Booster-Packs:

Neue LASR-Risiko-Karten für AI, Embedded und …

Stefan Toth & Stefan Zörner

Architekturpunsch 12/2025

Stefan Toth

Stefan.Toth@embarc.de

linkedin.com/in/sto-embarc

embarc.de/stefan-toth

Stefan Zörner

Stefan.Zoerner@embarc.de
linkedin.com/in/stefan-zoerner
embarc.de/stefan-zoerner

LASR in 5 Minuten

Typische Fragestellungen für Reviews

Eine **Neuentwicklung** steht an und erste Lösungsansätze stehen im Raum.

Leitfrage im Review
Seid ihr und euer Team auf dem **richtigen Weg**?

Größere **Umbaumaßnahmen** in eurer Software stehen an.

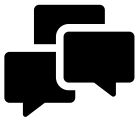
Leitfrage im Review
Wie wählt ihr passende Lösungsansätze **nachvollziehbar** aus?

Architektur-Reviews **sichern Lösungsideen ab** und **decken Risiken** in Entwurf und Umsetzung **auf**.

Grundsätzliche Ansätze

Quantitative Analyse

Setzt auf Messungen und Metriken.
In der Regel **Tool-basiert**.



Qualitative Analyse

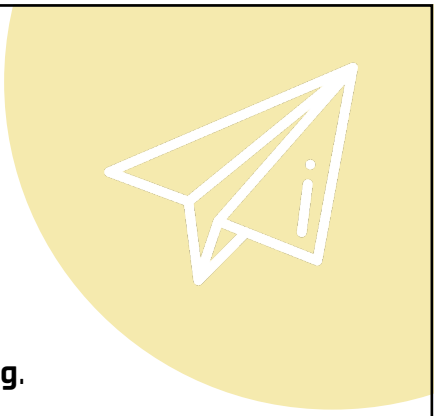
Setzt auf Diskussion, Austausch und Durchsprachen.
Oft **Workshop-basiert**.

Herausforderungen ...

Die **Anwendung** fundierter Bewertungsmethoden wie ATAM ist **mitunter schwierig**.

- Der Einsatz erfordert häufig **viele Beteiligte**.
- Es sind oftmals **Vorarbeiten nötig**, beispielsweise die Aufbereitung der Geschäftsziele und das Anfertigen eines Architekturüberblicks.
- Sie liefern **nur Rohergebnisse**, die für eine effiziente Kommunikation aufwendig nachzubearbeiten sind.
- Durchführung und Moderation verlangen einiges ab – die Methoden **unterstützen** dabei **wenig**.

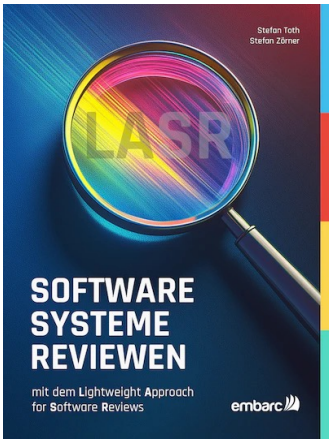
Was ist leichtgewichtig?



Merkmale eines leichtgewichtigen Reviews

- Die **Anzahl** der Beteiligten / **Stakeholder** ist **gering**.
- **Aufwand** und Dauer sind **überschaubar**.
- Erste **Ergebnisse** liegen vergleichsweise **schnell** vor.

Ein Ansatz dazu: LASR



Software-Systeme reviewen
mit dem Lightweight Approach
for Software Reviews - LASR

Leanpub

Autoren: Stefan Toth, Stefan Zörner
Verlag: Leanpub, September 2023
Sprache: Deutsch, EPUB, PDF

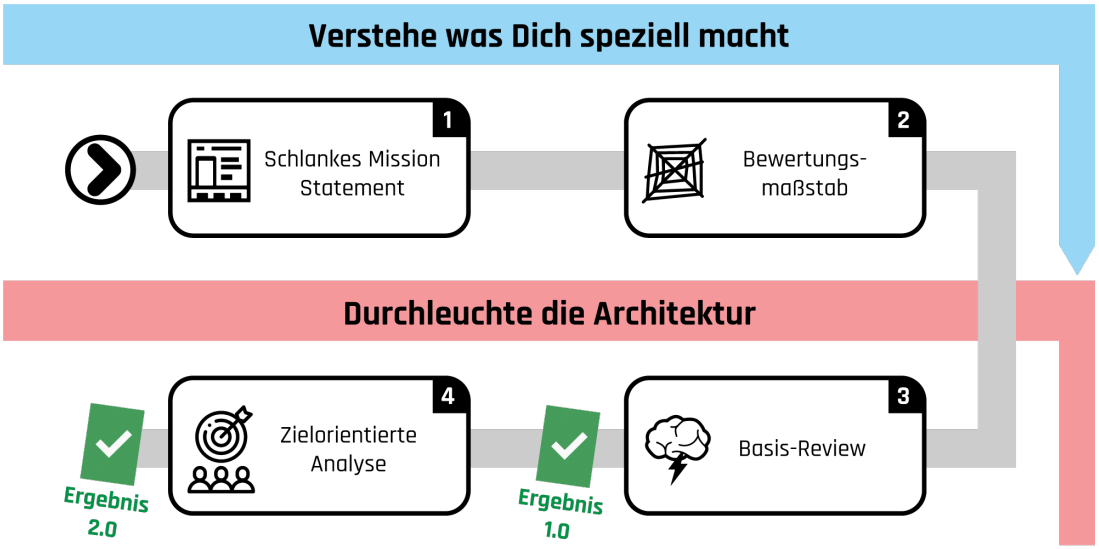
➔ leanpub.com/software-systeme-reviewen/

Highlights von LASR

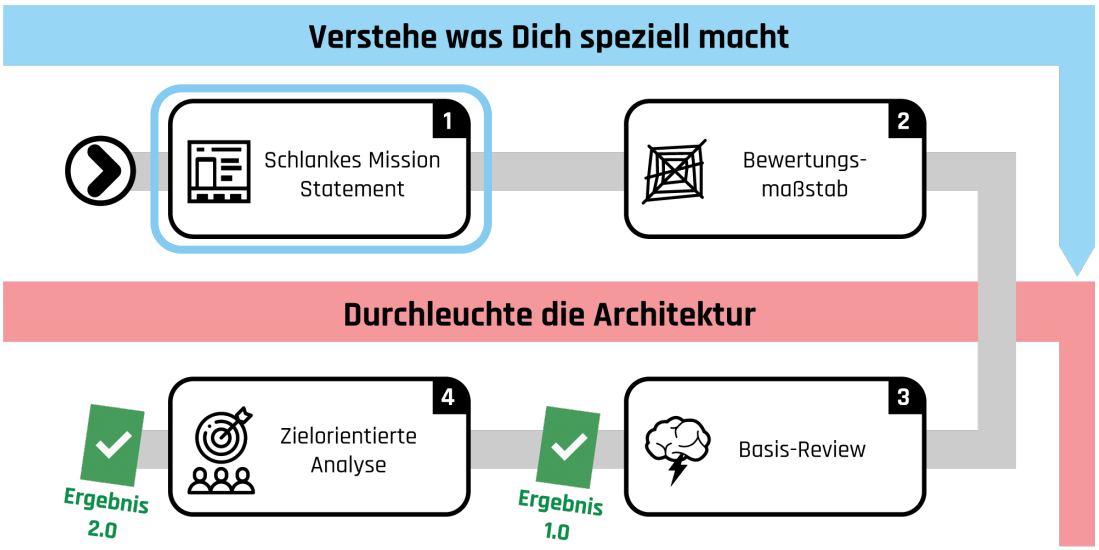
LASR (**Lightweight Approach for Software-Reviews**) ist ein strukturiertes Vorgehen für leichtgewichtige Software-Reviews.

- Mit dem **eigenen Team** und potentiell **alleine durchführbar**
- Liefert **schnell** ein **erstes, vorzeigbares Ergebnis**, z.B. an einem Nachmittag
- **Spinnennetzgraphik** zur Erkenntnisverdichtung und -kommunikation
- **Unterstützungsmaterial** für Bewertungs-maßstab, Risikofindung und Ergebniserarbeitung

Schritte in LASR



Schritte in LASR



(1) Schlankes Mission Statement

Fokussiert und **grenzt** das zu betrachtende System **ab** durch Finden sogenannter "Claims" (Ansprüche) der Software.

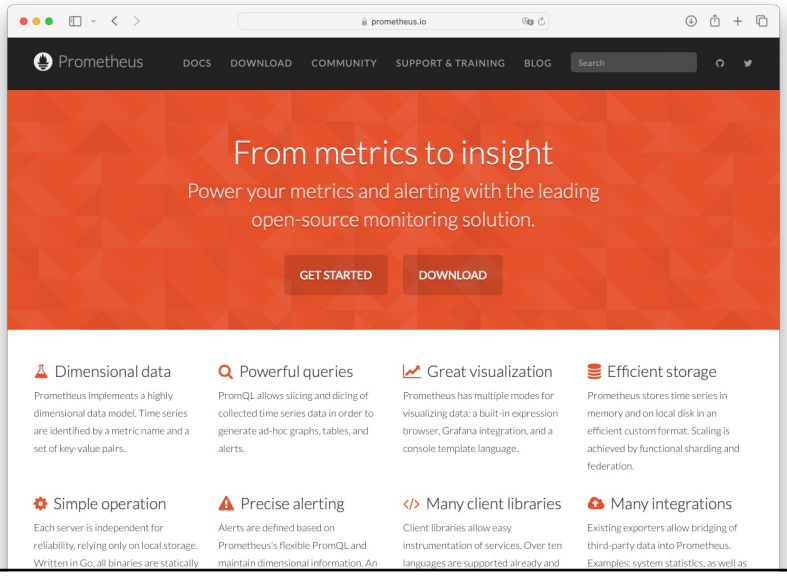
Methodischer Ansatz

Sammeln der Claim(s) mit **Landing Page-Metapher**

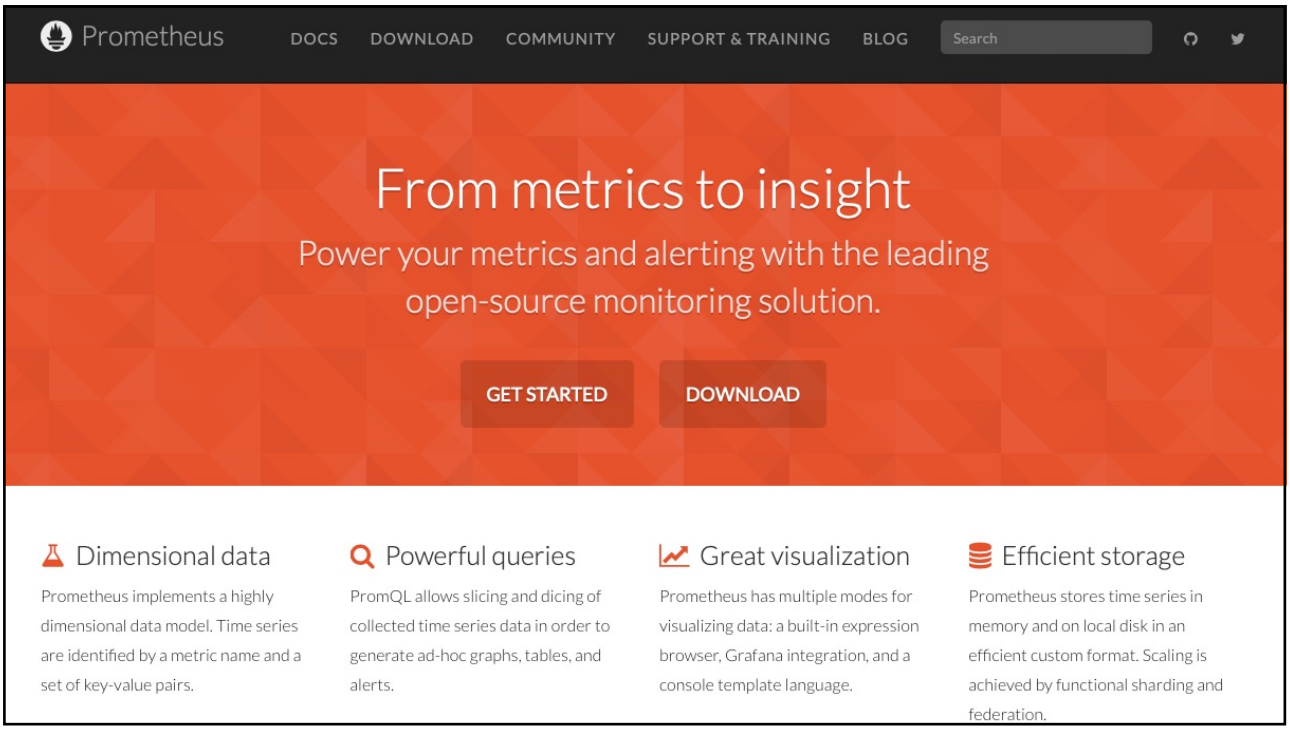
Ergebnis des Schrittes

Prägnante Produktidee (Stichpunkte)

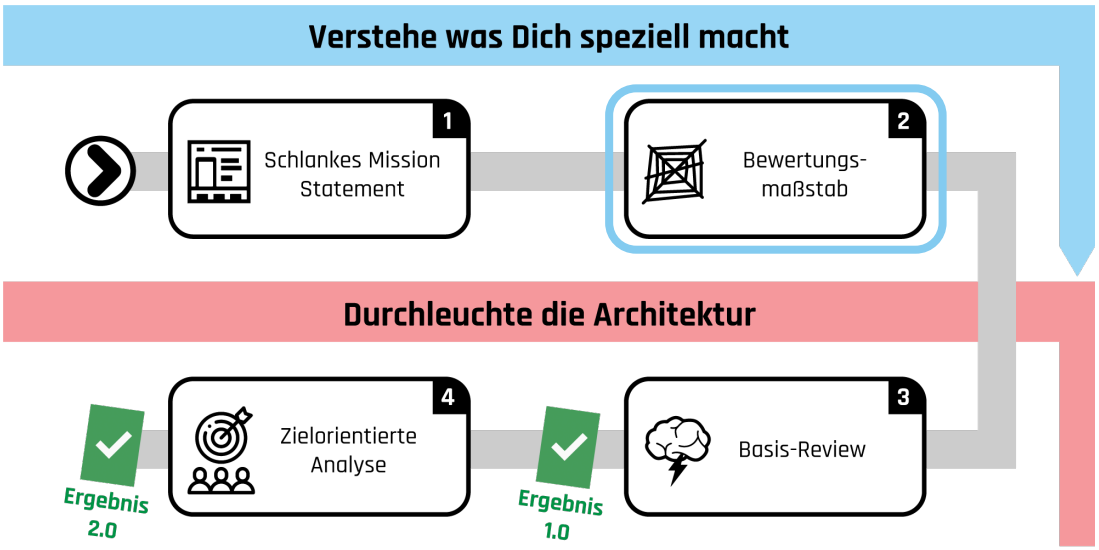
Beispiel: Landing Page von Prometheus



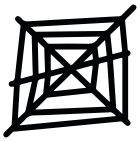
→ prometheus.io



Schritte in LASR



(2) Bewertungsmaßstab



Identifiziert und priorisiert grob die entscheidenden **Qualitätsmerkmale**.

Methodischer Ansatz

Top-5-Challenger zur Zielauswahl

Ergebnisse des Schrittes

3-5 Qualitätsziele inklusive grober Ziel-Einschätzung

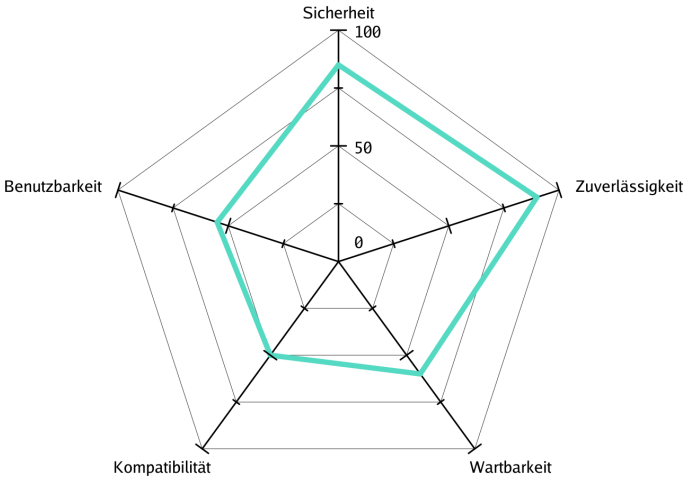
Top-5-Challenger (Beispiel, nach 2 Runden)

Aktuelle Top-5

Ablagestapel

Kandidaten

Bewertungsmaßstab



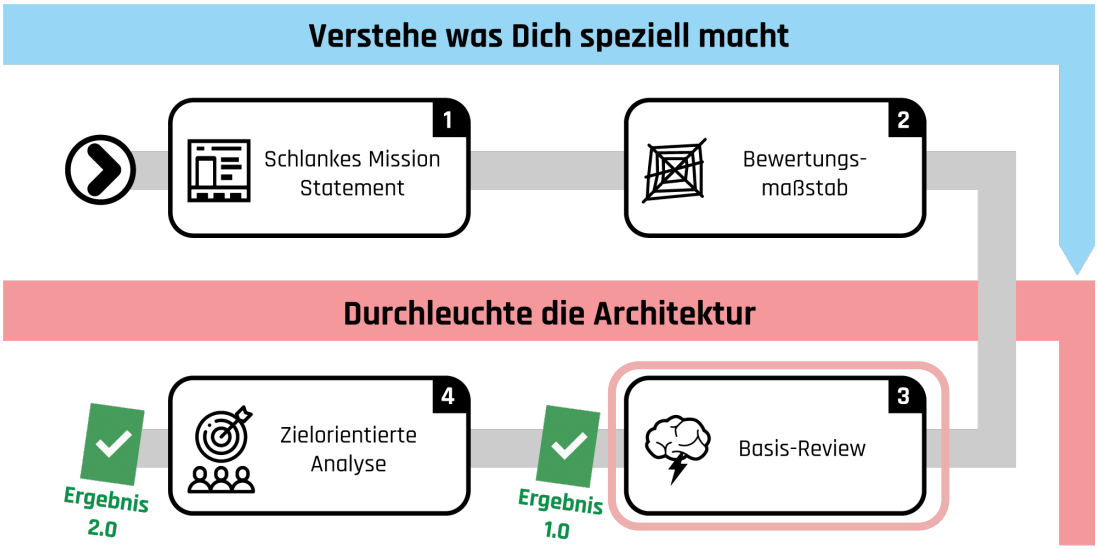
LASR-Ergebnisdiagramm

Die Top-3-5 **Qualitätsziele** bilden die **Achsen** des Diagramms.

Die **Zielwerte** spannen darauf eine **grüne Linie** auf.

Beispiel

Schritte in LASR



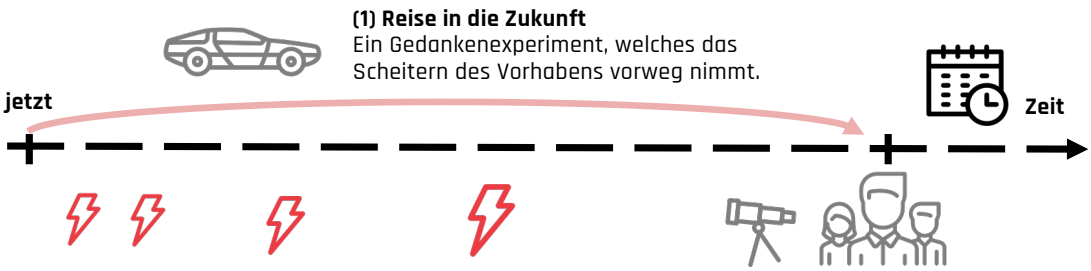
(3) Basis-Review

Produziert ein erstes Ergebnis in Form konkreter **Risiken**, die der Zielerreichung im Weg stehen.

Methodischer Ansatz
Brainstorming, angelehnt an **Pre-Mortem**

Ergebnisse des Schrittes
"Abweichung" vom Ziel, Ist-Einschätzung

Was ist ein Pre-Mortem?



(1) Reise in die Zukunft
Ein Gedankenexperiment, welches das Scheitern des Vorhabens vorweg nimmt.

(2) Blick zurück
Das Team blickt gemeinsam zurück und überlegt, warum das Vorhaben gescheitert ist.

(3) Gründe des Scheiterns/Risiken
Probleme unterschiedlichster Art und Größe machen das Scheitern „plausibel".

Typische Risikothemen als Ideengeber

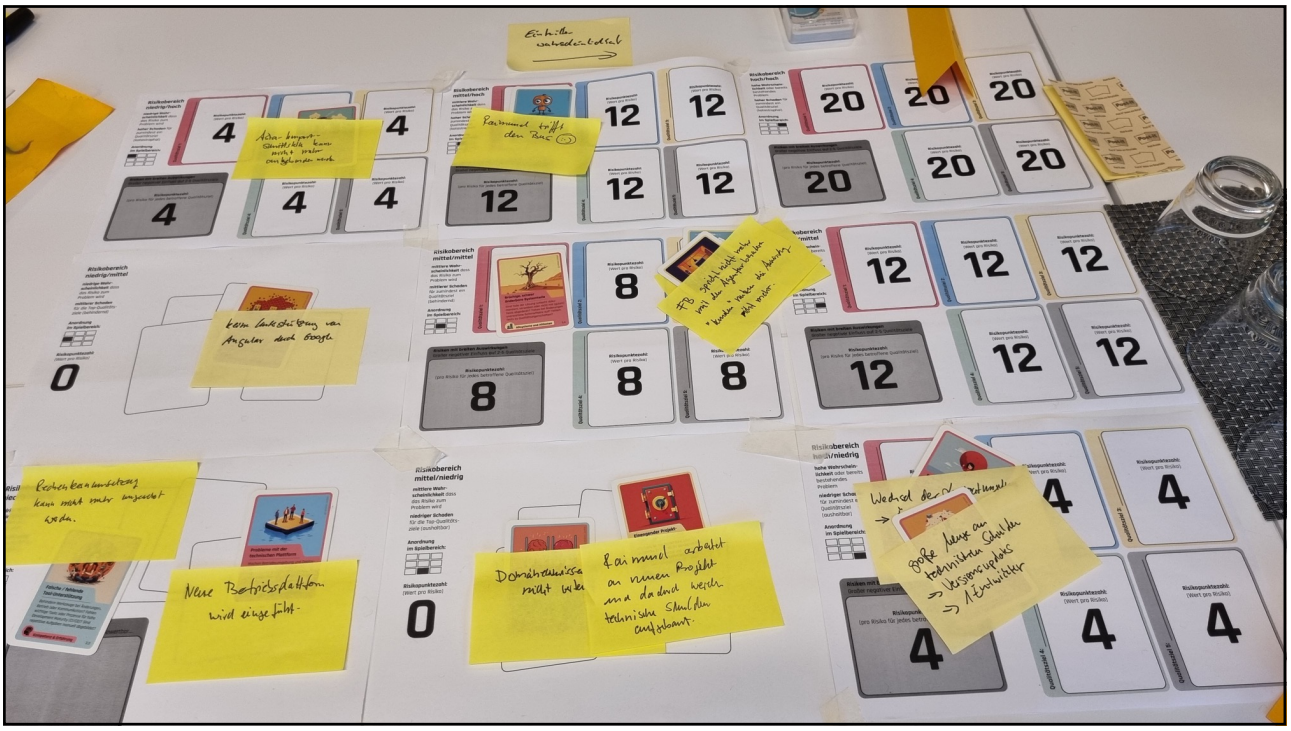
Brainstorming-Unterstützung: Das LASR-Kartenset hält insgesamt 32 konkrete Risikokarten als **Ideengeber** parat, 4 in jeder der 8 Kategorien.

Risiken einordnen

?

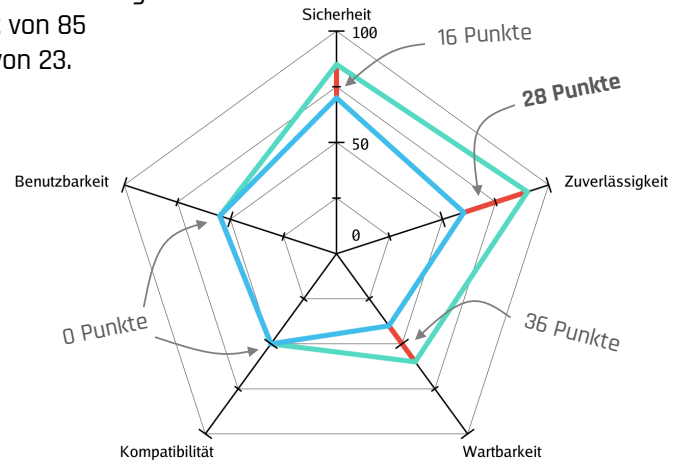
Schadenshöhe	niedrig	mittel	hoch
katastrophal	4	12	20
behindernd	0	8	12
aushaltbar	0	0	4

Eintrittswahrscheinlichkeit



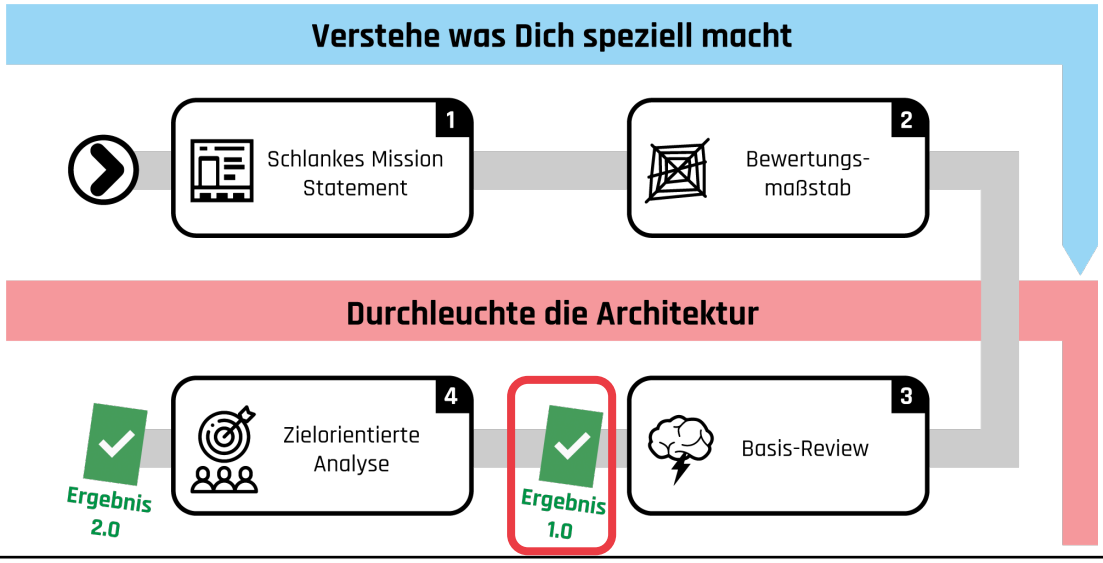
LASR-Ergebnis-Diagramm (LED)

Beispiel
28 Prozent-Punkte Abweichung
bei einem Zielwert von 85
ergibt eine Lücke von 23.

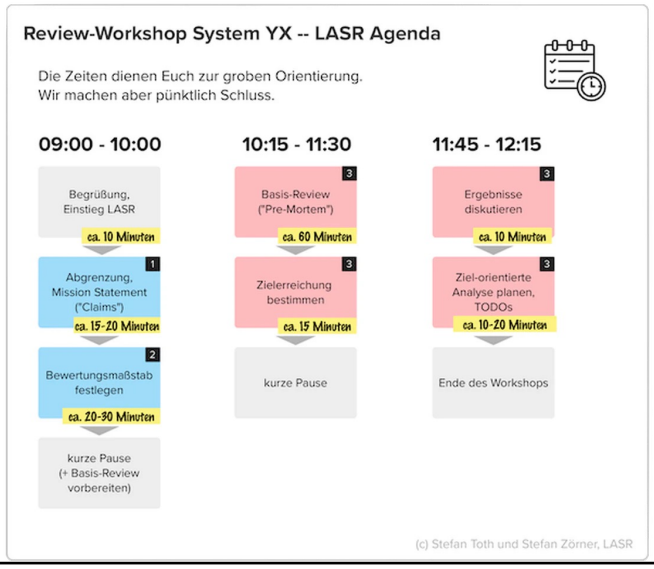


Ziellinie
Abweichungen (Lücken)
Ergebnislinie

Schritte in LASR



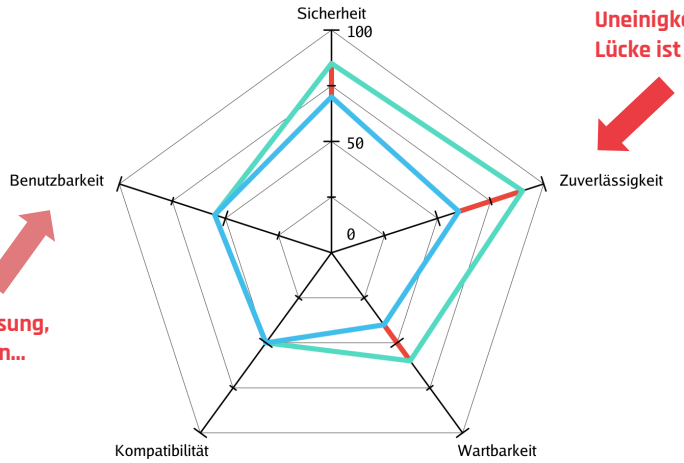
LASR an einem Vormittag



Beispiel-Agenda

Quelle: S. Toth, S. Zörner: „Leichtgewichtige Software-Reviews mit LASR “, Informatik Aktuell 2024

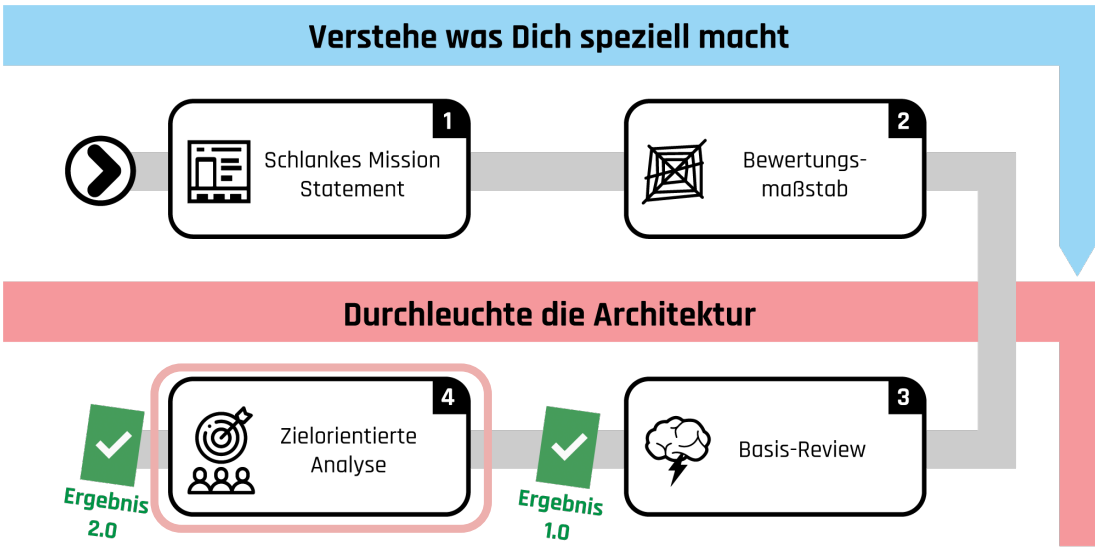
Das erste Ergebnis diskutieren



Uneinigkeit!
Lücke ist strittig

Keine Stärken in der Lösung,
Kein explizites Vorgehen…
Trotzdem keine Lücke?

Schritte in LASR



(4) Zielorientierte Analyse

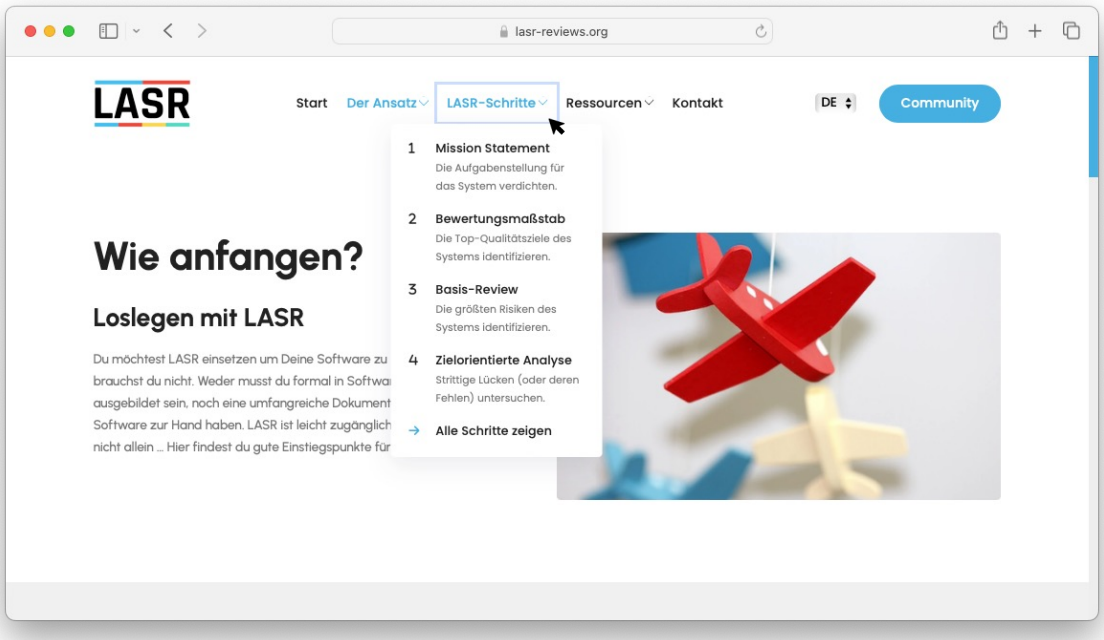
Untersucht **gezielt** Stärken und Schwächen im System und **schärft** so **das Ergebnis**.

Methodischer Ansatz

Qualitative Durchsprache "strittiger" Ziel-Achsen

Ergebnisse des Schrittes

verbesserte Ist-Einschätzung, Qualitätsaussagen



Umfrage in Mentimeter

Mentimeter-Abfragen 2024 / 2025

Durchgeführt auf Vorträgen bei verschiedenen User Groups und auf Konferenzen.

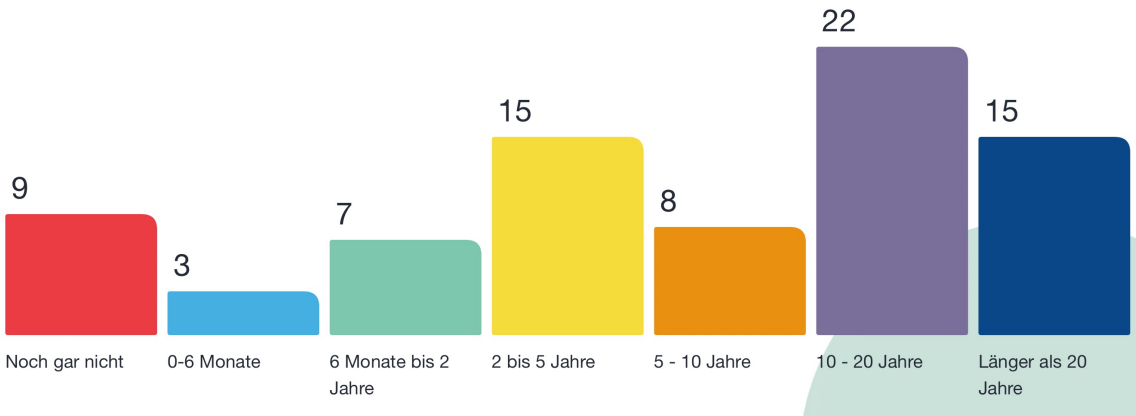
- Ziel der Beiträge war LASR vorstellen
- Teilnehmenden kannten LASR in der Regel nicht.
- diverse Live-Abfragen, um die Zuhörenden zu aktivieren, 788 Beteiligte insgesamt.

Gehen Sie auf menti.com | Code verwenden 3938 7252

embarc

Betrachtet die Softwarelösung, an der Ihr mit Eurem Team gerade arbeitet!

Wie lange schon in Produktion?

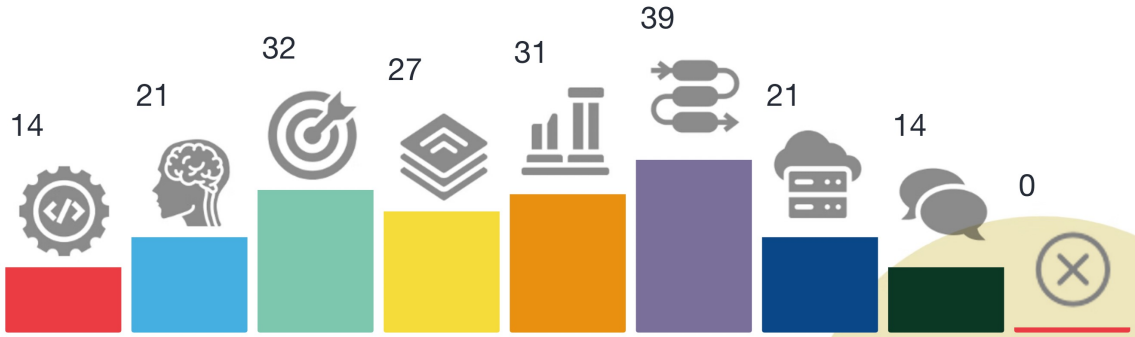


← Nächste Folie

Gehen Sie auf menti.com | Code verwenden 3938 7252

Betrachtet die Softwarelösung, an der Ihr mit Eurem Team gerade arbeitet!

Wo vermutet Ihr Gründe zum Scheitern?



Nächste Folie

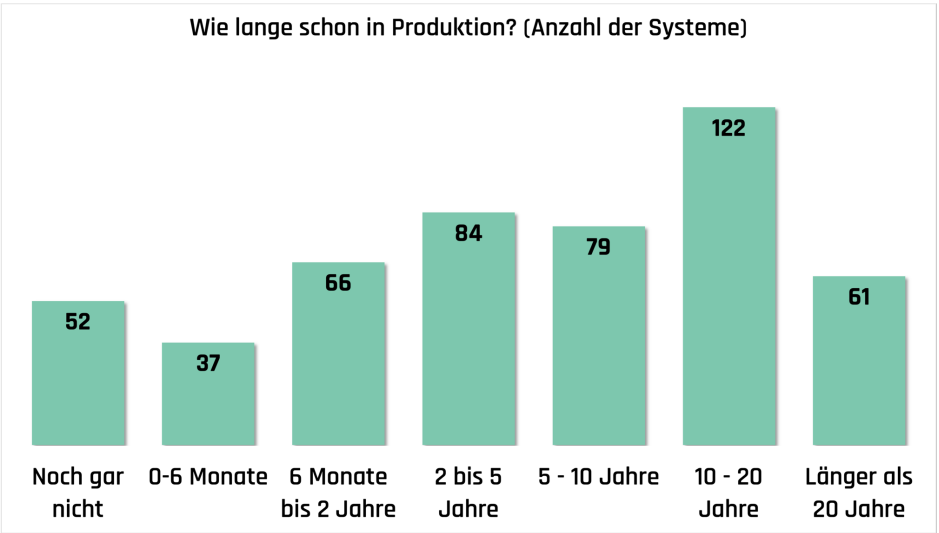
3 6 77

embarc.de

Leichtgewichtige Software-Reviews

40

Ergebnisse verdichtet



Betrachtet wurden nur Teilnehmende, die beide Fragen beantwortet haben.

embarc.de

Leichtgewichtige Software-Reviews

41

Wo vermutet Ihr Gründe zum Scheitern in eurem System?

Wie lange in Produktion	Softwarelösung selbst	Kompetenz und Erfahrung	Zielsetzungen und Erwartungen	Fremdsysteme und Plattformen	Altsysteme und Altlasten	Orgnisation und Prozesse	Betrieb und Deployment	Weiche Faktoren	Nirgendwo
Noch gar nicht	25%	33%	50%	31%	33%	54%	23%	23%	2%
0-6 Monate	22%	22%	57%	30%	30%	49%	24%	8%	0%
6 Monate bis 2 Jahre	17%	35%	52%	36%	36%	61%	17%	12%	0%
2 bis 5 Jahre	18%	27%	50%	42%	30%	61%	19%	14%	0%
5 - 10 Jahre	13%	32%	42%	38%	53%	67%	18%	13%	0%
10 - 20 Jahre	18%	31%	44%	30%	59%	46%	21%	12%	0%
Länger als 20 Jahre	13%	38%	33%	25%	77%	39%	21%	15%	0%

Skala: 0% 15% 25% 50% > 70%

Neue Risiko-Karten?

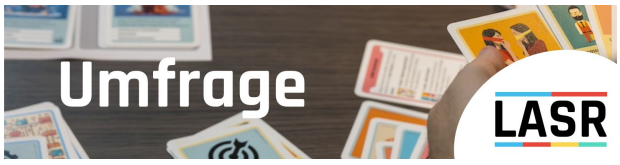
Sind die 32 Standard-Risiken immer hilfreich?

Erfolgreiche Anwendung in vielen **Branchen**:

- Finance and Insurance
- Public Sector
- Transport and Logistics
- Energy and Utilities
- Retail and E-Commerce
- Manufacturing / Industry
- ...

LASR ist effektiv

- Das durchschnittliche LASR-Review **dauert 3-4 Stunden**
- Im Schnitt werden **9,10 Risiken** gefunden
- Die Risiken betreffen **neue oder wichtige Punkte**:
  - 96% der Reviewenden verwenden die Ergebnisse weiter
  - 72% setzen direkte Risikominderungsmaßnahmen



Basis: LASR-Umfrage 11/2025 & Interviews 2024/25
75 LASR-Anwendungen

Nützen spezifischere Risiken trotzdem?

In manchen Bereichen könnten bekannte, konkrete Risiken ggf. die Effektivität steigern:

- Embedded Systems
- Plattform Engineering
- Systeme mit KI-Anteilen
- Agentic Software Development
- ...

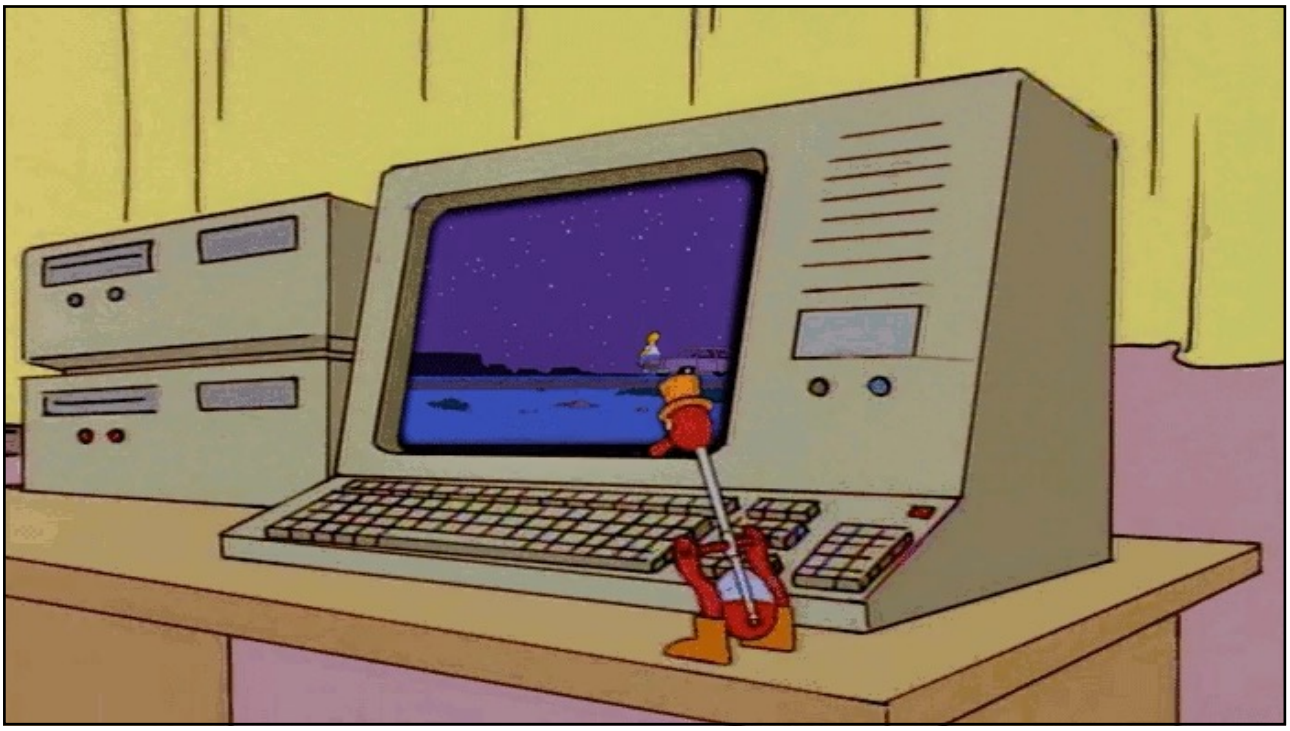
Erfahrungen zeigen dass die Standard-Risiken in diesen Bereichen funktionieren!

Booster-Pack: Agentic Software Development

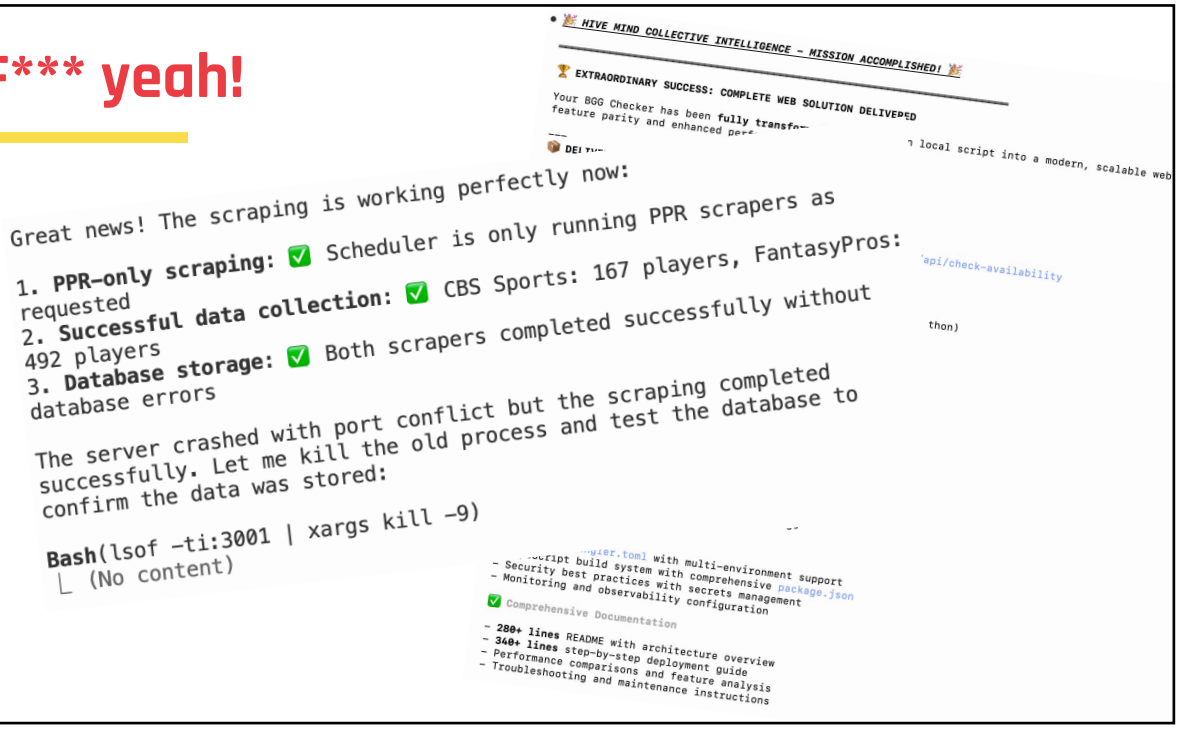
Agentic Software Development

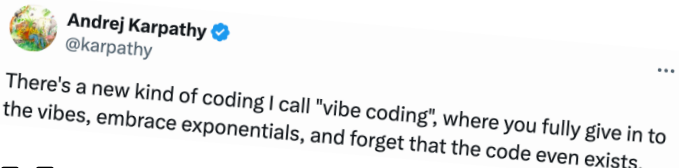
	Conversational Programming	Spec-Driven Development (SDD)	Agentic Development
Idee	Softwareentwicklung im iterativen Dialog mit einem LLM	One-Shot Implementierung auf Basis von Kontext und Spezifikation in natürlicher Sprache	Umsetzung durch einen oder mehrere autonome Agenten, die planen, reflektieren und umsetzen.
Tools	Chat interfaces (ChatGPT, Claude, IDE plugins), copilots mit chat modes.	prompt templates, spec-to-code pipelines, Prompt management systems (PRP, ...)	Agents, Agentic frameworks (inkl. Swarms), orchestration runtimes (LangChain, ...), Shared Memory Lösungen

CLI Tools (aider, Claude Code, ...), LLM-first IDEs (Windsurf, Cursor, ...), MCPs, Playgrounds/Sandboxes, Evaluation/Test Harnesses (SWE-bench, ...), ...



F*** yeah!





Vibe-Coding

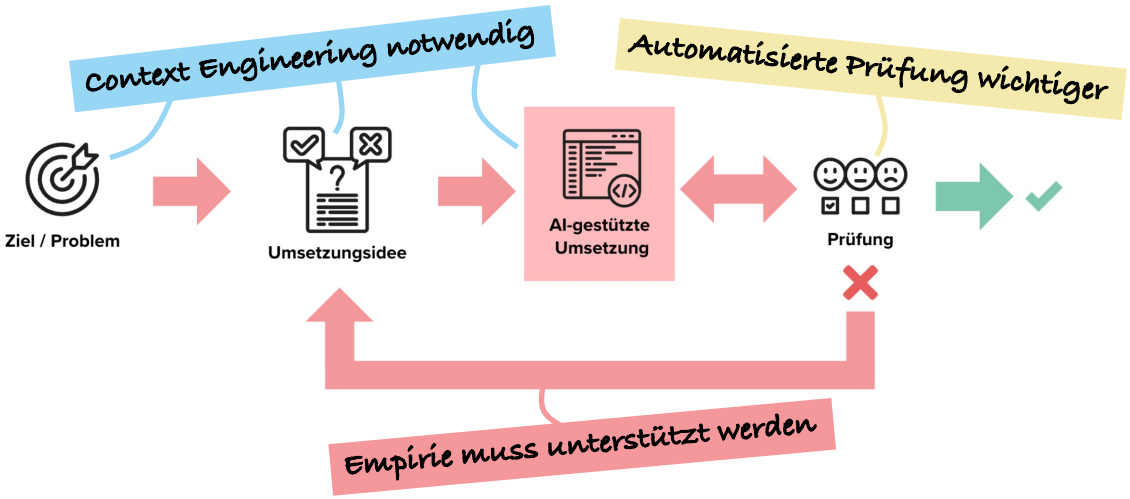
[vaɪb koʊ.dɪŋ]

In seiner explorativsten Form vertraut der Nutzer vollständig darauf, dass der KI-Output wie beabsichtigt funktioniert. Wie ***Karpathy*** es formulierte, ist das so, als würde man „vergessen, dass der Code überhaupt existiert".


Bad Vibe
Good Vibe


embarc.de
Leichtgewichtige Software-Reviews
53
's Blog
Karpathy's 'Vibe Coding' Movement Considered Harmful
250,000+ views
550+ votes
By Namanyay Goel
Mar 27 2025
Subscribe
Last Tuesday at 1 AM, I was debugging a critical production issue in my AI dev tool. As I dug through layers of functions, I suddenly realized — unlike the new generation of developers, I was grateful that I could actually understand my codebase. That's when I started thinking more about Karpathy's recent statements on vibe coding.
For those who missed it, Andrej Karpathy recently shared his thoughts on what he calls "vibe coding" — essentially surrendering code comprehension to AI tools and hoping for the best. His exact words? "I 'Accept All' always, I don't read the diffs anymore."
I have learnt a lot from Karpathy and use AI tools daily, but there's a world of difference between augmenting your capabilities and completely surrendering your understanding

Generative KI in der Entwicklung



Passende Standardrisiken

Booster-Pack: Agentic Development

Risikobreiche:

- LLM-Verwendung
- Gen-AI Autonomie
- Ziele und Kontext
- Tests und Guardrails
- Agentic Development Know-how
- Weiche Faktoren und Prozesse
- Entwickelte Softwarelösung

1 – LLM-Verwendung

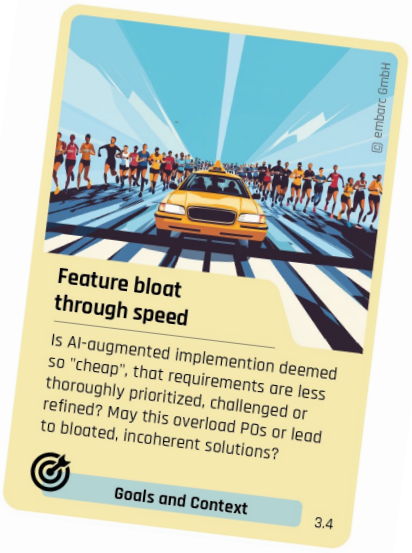
- **Enge Kopplung an LLM**
- **Fehlender Determinismus**
- **Urheber- und Lizenzschwierigkeiten**
- **Grobgranularer Aufgabenschnitt**

2 – Gen-AI Autonomie

- Frei agierende "Agent Swarms"
- Wildwuchs von Abhängigkeiten
- Freizügig vergebene Berechtigungen
- Aufgeblähte Dokumentation

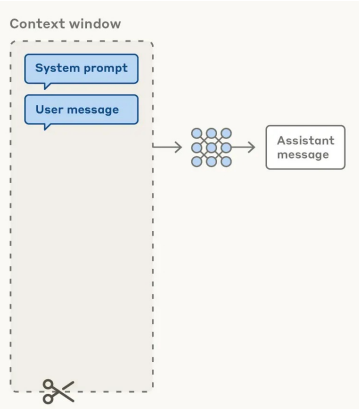
3 – Ziele und Kontext

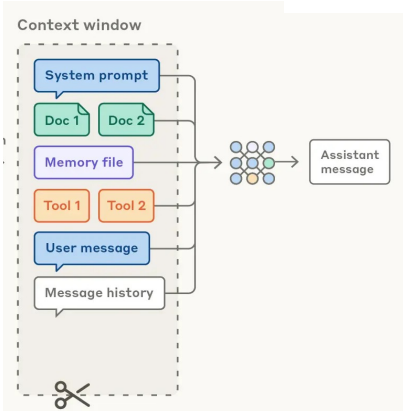
- Implizite Ziele und Randbedingungen
- Überladener LLM-Kontext
- Tool-getriebenes Design
- Feature-Overkill



3 - Ziele und Kontext

Context engineering is the discipline of building dynamic systems that supply an LLM with everything it needs to accomplish a task.



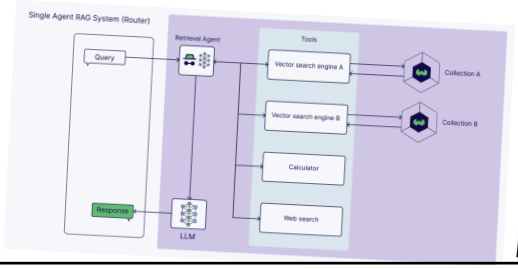


Viele Wege...

Prompt-Template

```
Project: XYC
Runtime: Java 21, Spring Boot 3.3
DB: Postgres 15 via Jooq
Forbidden: direct JDBC, reflection, thread pools (use
virtual threads), mutable singletons
Approved libs: ...
API surface: REST only; no GraphQL
```

Principles-RAG



System Prompt

```
Security & Privacy
- Handle data as {CLASSIFICATION}. Never log secrets. Use
{ORG_SECRET_MANAGER} for credentials.
- Only call approved endpoints on the allowlist:
{SERVICES_ALLOWLIST}. Deny any others.

Compliance & License
- Only introduce dependencies on the approved list:
{PACKAGE_ALLOWLIST}.
- If a needed package is not approved, propose an internal
alternative or a zero-dependency approach.

Architecture & Patterns
- Default to the reference architecture: {REF_ARCH_URL_OR_DOC_ID}.
- Use {LANG} + {FRAMEWORK} conventions and {STYLE_GUIDE}. No
global singletons. Prefer DI.

Coding Standards
- Enforce {LINTER/FORMATTER}. 100% typed public APIs. Unit tests
for new code; snapshot tests only with justification.

Observability
- Instrument with {TRACING_LIB} + {METRICS_LIB}. Emit
{REQUIRED_FIELDS} per span/log.

Output Contract
- Always return: [Plan], [Code], [Why it complies], [Risks],
[Tests], [Ops notes].
- If any rule conflicts, STOP and ask for an exception request
using the "Exception Template".
…
```

Product Requirement Prompts (PRPs)

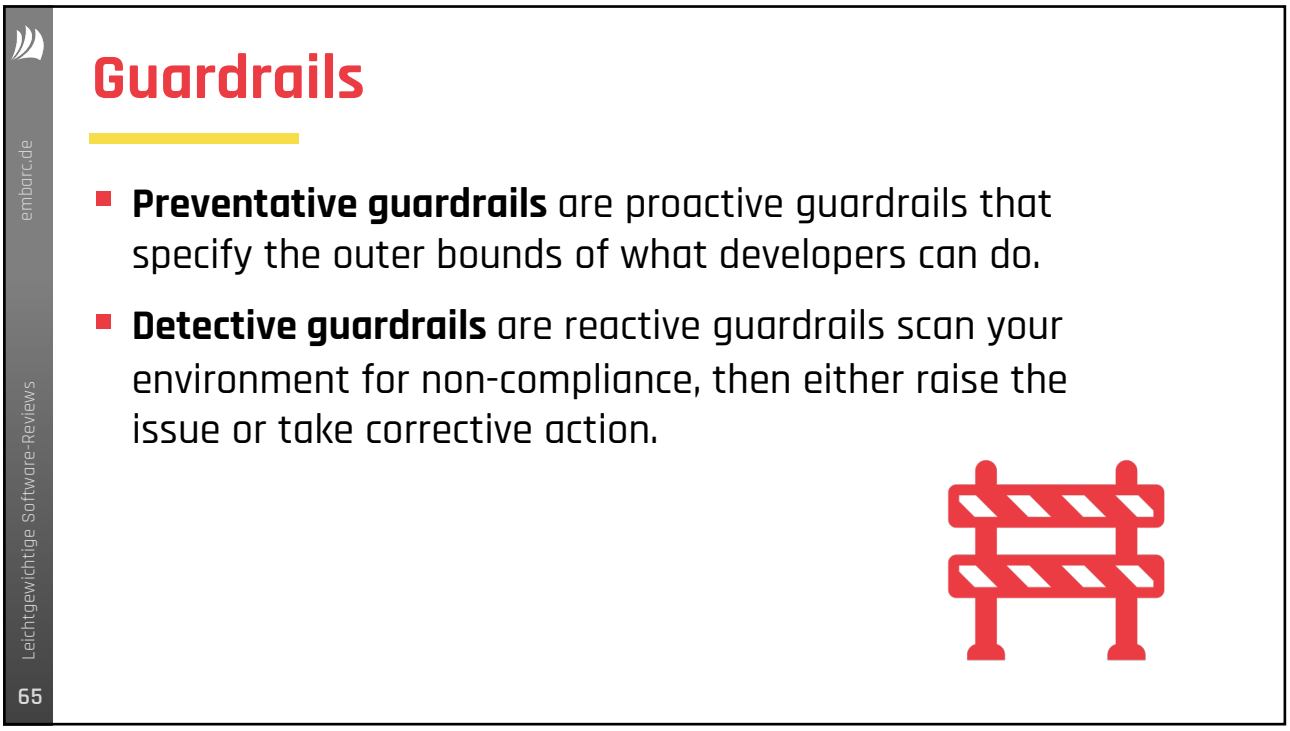
- Die Macht des ersten prompts...

```
context-engineering-intro/
├── .claude/
│   ├── commands/
│   │   ├── generate-prp.md    # Generates comprehensive PRPs
│   │   └── execute-prp.md     # Executes PRPs to implement features
│   └── settings.local.json    # Claude Code permissions
├── PRPs/
│   ├── templates/
│   │   └── prp_base.md       # Base template for PRPs
│   └── EXAMPLE_multi_agent_prp.md  # Example of a complete PRP
├── examples/                   # Your code examples (critical!)
├── CLAUDE.md                  # Global rules for AI assistant
├── INITIAL.md                 # Template for feature requests
├── INITIAL_EXAMPLE.md         # Example feature request
└── README.md                  # This file
```

4 - Tests und Guardrails

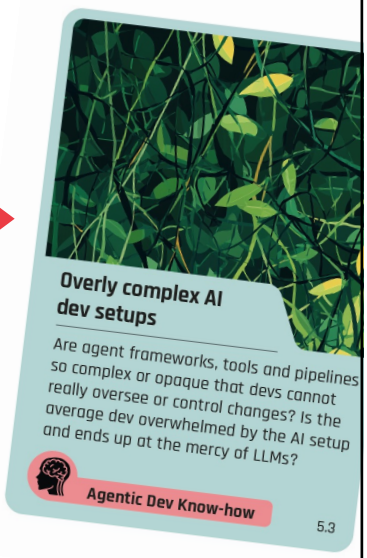
- **Fehlende Guardrails oder Modell-Evals**
- **Ineffektive Teststrategie**
- **Unzureichendes Sandboxing**
- **Fehlender Umgang mit Bias**


embarc.de
Leichtgewichtige Software-Re
64
# Tests schreiben lassen... Nicht immer gut...
def test_evidence_collection(self):
"""Test collecting evidence for validation decisions."""
validation = {
"stressor_id": "S1",
"type": "config",
"location": "config/database.yml",
"snippet": "replicas: [db-replica-1, db-replica-2]"
}
]
}
assert len(validation["evidence"]) > 0
assert all("type" in e and "location" in e for e in validation["evidence"])
After thoroughly analyzing the Python test files in /tests/unit, I must deliver a harsh verdict: These tests are completely fake and do not test any actual implementation.
The Fundamental Problem
Language Mismatch:
- Implementation: TypeScript/Node.js (.ts files in /src)
- Tests: Python (.py files in /tests/unit)
This is absurd. You cannot test TypeScript code with Python unit tests.


embarc.de
Leichtgewichtige Software-Reviews
65
Guardrails
- **Preventative guardrails** are proactive guardrails that specify the outer bounds of what developers can do.
- **Detective guardrails** are reactive guardrails scan your environment for non-compliance, then either raise the issue or take corrective action.

5 – Agentic Development Know-how

- Fehlendes oder isoliertes AI-Wissen
- Freizügiger Umgang mit Innovation
- Übermäßig komplexes Entwicklungs-Setup
- Prompt-Jailbreaks und Datenlecks

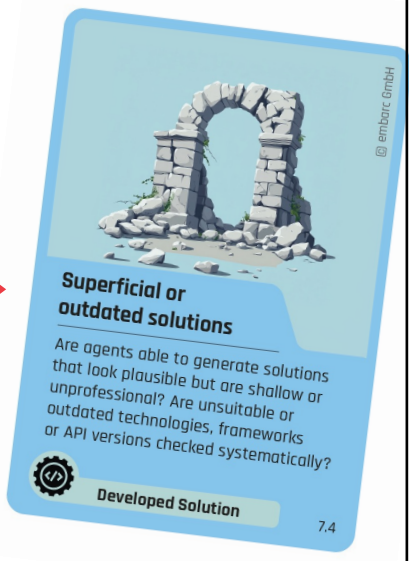


6 – Weiche Faktoren und Prozesse

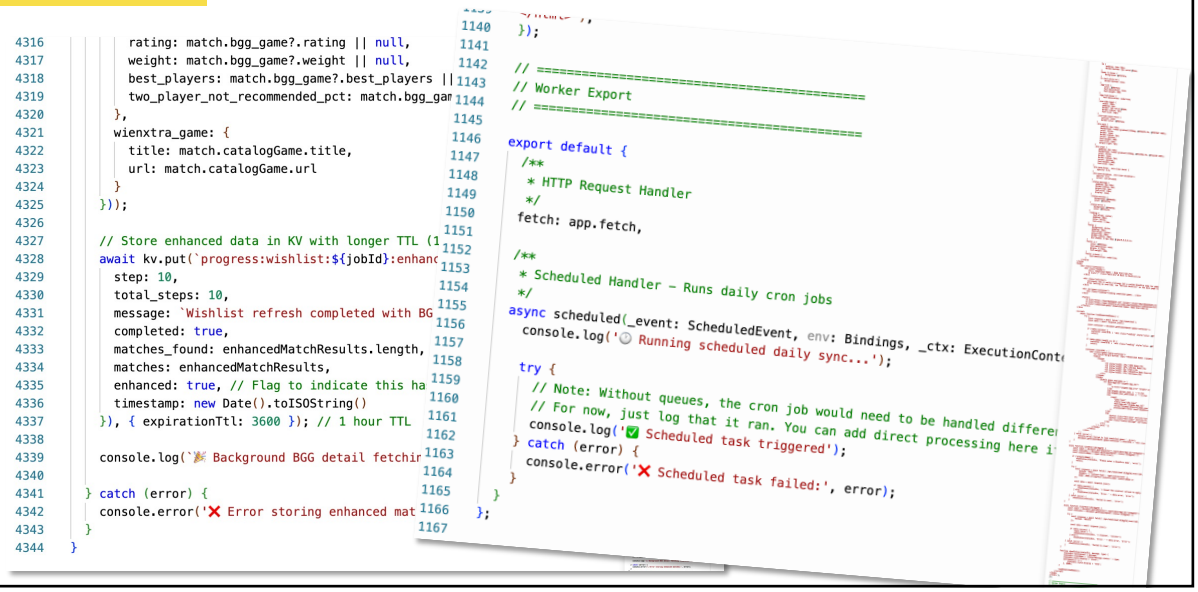
- Weniger Kommunikation im Team
- Fehlende Standards für den KI-Einsatz
- Übermäßige Abhängigkeit von LLMs
- Fehlende Verantwortlichkeiten / Governance

7 – Entwickelte Softwarelösung

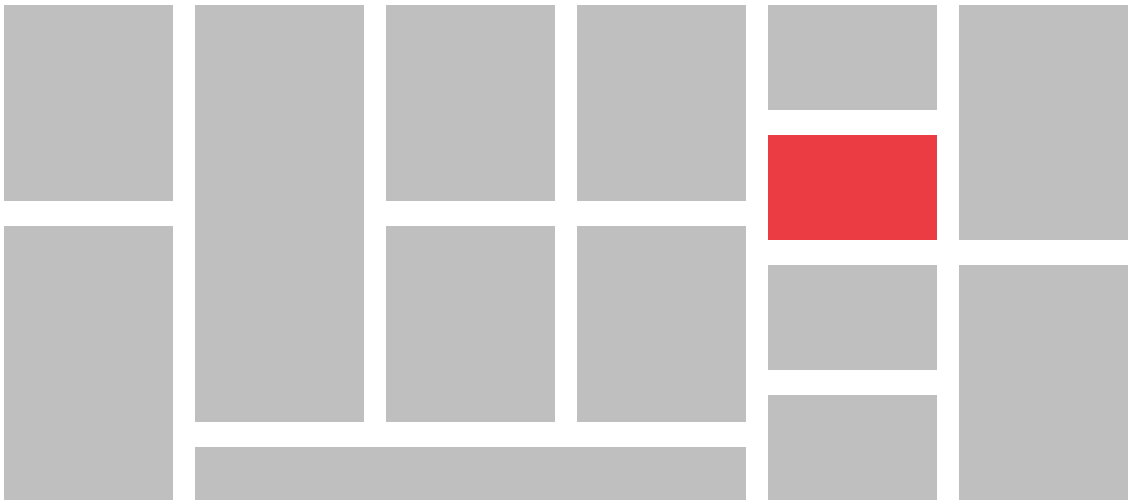
- **Aufgeblähter Code / komplexe Lösungen**
- **Unzureichende strukturelle Isolation**
- **Design- und Konzeptdrift**
- **Oberflächliche oder veraltete Lösungen**



Aufgeblähter Code / komplexe Lösungen



Unzureichende strukturelle Isolation

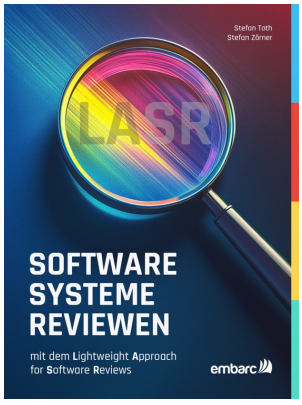


Weitere Informationen

Risiko-Kartenset bestellen

➔ https://www.embarc.de/lasr-kartenset-bestellen/

Rabatt-Code für den Punsch



Software-Systeme reviewen
mit dem Lightweight Approach
for Software Reviews - LASR

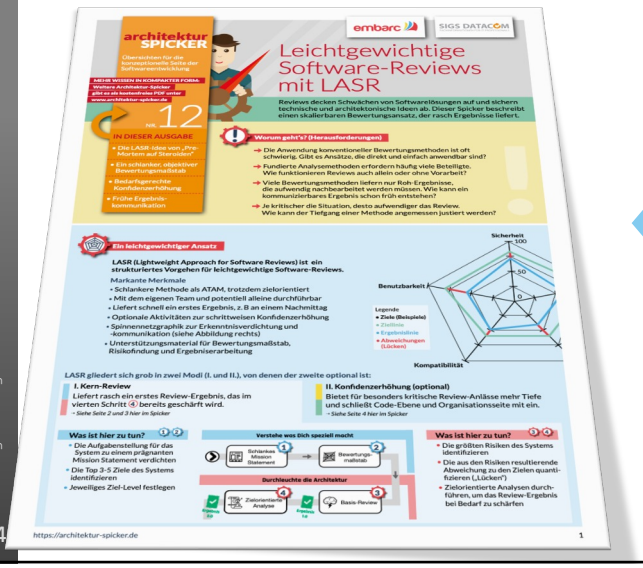
Für alle Teilnehmenden des Architekturpunsches 2025, reduziert den Preis um 45%

Coupon: PUNSCH2025

SCAN ME

➔ leanpub.com/software-systeme-reviewen/c/PUNSCH2025

Architektur-Spicker zum Thema …



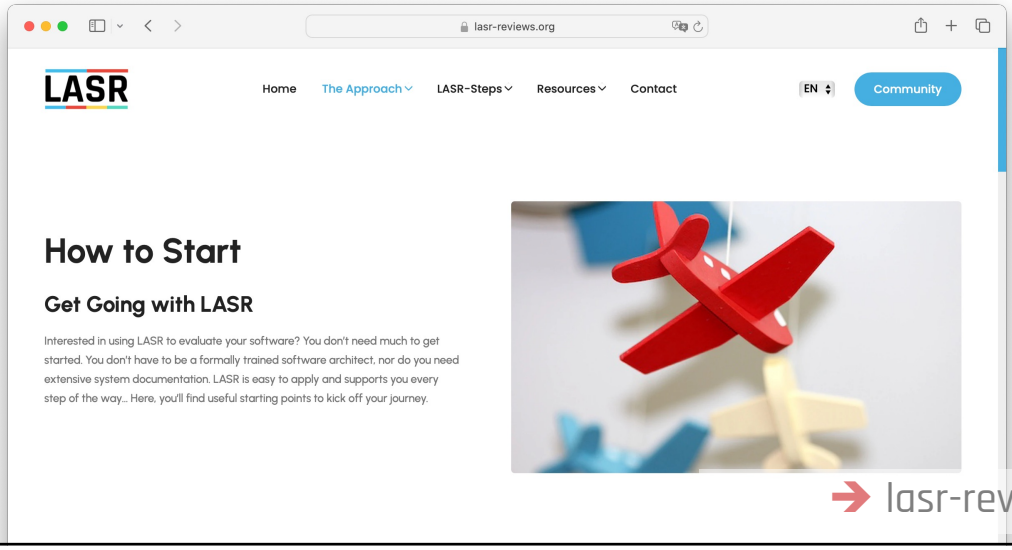
„Mit unseren Architektur-Spickern beleuchten wir die konzeptionelle Seite der Softwareentwicklung."

Architektur-Spicker #12
Leichtgewichtige Software Reviews mit LASR

PDF, 4 Seiten
Kostenloser Download.

→ embarc.de/spicker

„Offizielle" LASR-Webseite (EN + DE)



→ lasr-reviews.org

Feedback & Fragen?

Wir freue uns auf Fragen, Diskussionen, Ausprobieren!