Die Agenten-Review-Checkliste

Begleitpaket zum Buch „Vibe Coding, KI-Engineering & Agenten-Orchestrierung"

von Thorsten Behder

Die Security- und Governance-Prüflisten aus dem Buch als Checkliste für jeden KI-gestützten Pull-Request-Review, bevor ein Agenten-Workflow ins Team geht.

CHECKLISTE

Sechs Prompt-Vorlagen und zwei Sicherheits-Gates aus dem Buch, wörtlich übernommen. Die Gates arbeiten Sie ab, bevor Sie den nächsten Schritt gehen.

Sechs Vorlagen aus dem Prompt-System

Aus dem Buch — Prompt-System PS-04 bis PS-15

1. PS-04 Intent Brief

Rolle: Du bist mein technischer Sparringspartner.

Ziel: Ich möchte [Ziel] erreichen.

Kontext: [Branche, Nutzer, Daten, Systeme, Einschränkungen].

Nicht-Ziele: [Was ausdrücklich nicht passieren soll].

Erfolgskriterien: [Woran erkennen wir ein gutes Ergebnis?].

Risiken: Bitte nenne fachliche, technische, rechtliche und operative Risiken.

Arbeitsweise: Stelle zuerst maximal 5 Rückfragen, wenn Informationen fehlen. Erst danach arbeitest du weiter.

Output: Liefere eine strukturierte Umsetzungsplanung mit Annahmen, Optionen und einem klar benannten nächsten Schritt.

2. PS-06 Prototype Builder

Rolle: Du bist mein Bau-Partner für einen ersten, vorführbaren Prototyp.

Ziel (ein Satz): [Was soll der Prototyp sichtbar machen?]

Kontext:
- Nutzer/Rolle: [Wer bedient es?]
- Eingabe: [Welche Daten, welches Format? Beispiel anhängen.]
- Ausgabe: [Was soll am Ende auf dem Bildschirm stehen?]
- Umgebung: [Wo läuft es? z. B. eine einzelne HTML-Datei im Browser]

Ausdrücklich Prototyp:
Es geht um Erkenntnis und Vorführbarkeit, nicht um Produktionsreife.
Autonomie-Level 1: Der Prototyp darf nur ANZEIGEN und VORSCHLAGEN,
nichts auslösen, versenden oder verändern.

Arbeitsweise:
1. Stelle mir maximal 3 Rückfragen, falls Wesentliches fehlt.
2. Baue die EINFACHSTE Version, die das Ziel zeigt – eine Datei,
   so wenige Abhängigkeiten wie möglich.
3. Verwende gut sichtbare, benannte Stellschrauben (z. B. Schwellwerte)
   ganz oben im Code, damit ich sie leicht ändern kann.
4. Nenne mir am Ende ausdrücklich:
   a) welche Annahmen du getroffen hast,
   b) was dieser Prototyp NICHT kann,
   c) womit ich ihn als Erstes zum Absturz bringen würde.

3. PS-07 Prototype-to-Spec

Rolle: Du bist mein Anforderungs-Analyst. Wir überführen einen ungeprüften Prototyp in eine belastbare Spezifikation.

Ausgangslage: Der Prototyp macht Folgendes: [Kurzbeschreibung des sichtbaren Verhaltens].
Nutzerrollen: [alle beteiligten Rollen].
Datengrundlage: [welche Daten, Format, Herkunft].

Arbeite den Prototyp systematisch in eine Spezifikation um und liefere die folgenden sieben Teile:

1. User Stories je Rolle im Format "Als [Rolle] möchte ich [Ziel], damit [Nutzen]".
2. Datenmodell: die fachlichen Datenobjekte, ihre wichtigsten Felder und ihre Beziehungen zueinander.
3. Prozessfluss: die Schritte vom Input bis zum Ergebnis als nummerierte Kette.
4. Fehlerfälle: was bei fehlenden, falschen, leeren oder untypischen Eingaben passieren soll (je Fall ein Sollverhalten).
5. Qualitätsanforderungen: Nachvollziehbarkeit, Genauigkeit, Aktualität/Performance.
6. Sicherheitsgrenzen: welche Daten NICHT verarbeitet werden und welche Aktionen eine menschliche Freigabe brauchen.
7. Akzeptanzkriterien: je User Story prüfbare Bedingungen im Format "Gegeben / Wenn / Dann", inklusive Rand- und Fehlerfälle.

Wichtig: Markiere JEDE Stelle ausdrücklich, an der der Prototyp eine fachliche Annahme trifft, die noch bestätigt werden muss (z. B. Schwellwerte, Formate, Berechnungen).

4. PS-13 RAG Quality Checker

Rolle: Du prüfst die Qualität einer RAG-Antwort gegen ihre Wissensbasis.

Eingaben:
- Frage: [Nutzerfrage]
- Abgerufene Chunks (mit Quelle/Version): [eingefügte Top-k Chunks]
- Erzeugte Antwort: [Antwort des Systems]
- Rolle des Fragenden: [z. B. Einkauf]

Prüfe und antworte je Punkt mit BESTANDEN / FRAGLICH / DURCHGEFALLEN
plus einer knappen Begründung:

1. Deckung: Enthalten die Chunks die zur Frage nötige Information?
   Falls nein: Was fehlt vermutlich in der Wissensbasis?
2. Treue: Folgt jede Aussage der Antwort aus den Chunks?
   Markiere jede Aussage ohne Beleg als mögliche Halluzination.
3. Aktualität: Ist die genutzte Version die aktuell gültige?
   Nenne Datum/Version je Quelle.
4. Quellenkorrektheit: Passt die zitierte Fundstelle zum Inhalt?
5. Zugriffsgrenzen: Ist ein Chunk enthalten, den diese Rolle
   nicht sehen dürfte? Falls ja: markiere als VERSTOSS.

Abschluss: Gesamturteil + die zwei wichtigsten Korrekturen.

5. PS-10 AI Pull Request Reviewer

Rolle: Du bist ein kritischer Code-Reviewer für einen Pull Request.
Kontext: [Fachliche Absicht + Link/Auszug der Akzeptanzkriterien].
Eingabe: [Diff / geänderte Dateien].

Prüfe entlang von vier Achsen und trenne sie klar:

1. KORREKTHEIT
   - Setzt der Code die genannte Absicht um?
   - Wo verhält er sich an Grenzen/Sonderfällen falsch oder undefiniert?

2. SICHERHEIT
   - Neue Datenzugriffe, Berechtigungen, Abhängigkeiten, Eingabepfade?
   - Wo könnte nicht vertrauenswürdige Eingabe Wirkung entfalten?

3. TESTABDECKUNG
   - Prüfen die Tests die Anforderung oder nur den geschriebenen Code?
   - Welcher Test fehlt, um einen fachlichen Fehler zu fangen?

4. BLINDE FLECKEN
   - Welche Annahme trifft der Code, ohne sie zu benennen?
   - Was würde ein erfahrener Domänenexperte hier hinterfragen?

Ausgabe:
- Befunde nach Schwere sortiert (kritisch / wichtig / Hinweis).
- Zu jedem kritischen Befund ein konkreter Änderungsvorschlag.
- Am Ende: Was du NICHT beurteilen kannst und ein Mensch prüfen muss.

6. PS-15 Retrospective & Improvement

```
Rolle: Du bist mein Moderator für eine System-Retrospektive.
Zeitraum: [z. B. letzte 4 Wochen Betrieb].
Datengrundlage: [Eval-Reports, Incident-Liste, Nutzer-Feedback,
Akzeptanz- und Wiederöffnungsrate, auffällige Traces].

Führe mich strukturiert durch vier Fragen:
1. Was lief gut? Welche Prompts, Tools und Workflows haben
   verlässlich funktioniert (mit Beleg aus den Daten)?
2. Was lief schlecht? Wo gab es Fehlbewertungen, Incidents,
   abgelehnte Vorschläge oder teure Umwege?
3. Welche Evals fehlen oder greifen nicht? Welcher reale Fehler
   wurde von keinem Golden Case abgefangen?
4. Welche konkreten Änderungen leiten wir ab – je an Prompt,
   Tool Contract, Eval-Set oder Kontext?

Ausgabe:
- Priorisierte Liste von max. 5 Verbesserungen.
- Je Verbesserung: betroffenes Artefakt, erwartete Wirkung,
  wie wir den Erfolg messen.
- Was NICHT geändert wird und warum (bewusst stabil halten).
```

Gate 1: vor dem ersten Prompt

Aus dem Buch — Anhang E: Sicherheits-Checklisten

- [ ] Ich habe den Modus bewusst deklariert: Prototyp (Erkenntnis) oder produktionsnah (Wirkung) – der Übergang bleibt eine Entscheidung, kein Zufall.
- [ ] In den Prompt gelangen keine echten personenbezogenen oder vertraulichen Daten; für Tests nutze ich anonymisierte oder fiktive Daten („Musterlieferant GmbH").
- [ ] Kein Secret (API-Schlüssel, Passwort, Token) steht im Prompt – Zugangsdaten gehören neben den Agenten, nicht in den Kontext.
- [ ] Ich habe die Aufgabe des Copiloten in einem Satz benannt, damit ich später beurteilen kann, ob er nur das tut, was er soll.
- [ ] Mir ist klar, welche Datenwege das genutzte Werkzeug hat und ob eingegebene Daten beim Anbieter verbleiben oder zu Trainingszwecken verwendet werden könnten.
- [ ] Für jeden erzeugten Output gilt: Ich prüfe ihn, bevor ich ihm vertraue – souveräner Ton ist kein Korrektheitssignal.

Gate 2: vor dem ersten Agentenlauf

Aus dem Buch — Anhang E: Sicherheits-Checklisten

- [ ] Der Agent läuft in einer isolierten Sandbox, deren Grenzen er selbst nicht verändern kann.
- [ ] Alle Rechte folgen dem Least-Privilege-Prinzip; überflüssige Sichtbarkeit (etwa auf `erp.zahlungen`) und überflüssige Schreibrechte sind entfernt.
- [ ] Alle folgenreichen Tools laufen im Dry Run und werden erst durch eine menschliche Freigabe scharf geschaltet.
- [ ] Schreibende Aktionen sind über Transaktionen, Snapshots und ein vollständiges Protokoll rückrollbar; kritische Tools sind idempotent.
- [ ] Kein Secret steht im Prompt, im Log oder im Code; alle Zugangsdaten liegen im Vault mit eng geschnittenen Tokens und Rotation.
- [ ] Rate Limits pro Minute, pro Tag und als Kostenbudget in Euro sind gesetzt und lösen bei Überschreitung einen Stopp mit Meldung aus.
- [ ] Ein Kill Switch ist vorhanden, von außen bedienbar, global wirksam und allen Verantwortlichen bekannt.
- [ ] Der Rollback-Weg für einen Fehllauf ist einmal geprobt worden – nicht nur theoretisch vorhanden.

> Die weiteren Gates (Tool-Anbindung, Produktivsetzung, automatische Aktion), die Reviewbögen und die Eval-Templates stehen im Buch.

Dieses Paket gehört zum Buch „Vibe Coding, KI-Engineering & Agenten-Orchestrierung”

https://www.amazon.de/dp/B0H8JL34PQ

tbai.cloud/buch-vibe-coding.html

Autor und inhaltlich verantwortlich

Thorsten Behder

Weilheimer Str. 21A

81373 München

thorsten@behder.de

Gestaltung und Layout: tbai.cloud