MEHR KONTEXT IST NICHT AUTOMATISCH BESSER

Größere Kontextfenster sind nicht immer die Lösung:
zu viel Kontext verwirrt - und kostet Tokens.

ZU WENIG KONTEXT

**Raten**

Das Modell füllt Lücken mit plausiblen Annahmen.

ZU VIEL KONTEXT

**Rauschen**

Irrelevante Tokens verstopfen das Fenster.

GENAU PASSEND

**Fokus**

Gezielter Kontext als Leitplanke.

Context Engineering bleibt ein Thema.

# DIE FRAGE DES VORTRAGS

Wie geben wir KI sinnvollen Kontext -
nicht nur einmal, sondern dauerhaft?

Spoiler: Es geht nicht um bessere Prompts.
Es geht um die älteste Technologie der Menschheit.

# 02

# Sprache.

Disruptive Innovationen sind oft auch Sprachevolution.

KI ist keine Ausnahme.

# IT-EVOLUTION DURCH SPRACHE

| | |
|---|---|
| 1940er | Maschinencode (0/1) |
| 1950er | Assembler |
| 1970er | C, ASCII |
| 1990er | High-Level-Sprachen, HTTP, HTML |
| 2000er | Markdown, JSON, YAML |
| 2020er | Natürliche Sprache → KI-gestützte Entwicklung |

Evolution durch Abstraktion und Annäherung an **menschliche Sprache**

# DIE UNIVERSELLE SCHNITTSTELLE: TEXT

**MENSCH**

- Liest Text
- Schreibt Text
- Denkt in Sprache
- Versteht Struktur

**KI**

- Trainiert auf Text
- Generiert Text
- "Denkt" in Tokens
- Versteht Struktur

Text ist das einzige Format,
das **Mensch & Maschine** perfekt beherrschen

# MARKDOWN IST TEXT++

- **Einfach** - lernbar in 10 Minuten
- **Lesbar** - auch ohne Tools
- **Diffbar** - schon Word in Git gedifft?
- **Strukturiert** - Überschriften, Listen, Code
- **Offen** - kein Vendor, kein Format-Lock-In
- **Universell** - GitHub, GitLab, Notion, Obsidian, ...
- **Zukunftssicher** - Open Source, ohne Firma im Hintergrund

Plain Text mit minimaler Strukturschicht - mehr braucht es nicht.

# MARKDOWN IST ÜBERALL

GITHUB / GITLAB
**Alles**
README, Issues, PRs, Wikis

REDDIT
**Kommentare**
Markdown für Markdown

JEKYLL / HUGO
**SSG**
Markdown für statische Websites

GITHUB COPILOT
**LLM**
Antworten in Markdown

CLAUDE
**LLM**
Antworten in Markdown

CHATGPT
**LLM**
Antworten in Markdown

OBSIDIAN
**Notes**
Markdown als Speicherformat

FOAM
**Notes**
"Zettelwirtschaft" in VS Code

ASTRO
**SSG**
Markdown & MDX für statische Websites

# DER WANDEL IST REAL

## Blogosphäre

- https://daringfireball.net/2004/03/dive_into_markdown
- https://matduggan.com/markdown-ate-the-world/
- https://rogerwong.me/2026/01/how-markdown-took-over-the-world
- https://michaelcraig.group/stories/markdown-eats-the-office/
- https://www.makeuseof.com/ditched-word-for-markdown-writing-setup/

## Konzerne entdecken Markdown

- Microsoft Markdown-Konverter
- Markdown in Google Docs
- Cloudflare - Markdown für Agenten
- Adobe Experience Manager

# Vom Nerd-Format zum Mainstream

Warum ausgerechnet Markdown?

Wieso gerade jetzt?

# MARKDOWN IST DIE MUTTERSPRACHE DER LLMS

Nicht .docx
Nicht .pdf
Nicht .vsdx
Auch nicht .txt

**Sondern: Markdown**

> Wer Markdown spricht, spricht die Sprache, in der die KI **denkt**.

# 03

# Die Lüge.

"Code ist Dokumentation." Nicht.

# "CODE IST DOKUMENTATION"

## Nein. Code ist Code.

Code zeigt das **Was**.

Nicht das **Warum**.

Nicht die verworfenen Alternativen.

Nicht den fachlichen Kontext.

Wir haben uns das schöngeredet,
weil echte Doku schlicht **zu teuer** war.

# DAS "SARAH"-PROBLEM

Jedes Team hat eine Sarah.

Sarah weiß, **warum** wir damals den Message Bus eingeführt haben.

Sarah kann COBOL. Als **Einzige** bei uns.

Sarah weiß vor allem, warum der verdammte Message Bus **immer noch** da ist.

**Sarah hat 2024 gekündigt.**

# WARUM DOKU SCHEITERT

### DIE ALTE RECHNUNG

- Schreiben: ~25 % der Entwicklungszeit
- Pflegen: nochmal 25 %
- Nach 3 Code-Änderungen: "später..."
- Konsequenz: Doku verrottet und **lügt**

### DIE EHRLICHE WAHRHEIT

- Zu weit weg vom Code
- Sperrige, unhandliche Formate (Word, Visio, PDF)
- Niemand vermisst schlechte Doku
- Das Erste, was bei Druck gestrichen wird
  (nehmt das, Tests!)

Das Ergebnis:
"Pi mal Daumen" und Inselwissen statt
Professionalität und Zukunftssicherheit.

# 04

# Die neue Tokenökonomie.

Und was hat Text und Doku mit KI zu tun?

# DOKU WIRD DANK KI ENDLICH BEZAHLBAR

**VORHER**

- Doku schreiben: ~2-4 h (mit Iterationen: 20h)
- Aktualisieren: 30 min/Änderung
- Doku & Code synchron halten: ¯\(ツ)/¯

**MIT KI**

- Doku-Entwurf: ~5 min
- Aktualisieren: ~2 min
- Synchronisation: nahezu kostenlos

Erstmals: Doku ist nicht nur erstellbar, sondern *wartbar*

# THESE: WIR MÜSSEN SOWIESO DOKU SCHREIBEN

**Wir müssen** der KI Kontext geben - sonst rät sie blind.

**Also:** Warum schreiben wir den Kontext nicht gleich so, dass auch *wir Menschen* etwas davon haben?

Doppelter Nutzen bei gleichem Aufwand.

Doku wird zum **Nebenprodukt** der Entwicklung - nicht mehr zum Luxusgut.

# ABER: TOKENS SIND NICHT GRATIS

KI wird absehbar **teurer**, nicht günstiger:

- Anthropic kürzt Pro-Pläne
- GitHub Copilot wechselt zu Token-Billing
- Größere Kontextfenster = höhere Kosten

Token-Ökonomie rückt noch stärker in den Vordergrund.

Was bedeutet das für die Wahl der Formate?

# TEXT GEWINNT. KLAR.

| PDF / OFFICE | AUDIO / VIDEO | MARKDOWN |
| --- | --- | --- |
| ❌ | ❌ | ✅ |
| Hohe Token-Kosten, schlechte Strukturerkennung | Sehr teuer, langsam, fehleranfällig | Maximale Informationsdichte pro Token |

Anstatt der KI umständliche Formate zu füttern, die sie dann teuer in Markdown verwandelt, können wir ihr die Arbeit abnehmen und gleich Markdown liefern.

Markdown als Format, das Mensch **und** Maschine verstehen,
ist in punkto Einfachheit, Informationsdichte und Token-Ökonomie
**unschlagbar**.

# 05

# Prompt Customization.

Klingt kompliziert, ist es aber nicht. Im Gegenteil.

# VORHER: ANEINANDER-VORBEI-REDEN

- Ellenlange Prompts mit immer denselben Grundlagen
- Trotzdem: hohe Varianz in den Ergebnissen
- KI weiß nichts über euren Stack
- KI weiß nichts über eure Konventionen
- Jede Session beginnt bei Null

Und täglich grüßt das Murmeltier.

# DIE NEUEN WERKZEUGE

PERMANENT

## AGENTS.md

Stack, Konventionen, Regeln

AUFGABEN-SPEZIFISCH

## Skills

"Wie macht man X bei uns?"

ERWEITERUNGEN

## Hooks & MCP

Automatisierte Kontext-Versorgung

Außerdem: Custom Agents, Subagents, Plugins - alles Varianten desselben Prinzips.

Das, was Du einem neuen Mitarbeiter erzählst,
musst Du auch der KI erzählen
- und zwar **dauerhaft**.

# DER AUGENÖFFNER

Diese Features sind **keine Magie**

Sondern einfach nur **Text**, der an bestimmten Stellen automatisch in den Kontext gehängt wird.

Ich hätte das auch von Hand simulieren können - per Copy&Paste in den Prompt.

## Prompt Customization = standardisierte Konventionen.

# DOPPELTER NUTZEN, SCHON WIEDER

### FÜR DIE KI

- Kontext, ohne ihn zu wiederholen
- Konventionen ohne Erklärung
- Beispiele aus dem eigenen Code
- Saubere Trennung von Prompt und Kontext

### FÜR UNS

- Onboarding-Doku
- Architektur-Übersicht
- Konventionen, schwarz auf weiß
- Weniger Tipparbeit, weniger Nacharbeit

Derselbe Text. Zwei Zielgruppen.

## Win-Win

# SOGAR DREIFACHER NUTZEN 🚀 🚀 🚀

## Markdown dient als:

1. Kontext für die KI
2. Doku für Menschen
3. Prompt Customization

...und das alles gleichzeitig - ohne Mehraufwand.

## Win-Win-Win

> Prompt Customization ist kein Spielzeug - es ist der neue Status Quo.

# 06

# Die Summe der Einzelteile.

Sechs Argumente, ein Format.

# DIE AHA-EFFEKT-LAWINE

KI braucht Kontext. Dauerhaft. Möglichst Token-effizient.

↓

Token-effizient heißt: Markdown.

↓

Dauerhaft heißt: Git.

↓

In Git versioniert ist Markdown automatisch Doku für Menschen.

↓

An der richtigen Stelle abgelegt wird Markdown gleichzeitig Prompt Customization.

↓

**Eine Datei. Drei Zwecke. Kein Mehraufwand. Einfach Text**

## TEXT-FIRST

Wenn es Text sein kann,
sollte es Text sein.

> Programmieren mit KI ist Programmieren mit Markdown.
> Ob wir wollen oder nicht.

Ja, aber ist das nicht das Gleiche wie...

# SPEC-DRIVEN DEVELOPMENT (SDD)

Ein verwandter Ansatz, der gerade groß wird:

- **Spec first** - Anforderungen vor Code explizit machen
- **Maschinenlesbare Contracts** - OpenAPI, JSON Schema (= Text!)
- **Gestufte Pipeline** - Intent → Spec → Plan → Tasks → Code
- **Verifikation** - Tests validieren gegen die Spec

Tools: GitHub Spec-Kit, get-shit-done, Kiro, ...

# SDD-FRAMEWORKS: PRO & CONTRA

**STÄRKEN**

- Spec als Single Source of Truth
- Reduziert KI-Varianz
- Architektur vor Code
- Verifikation eingebaut

**RISIKEN**

- Hoher Token-Verbrauch
- Zusätzliche Komplexität
- Noch eine Black Box zusätzlich zur KI
- Abstraction Lock-In

## Brauchen wir wirklich (noch) ein neues Framework?

# TEXT-FIRST: SDD OHNE FRAMEWORK-ZIRKUS

### SCHWERE SDD-FRAMEWORKS

- Überfluss an Spec-Dateien
- eigenes Tooling
- nicht immer LLM-agnostisch
- hoher Token-Verbrauch
- Kanonen auf Spatzen

### TEXT-FIRST

- Spec-Driven, aber nur wenige Dateien
- kein Tool, kein Download, kein Framework
- zukunftssicher: funktioniert mit jedem LLM
- Token-effizient, da markdownbasiert
- klein anfangen, iterativ erweitern

> SDD ist GitFlow. Text-First ist GitHub Flow.

# 07

# Text-First in der Praxis.

Konvention statt Framework - leichtgewichtig und ohne Lock-In.

# KONVERSATION INS REPO

> Die Diskussion mit der KI ist bereits Markdown.
> Sie enthält das **Warum**, die **Alternativen**, das **Wie**.

## Copy. Paste. Commit.

### WARUM ES REICHT

- Keine zusätzliche Formatierung
- KI versteht ihren eigenen Output sofort
- Kontext, der **wirklich** so entstanden ist
- Kein Template, kein extra Tool

### WAS WIR GEWINNEN

- Persistenter Kontext für die nächste Session
- Doku als Nebenprodukt, nicht als Aufgabe
- Audit-Trail in Git, datiert per Commit
- Weniger Tipparbeit beim nächsten Mal

# BEISPIEL: KONVERSATION ALS DOKU

```
# doc/api-endpoints.md

> Welche Endpunkte braucht ein einfaches ToDo-API?

CRUD reicht für den MVP:
- GET /todos, GET /todos/{id}
- POST /todos
- PUT /todos/{id}
- DELETE /todos/{id}

> Brauchen wir Filter wie ?completed=true?

Nicht im MVP. Lässt sich nachträglich ohne Breaking Change ergänzen.

> Status Codes?

200, 201, 204, 400, 404. Keine generischen Wrapper im Body.
```

Ein Dump der Konversation, im Repo versioniert. Reicht.

# BEFREIT DOKU AUS DEN DATENSILOS

Doku muss **näher an den Code** sonst nutzt sie weder Mensch noch KI.

**FRÜHER**

- Visio, Word, PDFs
- Fotos von Whiteboards - die keiner lesen kann
- Confluence-Seiten - die niemand liest oder pflegt
- "GitDocs"-Versuche scheitern an Aufwand

**HEUTE**

- Im gleichen Repo, neben dem Code
- KI-Konversationen im Repo
- AGENTS.md, Skills
- KI hilft bei Pflege → endlich praktikabel

Tipp: KI nutzen, um Doku *für die KI* zu erzeugen

**Mehr Meta geht nicht.**

# "ABER MEIN CHEF WILL PDF!"

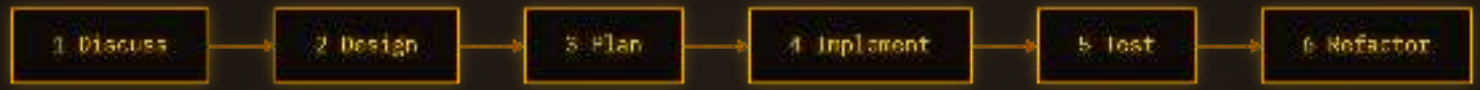
## Markdown ist die Quelle, nicht das Ziel.

Nicht-technische Stakeholder werden weiterhin
mit PDFs und PowerPoints versorgt

- auf Knopfdruck aus Markdown generiert.

> Write once, publish everywhere.

# DIE METHODIK IM ÜBERBLICK




### VOR-CODE-PHASEN

- **Discuss** - Brainstorming, Anforderungen
- **Design** - Diagramme, Modelle
- **Plan** - schrittweise Roadmap

### CODE-PHASEN

- **Implement** - Plan abarbeiten
- **Test** - Autokorrektur & Validierung
- **Refactor** - Qualitätssicherung

# DIE ARTEFAKTSPUR

| Phase | Artefakte | Nutzen |
|---|---|---|
| Discuss | Gesprächsnotizen, offene Fragen | Anforderungen werden sichtbar |
| Design | Konversation, Mermaid, API-Dokumentation | Entscheidungen werden belastbar |
| Plan | Implementierungsplan | Arbeit wird prüfbar und fortsetzbar |
| Implement | Code + aktualisierte Doku | Umsetzung folgt dem Plan |
| Test | Unit- und Integrationstests | KI bekommt Feedback |
| Refactor | Qualitätsnotizen, kleinere Umbauten | Ergebnis wird wartbar |

Jede Phase erzeugt Artefakte, die Kontext als Doku persistieren.

# VOR DEM START: AGENTS.MD

```
# AGENTS.md - ToDo-API

## Tech-Stack
- Spring Boot 4.0.4, Java 25, Gradle 9
- JUnit 6 + Mockito + RestTestClient
- Spring MVC (keine reaktiven APIs)

## Konventionen
- Boxed Boolean in DTOs (nicht boolean)
- Java- & REST-Best-Practices
- Persistenz aktuell: Map<Long, Todo>

## Required Reading vor jeder Aufgabe
- doc/api-endpoints.md
- doc/todo-class-diagram.md
```

Permanenter Kontext - jede Session startet damit.

AGENTS.md protokolliert globale Regeln, Strukturen und Konventionen.

# VOR DEM START: ENDPOINT-SKILL

```
---
name: create-endpoint
description: Neuen REST-Endpunkt im ToDo-API anlegen
---

# Endpoint anlegen

1. Controller-Methode in passende Controller-Klasse
2. Service-Methode für Geschäftslogik
3. JavaDoc inkl. curl-Beispiel
4. Unit- + Integrationstest (RestTestClient)
5. curl-Beispiel zu doc/curl-examples.md
6. README.md aktualisieren, falls nötig
```

Aufgabenspezifischer Kontext - on-demand abrufbar.

Ein Skill ist ein Workflow in Textform: Regeln, Templates, Beispiele und ggf. kleine Hilfsskripte.

# PHASE 1: DISCUSS

KI als **Sparringspartner** für Ideen und Anforderungen.

- Brainstorming-Session
- Technologieauswahl treffen
- Vor- und Nachteile abwägen
- **Tipp:** Konversation ins Repo kopieren - sonst ist sie weg

# → Demo

## #1 Discuss

Anforderungen für ToDo-API sammeln, Konversation ins Repo legen

```
What endpoints should a simple ToDo API provide?
```

# PHASE 2: DESIGN

Diagramme **als** **Code**

- **Mermaid** in Markdown - kein Visio, kein UML-Tool
- Versionskontrolliert in Git
- "Rapid Prototyping" zwischen Mensch und KI
- GitHub & GitLab rendern Mermaid nativ
- VS Code: Extensions

# BEISPIEL: KLASSENDIAGRAMM


Reiner Text. Reproduzierbar. Wartbar. Diffbar.

> "Design-Ping-Pong mit der KI"

# → Demo

## #2 Design

### Klassendiagramm + API-Endpunkte als Mermaid & Markdown

```
Propose a Mermaid class diagram for a simple ToDo resource served by the simple ToDo API.

 ──

Leave createdAt and updatedAt out for now.
```

```
- Add the Mermaid diagram to `doc/todo-class-diagram.md`.
- Add the API endpoints you proposed above to `doc/api-endpoints.md`.
```

# PHASE 3: PLAN

Plan-Mode statt **impulsivem** **One-Shot-Coding**

**CODE-FIRST**

- "Bau mir X": Generiert sofort Code
- KI rät, halluziniert
- Viel Nacharbeit
- Kein roter Faden

**PLAN-FIRST**

- KI recherchiert, fragt nach
- Schritt-für-Schritt-Plan
- Erst freigeben, dann umsetzen
- Plan ins Repo committen

Plan ist gleichzeitig Doku für Menschen und Anker für die nächste Session.

# PLAN-PROMPT: KURZ, WEIL KONTEXT EXISTIERT

```
Plan the initial implementation of the TODO API.

Steps in implementation order.
```

Der Prompt muss nicht mehr alles erklären.
Das Repository erklärt mit.

# → Demo

#3 Plan

Implementierungsplan auf Basis aller bisherigen Artefakte

# PHASE 4: IMPLEMENT

> Implementieren **nach** **Plan**, nicht nach Impuls.

**KI**

- Pair-Programmer mit Kontext
- Hakt Schritte ab
- Setzt Schritte fort
- Fragt bei Unklarheiten nach
- Dokumentiert Plan-Updates

**MENSCH**

- Überwacht den Prozess
- Gibt Feedback zu Plan und Code
- Bei Unklarheiten: Plan anpassen, nicht den Code
- **Niemals durchwinken** - prüfen, verstehen, lernen
- Plan-Updates dokumentieren (lassen)

# → Demo

#4 Implement

Plan ausführen

# PHASE 5: TEST

> Ohne Tests **stochert die KI im Dunkeln**

- Tests = Autokorrekturmechanismus
- KI kann iterieren, ohne dass du Logs vorlesen musst
- Tests sind **Living Documentation**
- KI erfindet mittlerweile Tests - **im Code verankern!**
- Bestehende Tests = Kontext für neue Tests

# → Demo

#5 Test

Geschieht automatisch während der Implementierung

# PHASE 6: REFACTOR

KI verliert bei großen Aufgaben den **Überblick**.
Dank Refactoring behältst **du** ihn.

- KI tendiert zu Redundanz und Drift
- Refactoring als **fester** Bestandteil, nicht nice-to-have
- KI kann Vorschläge machen - Du entscheidest
- Deterministische IDE-Tools bevorzugen, wo möglich

# → Demo

## #6 Refactor

Refactoring-Vorschläge einholen, gezielt anwenden

```
Fasse ähnliche Code-Abschnitte zusammen, ohne Funktionalität zu ändern.
```

# DER EIGENTLICHE GEWINN

**QUALITÄT** ↑

Mehr relevante Constraints im Kontext.

**VARIANZ** ↓

Weniger Roulette bei Architektur und Stil.

**NACHARBEIT** ↓

Fehler werden früher und systematischer abgefangen.

Nicht "die KI wird magisch besser".
Das Team erklärt nur seine Erwartungshaltung besser.

# WAS BLEIBT HÄNGEN

1. **Sprache** ist die universelle Schnittstelle - auch zur KI.
2. **Markdown** ist die Muttersprache der LLMs.
3. **Doku** ist erstmals bezahlbar - nutzt das.
4. **Kontext** ist König. Eine Datei, drei Zwecke.
5. **Text-First** Wenn es Text sein kann, sollte es Text sein.

Kein Tool-Zirkus. Kein Hokuspokus.
Einfach nur Text.

## Text-First.

> KI ist gekommen, um zu bleiben.
> KI ist explizit nicht die Blockchain.
> Wer sich KI konsquent verwehrt, wird abgehängt. -Martin Boßlet

> AI will not replace you. A person using AI will. -Garry Kasparov

# DANKE FÜR EURE AUFMERKSAMKEIT.

Let's make AI great already 🎉

Fragen?

- https://www.gedoplan.de/
- https://www.linkedin.com/in/martin-bosslet/
- mailto:hello@martinbosslet.dev
- `@klautcode` on BlueSky
- `@klautcode` on Mastodon
- `_emboss_` on X