Att skriva prompts som får en LLM att bygga din mjukvara.

BY **MARC GLOOR** · SIDA 1 AV 2 — NIO GRUNDREGLER OCH REFERENSMALLEN

Hur väl en modell kan översätta en idé till fungerande kod hänger nästan helt på hur tydlig och välstrukturerad förfrågan är. Aktuell forskning inom kravhantering och kodgenerering pekar alla åt samma håll: enradsprompts ger enradsmjukvara. Det som följer är en koncentrerad samling praktiker — hämtade ur granskade studier och yrkesguider — för att förvandla krav till prompts som en LLM faktiskt kan utföra.

NIO GRUNDREGLER

Från krav till kod, från terminal till webbläsare — praktikerna som lyfter resultatet från prototyp till produktion.

1 Sätt roll och kontext först

Inled med en roll — ”Agera som en senior backend-ingenjör i en tjänsteorienterad arkitektur” — och ett stycke som namnger projektet, dess begränsningar och de standarder som måste följas. En roll smyger in prioriteringar (säkerhet, skalbarhet, idiomatisk stil) som modellen annars måste gissa sig till.

2 Skriv bestämt och specifikt

Byt ut ”bör” och ”kan” mot ”ska” och ”är skyldig att”. Vaga verb bjuder in vag kod. Namnge filer, klasser, metoder och befintliga mönster som den nya koden måste följa i stället för att beskriva dem abstrakt.

> ✗ ”Skriv en funktion som bearbetar användardata.”
> ✓ ”Lägg till *transformUser* i *UserProcessor.js*, i samma stil som *transformPaymentData*.”

3 Lås den tekniska miljön

Ange språk och exakta versioner, ramverk, vilka bibliotek som tillåts (och förbjuds), runtime, målplattform och kodkonventioner. Utan det faller modellen tillbaka på de mönster som dominerar dess träningsdata — och de strider ofta mot din verkliga kodbas.

4 Strukturera kraven som en numrerad lista

Bryt ned vad mjukvaran ska göra i åtskilda, numrerade och testbara punkter. Numrerade listor hanteras mer pålitligt än löpande text och tvingar dig att hantera otydligheter innan modellen gör det. Gruppera funktionella, icke-funktionella och gränssnittskrav var för sig.

5 Definiera in- och utdata samt gränsfall

Specificera I/O-format, pre- och postvillkor, felfall och minst två eller tre konkreta exempel — inklusive ett hörnfall. Few-shot-exempel slår genomgående abstrakta specifikationer, särskilt vid uppgifter där dataformen är central.

6 Använd stegvis promptning

Allt som är större än en enskild funktion: be inte om koden i ett enda svep. Led modellen genom en kedja: skärp kraven, härled en design, skriv testerna, skriv koden. Det ena stegets utdata blir nästa stegs indata. Det liknar vattenfallsmodellen och ger dig en granskningsyta i varje steg.

7 Bygg in verifiering i prompten

Be modellen skriva tester före — eller parallellt med — implementationen och själv granska resultatet mot dina acceptanskriterier, med antagandena och de oadresserade gränsfallen listade. Tester är dina krav i körbar form; utan dem kan du inte avgöra om koden möter dem.

8 Matcha noggrannheten mot risken

Ett engångsskript tål en ”lat” prompt; ett betalningsflöde gör det inte. Kalibrera ansträngningen mot insatsen: prototyper med låg risk klarar sig med löst, utforskande promptande; kritiska funktioner motiverar ett fullt spec-före-kod-pass med bevarade krav, design och tester.

9 Ange gränssnittsparadigm — TUI eller GUI

Namnge gränssnittet uttryckligen och lås dess konventioner; annars väljer modellen standardvärden som sällan stämmer med din plattform.

> **TUI** bibliotek (textual / curses / prompt_toolkit); tangentschema (vi / emacs / namngivet); minsta bredd; --json för piping; svartvitt först.
> **GUI** designsystem; tillstånd (tomt, laddar, fel, klart); tillgänglighet (WCAG AA); kontrast; full tangentbordsparitet; microcopy i prompten.

REFERENSMALL · EN ÅTERANVÄNDBAR PROMPTSTOMME

Åttablocksstrukturen som håller i de flesta projekt.

```
ROLE          Du är en senior {språk}-ingenjör. Prioritera tydlighet, testbarhet, säkerhet.
CONTEXT       Projekt: {namn, enradsbeskrivning}. Kodbasstil: {mönster, filer att efterlikna}.
STACK         Språk {X.Y}. Ramverk {X.Y}. Tillåtna bibliotek: {…}. Förbjudna: {…}.
UI / UX        Paradigm: {TUI | GUI}.
              TUI — bibliotek {textual/curses/prompt_toolkit}; tangenter {vi/emacs/namngivet}; bredd ≥ {N} kolumner; --json för piping;
svartvitt först.
              GUI — designsystem {…}; tillstånd tomt/laddar/fel/klart; tillgänglighet {WCAG AA}; kontrast ≥ 4,5:1; microcopy här.
REQUIREMENTS  1. {ska} 2. {ska} 3. {får inte} … (numrerat, bestämt, testbart).
I/O & EDGES   Indata: {typer, format}. Utdata: {typer, format}. Gränsfall: {…}. Exempel: {…}.
PROCESS       Steg 1 — föreslå en funktionssignatur + docstring; vänta på mitt godkännande.
              Steg 2 — skriv tester som faller på kraven ovan.
              Steg 3 — implementera; granska sedan själv mot kraven och lista antaganden.
OUTPUT        Fullständiga filsökvägar före varje block. Produktionsklart, ingen avhuggning, inga platshållare.
```

UNDVIK

Krav på en rad. **Att blanda vad och hur** i samma mening. **Att låta modellen** hitta på stacken. **Att godta första utkastet** som slutgiltigt. **Inga tester, ingen acceptans.**

7×

Fler små men betydelsefulla rättningar dyker upp när en prompt delas i **krav → design → tester → kod** i stället för att avfyras i ett svep. Stegvis promptning fångar dem *innan* de når kodbasen.

”Vi avråder från att förlita sig på enradskravsmetoder i LLM-baserad mjukvaruutveckling. I stället framhåller vi vikten av att lägga tid och möda på att formulera högkvalitativa, detaljerade krav.”

ARORA ET AL. · REQUIREMENTS ARE ALL YOU NEED (2024)

KÄLLOR · White et al., *ChatGPT Prompt Patterns for Software Engineering* (arXiv 2303.07839) · Arora et al., *Requirements are All You Need: From Requirements to Code with LLMs* (arXiv 2406.10101) · *Prompt Engineering for Requirements Engineering: A Literature Review & Roadmap* (arXiv 2507.07682) · Ullrich, Koch & Vogelsang, *From Requirements to Code: Developer Practices in LLM-Assisted SE* (arXiv 2507.07548) · *Guidelines to Prompt LLMs for Code Generation* (arXiv 2601.13118) · Mosofsky, *Spec-Then-Code* (GitHub) · Palantir, *Prompt-Engineering Best Practices*.

EXEMPEL · ETT KOMPLETT BYGGE FRÅN BÖRJAN TILL SLUT

Bygg en textredigerare. Kopiera mallen. Fyll i parenteserna.

*Två block nedan. **Det första** är den nakna åttablocksstommen från sida ett — klistra in den i den modell du föredrar och ersätt varje `{parentes}` med detaljerna i ditt projekt. **Det andra** är samma stomme, redan ifylld för ett litet men realistiskt mål: en enfönsters textredigerare i webbläsaren (en GUI). TUI-alternativet beskrivs i UI / UX-blocket i den nakna mallen. Använd något av blocken som referenslösning, eller kopiera och anpassa.*

BLOCK 1 Den nakna mallen — kopiera denna

ERSÄTT VARJE {PARENTES}

ROLE	Du är en senior {språk}-ingenjör. Prioritera tydlighet, testbarhet, {prioriteringar}.
CONTEXT	Projekt: {namn + enradsbeskrivning}. Kodbasstil: {mönster, filer att efterlikna}.
STACK	Språk {X.Y}. Ramverk {X.Y}. Tillåtna bibliotek: {…}. Förbjudna: {…}.
UI / UX	Paradigm: {TUI | GUI}. TUI — bibliotek {textual/curses/prompt_toolkit}; tangenter {vi/emacs/namngivet}; bredd ≥ {N} kolumner; --json för piping; svartvitt först. GUI — designsystem {…}; tillstånd tomt/laddar/fel/klart; tillgänglighet {WCAG AA}; kontrast ≥ 4,5:1; microcopy här.
REQUIREMENTS	1. {ska} 2. {ska} 3. {får inte} … (numrerat, bestämt, testbart).
I/O & EDGES	Indata: {typer, format}. Utdata: {typer, format}. Gränsfall: {…}. Exempel: {…}.
PROCESS	Steg 1 — föreslå filstruktur + typer; vänta på godkännande. Steg 2 — skriv tester som faller på kraven. Steg 3 — implementera; granska själv; lista antaganden.
OUTPUT	Fullständiga filsökvägar före varje block. Produktionsklart, ingen avhuggning, inga platshållare.

BLOCK 2 Genomarbetat exempel — en textredigerare i webbläsaren

KOPIERA OCH ANPASSA

ROLE	Du är en senior TypeScript-/React-ingenjör. Prioritera tydlighet, testbarhet och tillgänglighet. Välj enkel, idiomatisk kod framför smarta abstraktioner.
CONTEXT	Projekt: **NotePad** — en enfönsters textredigerare som körs i webbläsaren. Stil: små, fokuserade komponenter, rena funktioner för textoperationer, inget globalt tillstånd. Följ mönstren i `src/components/Toolbar.tsx`.
STACK	TypeScript 5.4. React 18.3. Vite 5. CSS Modules. Tillåtna bibliotek: endast `react` och `react-dom`. Förbjudna: Redux, MobX, jQuery och alla rich-text-bibliotek (Quill, Slate, CKEditor). Editorn SKA vara en kontrollerad `<textarea>`, inte `contenteditable`.
UI / UX	Paradigm: **GUI** (webbapp i ett fönster). Design: minimal — systemtypsnitt, neutral palett, ingen utsmyckning utöver verktygsraden. Tillstånd: tomt dokument → centrerad platshållare ”Börja skriva eller öppna en fil”; laddar → spinner i verktygsraden; fel → toast i toppen, kan stängas; klart (sparat) → ”Sparad”-bricka i 2 s. Tillgänglighet (WCAG AA): full tangentbordsparitet, inga åtgärder bara för mus; `<textarea>` märkt; verktygsknappar med `aria-label`; synlig fokusring; kontrast ≥ 4,5:1; respekterar `prefers-reduced-motion`. Microcopy: Spara-knapp ”Spara .txt”; statusrad ”Ord: {n} · Tecken: {n}”; stängvarning ”Det finns osparade ändringar. Vill du kasta dem?”
REQUIREMENTS	1. Visa dokumentet i en helfönsters `<textarea>`. 2. Verktygsraden SKA tillhandahålla: Ny, Öppna (.txt), Spara (laddar ner .txt), Ordräkning. 3. Öppna SKA läsa via FileReader och ersätta dokumentet. 4. Spara SKA utlösa en nedladdning av dokumentet som UTF-8-text. 5. Ordräkningen SKA uppdateras vid varje tangenttryck och visas i en statusrad. 6. Dokumentet SKA sparas till localStorage vid varje ändring och återställas vid laddning. 7. Ctrl/Cmd+S SKA utlösa Spara; Ctrl/Cmd+O SKA öppna filväljaren. 8. Appen SKA varna vid stängd flik om det finns osparade ändringar (beforeunload).
I/O & EDGES	Indata: tangenttryck; .txt-filer upp till 5 MB. Utdata: en nedladdad .txt-fil, UTF-8. Gränsfall: tomt dokument vid första start; återställning av en 4 MB-fil från localStorage; inklistring av binärt skräp (får inte krascha); filnamn med mellanslag. Exempel: öppna `notes.txt` med innehållet ”hello\nworld” → textarean visar två rader, ordräkning = 2.
PROCESS	Steg 1 — Föreslå filstruktur (komponenter, hooks, utils) och TypeScript-typer för AppState. Vänta på mitt godkännande. Steg 2 — Skriv Vitest-tester som täcker: ordräkningsfunktionen, härledningen av filnamn vid sparning, tur-och-retur till localStorage och hanteringen av tangentbordsgenvägar. Steg 3 — Implementera; granska själv mot de åtta kraven ovan och lista alla antaganden.
OUTPUT	Fullständiga filsökvägar före varje block (t.ex. `src/App.tsx`). Produktionsklart, ingen avhuggning, inga platshållarkommentarer. Vid ändringar i befintliga filer: returnera hela den uppdaterade filen.

Röd linje = den nakna mallen. Klistra in den som utgångspunkt och ersätt varje `{parentes}`. **Grön linje** = en helt ifylld instans. Samma stomme, gjord konkret för en liten produkt.

SÅ ANVÄNDER DU DEN **1.** Kopiera block 1 till chatten. **2.** Ersätt varje `{parentes}` med ditt projekts detaljer — block 2 visar den detaljnivå som fungerar. **3.** Skicka prompten och vänta vid **steg 1** på modellens föreslagna filstruktur innan du godkänner steg 2 och 3. **4.** Behandla resultatet som ett utkast: granska antagandelistan, kör testerna och iterera sedan.

KOMPLEMENT TILL SIDA 1 · Åttablocksstommen vilar på samma källor som anges där — främst Arora m.fl. (2024) om stegvis promptning, Mosofskys *Spec-Then-Code* om kalibrering av noggrannhet mot risk samt de empiriska riktlinjerna i Mastropaolo m.fl. (2026). Målet ”bygg en textredigerare” är illustrativt; samma struktur fungerar för ett CLI-verktyg, en mikrotjänst eller en datapipeline.