SYNONET · WERKSTATT FÜR LOKALE KI · BLOG

Spec-driven Development — 10 einsatzbereite Prompts

Passend zum Blogbeitrag „Spec-driven Development: Der Plan, der dem Code vorausgeht". Die folgenden Prompts unterstützen Teams dabei, KI-Coding-Agenten mit klaren Spezifikationen zu führen — vom ersten Entwurf bis zur Qualitätskontrolle. Alle Prompts sind auf Deutsch, direkt kopierbar und für gängige Coding-Agenten gedacht.

Stand: 30.09.2026 · Erstellt von Synonet · Alle Prompts frei zur eigenen Nutzung

1 · Spec-Generator für ein neues Feature

Einstieg

Startet ein Feature sauber: aus einer groben Absicht wird eine strukturierte Spezifikation.

PROMPT:

```
Du bist Spec-Engineer. Formuliere für das folgende Feature eine maschinenlesbare
Spezifikation im Stil von Spec-driven Development.
Anforderungen an die Spec: High-Level-Vision zuerst, dann Details in domänennaher
Sprache (kein Tech-Jargon). Nutze Pre- und Postconditions sowie Invarianten.
Definiere klare Constraints und Grenzen (was der Agent nie ändern darf). Ende mit
Akzeptanzkriterien im Given-When-Then-Format.
Feature: [Hier das Feature beschreiben – 2 bis 3 Sätze reichen]
Frage gezielt nach, was Dir an Kontext fehlt, bevor Du die Spec ausgibst. Maximal 1
Rückfrage auf einmal.
```

2 · Anforderungen aus Kundendialog extrahieren

Analyse

Wandelt ein unstrukturiertes Kundengespräch in prüfbare Anforderungen um.

PROMPT:

```
Extrahiere aus dem folgenden Dialog alle Anforderungen an das zu bauende System.
Erstelle daraus eine Liste im Format: ID, Anforderung (ein Satz, messbar), Quelle
(Zitat), Priorität (Muss/Soll/Kann). Fasse zusammengehörige Punkte zusammen und
markiere Widersprüche oder offene Fragen explizit. Erfinde nichts, was nicht im
Dialog steht.
Dialog: [Kundentext einfügen]
```

3 · Akzeptanzkriterien in Given-When-Then

Spezifikation

Macht Ergebnisse deterministisch prüfbar — die Basis jeder guten Spec.

PROMPT:

```
Übersetze die folgenden Akzeptanzkriterien in das Given-When-Then-Format. Jeder
Testfall muss mit einem einzigen Satz pro Zeile auskommen (Given …, When …, Then …).
Vermeide vage Wörter wie "schnell", "einfach" oder "gut". Wenn ein Kriterium nicht
testbar ist, sage es und schlage eine testbare Alternative vor.
Kriterien: [Hier die gewünschten Kriterien eintragen]
```

4 · Domänensprache statt Tech-Sprech

Qualität

Sorgt dafür, dass die Spec in der Sprache des Fachbereichs steht, nicht in Implementierungsdetails.

PROMPT:

```
Review die folgende Spezifikation und formuliere sie in domänennaher Sprache um:
Ersetze technische Implementierungsdetails (Framework-, Datei- oder Klassenbegriffe)
durch Begriffe aus dem Fachbereich. Behalte Deutsch bei. Kennzeichne Fachbegriffe,
für die es noch keinen definierten Begriff gibt. Liefere am Ende eine kleine
Glossary-Tabelle (Begriff, Bedeutung, Kommentar).
Spezifikation: [Spec einfügen]
```

5 · Boundaries definieren (Always / Ask / Never)

Sicherheit

Gibt dem Agenten klare Leitplanken, bevor er überhaupt Code anfasst.

PROMPT:

```
Erstelle für das Projekt ein Drei-Stufen-Boundary-System:
- ALWAYS DO: Aktionen, die der Agent ohne Rückfrage ausführen darf.
- ASK FIRST: Aktionen, bei denen er vorher explizit nachfragen muss.
- NEVER DO: Verbote, die niemals umgangen werden dürfen (Dateien, Berechtigungen,
Daten).
Nutze dafür diesen Kontext: [Projektbeschreibung, Teamgröße, kritische Systeme].
Halte die Regeln kurz und präzise – wenige, klare Regeln schlagen eine lange Liste.
```

6 · Spec-Review gegen Spec Drift

Wartung

Erkennt, wo Spezifikation und implementierter Code auseinanderlaufen.

PROMPT:

```
Vergleiche die folgende Spezifikation mit dem aktuellen Stand des Codes (ich gebe Dir beides unten). Liste alle Abweichungen nach Schweregrad auf: (a) Code entspricht Spec nicht, (b) Spec ist veraltet, (c) unklar, wer richtig liegt. Für jede Abweichung: betroffene Stelle, mögliche Ursache, Empfohlene Korrektur (Code oder Spec). Sei streng – kleine Abweichungen wachsen zu Spec Drift.
Spec: [Spec einfügen]  Code: [Codeauszug oder Repo-Pfad einfügen]
```

7 · Spec-as-Source: Tests & Doku generieren

Automatisierung

Nutzt die Spec als einzige Wahrheit und erzeugt daraus Tests und Dokumentation.

PROMPT:

```
Generiere aus der folgenden Spezifikation (a) eine Testliste in Given-When-Then-Form und (b) eine Kurzdokumentation für Anwender. Beide müssen sich 1:1 auf die Spec beziehen: Jede Test-Zeile und jede Doku-Aussage muss auf eine Spec-Anforderung zurückführbar sein (referenziere die IDs). Markiere Stellen, für die die Spec keine eindeutige Aussage liefert, als offen.
Spec: [Spec einfügen]
```

8 · Agent-Readiness-Check der Spec

Prüfung

Prüft, ob eine Spec reif genug ist, an einen Coding-Agenten übergeben zu werden.

PROMPT:

```
Bewerte die folgende Spec auf einer Skala von 1–5 in fünf Dimensionen: Eindeutigkeit, Testbarkeit, Vollständigkeit, Grenzen (Boundaries), Freiheit von Widersprüchen. Nenne für jede Dimension die drei größten Risiken konkret. Beende mit einer Empfehlung: "reif für den Agenten" oder "noch überarbeiten" – und den wichtigsten drei Änderungen zuerst.
Spec: [Spec einfügen]
```

9 · Retro für die nächste Iteration

Prozess

Lernt aus der letzten SDD-Runde und verbessert die nächste.

PROMPT:

```
Du bist Retro-Moderator. Aus der letzten Entwicklungsrunde gebe ich Dir: (1) die Spec, (2) die Erkenntnisse aus Review und Tests, (3) die Fehlerliste. Analysiere ehrlich: Wo hat die Spec den Agenten fehlgeleitet? Wo fehlten Kriterien? Was davon lag an der Spec, was an der Umsetzung? Liefere maximal 5 konkrete Verbesserungen, die direkt die nächste Spec betreffen. Keine Allgemeinplätze — jede Empfehlung an einem Beispiel festmachen.
```

10 · Halluzinations-Check vor dem Commit

Qualität

Der letzte Filter: prüft generierten Code gegen die Spec, bevor er übernommen wird.

PROMPT:

```
Überprüfe den folgenden generierten Code gegen die Spezifikation. Suche gezielt nach: (a) erfundenen APIs oder Funktionen, die nicht im Projekt existieren, (b) Annahmen, die in der Spec nicht gedeckt sind, (c) stillen Abweichungen von Constraints. Für jeden Fund: Stelle, Problem, Schweregrad, konkrete Korrektur. Wenn der Code sauber ist, sage explizit "kein Befund". Keine Falschmeldungen.
Spec: [Spec einfügen]  Code: [generierter Code einfügen]
```