Vibe Coding Codex — Hoofdstukken 1 t/m 4 (Boekversie)

> **Context**: Dit document bundelt de complete **boekversie** van de Vibe Coding Codex, **Hoofdstuk 1 t/m 4**.

> Focus: professioneel toepasbaar, strak gestructureerd, zonder chatruis.

> Taal: **Nederlands**, met Engelstalige citaten waar relevant.

> Doel: direct inzetbaar in Cursor/ChatGPT/Claude, en bruikbaar als werkhandboek.

Inhoud

Hoofdstuk 1 — De Fundering van Vibe Coding

Vibe coding is **intuïtief bouwen met AI**, maar altijd gegrond in **intentie en structuur**.

Zonder fundament wordt AI een chaosversterker; met fundament wordt AI jouw **co■founder**.

1.1 Spec■Driven Development (Intentie vóór Code)

Kernidee: AI bouwt pas goed als jij het *waarom* en *wat* helder definieert.

Een **Mini■Spec** van 1 pagina volstaat.

Mini■Spec sjabloon

```markdown
# Mini■Spec — [Naam Feature / Module]

## ■ Probleem / Doel
Wat moet dit oplossen? Voor wie?

## ■ Gebruiker(s)
Wie gebruikt dit? Kennisniveau?

## ■ Functionaliteit (MVP)
Welke acties / functies zijn nodig?

## ■ Definition of Done
- [ ] Voorwaarde 1
- [ ] Voorwaarde 2
- [ ] Test(s) aanwezig

## ■ Testcases / Voorbeelden
Input → Verwacht resultaat
```

AI■prompt (Spec■review)

```

Ik werk spec■driven. Dit is mijn Mini■Spec:

[PLAATS SPEC]

Lees als lead engineer:

- Waar is de spec incompleet?

- Voorstel architectuur (modules/functies)

- Vraag mijn GO voor implementatie

```

1.2 Product Manager Framework (AI als Co■Founder)

Doel: laat AI eerst denken als **PM**: probleem, doelgroep, use■cases, kritische output, risico's.

PM■kader

```markdown

1) Probleem: wat lossen we op?

2) Doelgroep: voor wie?

3) Use■cases: 1–2 scenario's

4) Kritische output: wat móet bestaan?

5) Risico's/edge cases

```

AI■prompt (PM■modus)

```

Denk als Product Manager. Beantwoord 1■5 hierboven.

Geen code totdat we akkoord zijn.

```

1.3 Minimal Stack & MCP■discipline

Minimal Stack = maximaal 3 kerntools (AI coder, PM/tickets, optioneel design).

MCP (Model Context Protocol) discipline = één centrale "ruggengraat" (Linear/Notion) waar élke feature als ticket leeft.

Regels

- Niets bouwen buiten het ticket

- Context (spec + PM) hoort in het ticket

- AI verwijst naar ticket en vraagt om GO

AI■prompt (MCP■modus)

```

We werken MCP■driven. Ticket/spec:

[PLAATS TICKET/SPEC]
```

Lees context, stel ontbrekende vragen, doe een ontwerpvoorstel.

Bouw nooit buiten deze scope.

```

## Hoofdstuk 2 — De 21 Vibe Coding Tips (Volledig Uitgewerkt)

**Overzicht secties**

- 2.1 Flow & Intentie (Tips 4–7)
- 2.2 Architectuur & Structuur (Tips 8–12)
- 2.3 Refactor & Debug Discipline (Tips 13–15)
- 2.4 Versneller Prompts — Top 1% (Tips 16–18)
- 2.5 Self■Healing & Polishing (Tips 19–21)

> *Opmerking*: de nummering sluit aan op eerdere communicatie. Tip 1–3 zijn de fundamenten uit Hoofdstuk 1.

### 2.1 Flow & Intentie (Tips 4–7)

**Tip 4 — Conversational Warmup**

Warm AI op met context. Laat het **terugvertellen** wat jij bouwt, vóór code.

```

Contextbriefing: [project, doelgroep, huidige taak]

Begripscheck: Leg in je eigen woorden uit wat ik nu wil bouwen.

Wacht op mijn GO.

```

**Tip 5 — Break the Monolith**

Bouw in micro■loops: 1) scaffold 2) implement 3) validate.

```

STAP 1 alleen: maak skeleton (signatures + docstrings). Geen implementatie.

Vraag om GO voor STAP 2.

```

**Tip 6 — Chain of Thought (CoT)**

Forceer “hardop denken”: doel, randvoorwaarden, risico’s → dan pas code.

```

Denk hardop: doel, randgevallen, risico’s. Schrijf daarna pas de implementatie.

```

**Tip 7 — Confirm Understanding**

Laat AI de opdracht samenvatten en wachten op jouw GO.

```

Vat samen wat je gaat bouwen en waarom. Wacht op mijn GO.

```
```

2.2 Architectuur & Structuur (Tips 8–12)

Tip 8 — Architect First, Code Second

```
Architect■modus:
1) Mappenstructuur
2) Modules/functies
3) Dataflow
4) Risico’s
Geen code. Vraag om GO.
```

Tip 9 — Plan the Data Flow

```
Beschrijf: input → transformatie → output → logging/fouten.
Schrijf daarna pas code.
```

Tip 10 — Force Documentation Inline

Elke functie krijgt docstring (doel, randgevallen, returns).

Tip 11 — AI as Risk Engineer

```
Noem 3 risico’s/edge cases en geef mitigaties. Pas daarna implementatie.
```

Tip 12 — Modularity Pact

Eén verantwoordelijkheid per module (core vs ui vs logging strikt gescheiden).

```
Zorg dat core/ui/logging gescheiden blijven. Vraag GO als je wil combineren.
```

2.3 Refactor & Debug Discipline (Tips 13–15)

Tip 13 — Refactor■First, Implement■Second

```
Refactor deze code:
- Verwijder duplicatie
- Docstrings
- Leesbaarheid/modulariteit
Gedrag ongewijzigd.
```

Tip 14 — Debug Detective
```
Analyseer als debug■expert:
1) Mogelijke runtime fouten
2) Logische fouten/edge cases
3) Fixvoorstellen
Nog geen code.
```

Tip 15 — Add Test Harness
```
Schrijf unit tests (pytest-stijl):
- Positief
- Negatief
- Edge cases
```

2.4 Versneller Prompts — Top 1% (Tips 16–18)

Tip 16 — Parallel Prompts (A/B/C)
```
Geef 3 verschillende oplossingen:
A) Minimalistisch
B) Robuust/Redundant
C) Creatief/Out■of■the■box
```

Tip 17 — Synthesis Command
```
Combineer de beste elementen van A/B/C tot één finale versie.
```

Tip 18 — Auto■Polish Loop
```
Voer polish■pass uit:
- Leesbaarheid
- Naming■conventies
- Foutafhandeling
- Docstrings
Toon alleen verbeterde versie.
```

2.5 Self■Healing & Polishing (Tips 19–21)

Tip 19 — Self■Healing Code
```
Self■check:
- Ongebruikte variabelen
- Kwetsbare patronen
- TODO■fixes als comments
```
Vervolgens “auto■fix mode” op gemarkeerde punten.

Tip 20 — Self■Healing UI
```
UI■audit op: spacing/padding, typografie■hiërarchie, alignment, toegankelijkheid (contrast).
Lever verbeterde layout.
```

Tip 21 — Continuous Polish Loop
```
Evalueer codekwaliteit:
- Naamgeving
- Structuur
- Generaliseerbaarheid
Pas verbeteringen toe zonder scope te wijzigen.
```

Hoofdstuk 3 — Vibe Design & UI Blueprint

Doel: zonder designer toch een **professionele, consistente UI** leveren.

3.1 The Vibe Design Blueprint (4 zones)
1) **Header/Hero** — context & doel (wat is dit, voor wie?)
2) **Core Action Zone** — inputs & primaire acties
3) **Feedback Zone** — resultaten, validatie, errors
4) **Trust/Info Support** — hulpteksten, status, links

Prompt
```
Ontwerp een UI met 4 zones (Header, Actie, Feedback, Info).
Geef 3 variaties (A minimalistisch, B dashboard, C conversational).
Geen code, alleen layout.
```

3.2 De 3■Varianten Methode

A (minimalistisch), B (structureel), C (creatief). Daarna **synthese** (Tip 17).

3.3 Polish & Heuristic Review

Heuristieken: **Contrast, Alignment, Consistency, Feedback**.

```

UI■audit: spacing/padding consistentie, typografie■hiërarchie, contrast.

Verbeter alleen visuele structuur. Lever verbeterde variant.

```

3.4 Light Design System (Tokens)

Maak **design tokens** die AI overal toepast (kleur, typografie, spacing, radii).

Prompt

```

Genereer een Light Design System (tokens):

- Kleuren: primary/secondary/error/background

- Typografie: h1/h2/body/caption

- Spacing: sm/md/lg

- Component style: buttons/cards

Lever tabel + code■constanten.

```

3.5 Visual Prompt Library (UI)

- **Layout Architect** (structuur in tekst/ASCII)

- **UX Copywriter** (microcopy, foutmeldingen)

- **Visual Polish** (consistentie & toegankelijkheid)

- **Component Generator** (herbruikbare blokken)

- **Responsive/Mobile Check** (stacking, touch targets)

Hoofdstuk 4 — Scale, Ship & Product Lifecycle

AI inzetten om **gericht te leveren** (ship), kwaliteit te bewaren (regressie) en continu te verbeteren.

4.1 Roadmap Discipline

Niveaus: Epic → Feature → Task. Alles leeft in je MCP■systeem (Linear/Notion).

Prompt

```

Maak een roadmap voor deze feature:

- Epic (doel)

- Features (subdoelen)
```

- Tasks (concrete acties)
- Blockers

Lever als Markdown tabel.
```

### 4.2 Ship Loops

**Ship = Product.** Elke oplevering bevat changelog, release note, documentatie■update.

**Prompt**
```
Maak ship■afronding:
- Changelog (wat is nieuw)
- Release note (voor gebruiker)
- Documentatie update (README/handboek)
```

### 4.3 Regression Shield

Bescherm tegen “zombie bugs” met regressietests en audits.

**Prompt**
```
Regression■audit:
1) Zijn eerdere edge cases behouden?
2) Zijn oude tests nog geldig?
3) Maak regressietests waar nodig.
```

### 4.4 Continuous AI Co■Founder

Gebruik AI na oplevering voor **ideeën, simulaties, performance■checks, doc■reviews**.

**Prompt**
```
Beoordeel dit onderdeel als product■coach:
- Wat kan beter?
- Wat mist de gebruiker?
- Performance/UX■risico’s?
Stel 3 verbetercycli voor.
```

---

### Bijlage — Snelle Prompt Index

- **Spec■review** · **PM■modus** · **MCP■modus**
```

- **Architect■modus** · **Dataflow** · **Docstrings** · **Risicoanalyse** · **Modularity Pact**
- **Refactor■first** · **Debug detective** · **Unit tests**
- **Parallel prompts** · **Synthese** · **Auto■polish**
- **Self■healing code/UI** · **Continuous polish**
- **Roadmap** · **Ship loop** · **Regression shield** · **Product■coach**

> *Hoofdstuk 5 (Promptbibliotheek) wordt separaat aangeleverd wanneer compleet.*