BARCAMP · 11. DUBNA 2026

Vibe-coding? Hype-coding! AI Coding.

Za 2 měsíce víc features než za 2 roky. Bez psaní kódu.

Pavol Hejný · Promptbook

Vibe coding, vibe coding, vibe coding.

Za poslední rok se z toho stalo buzzword. Všude.

Jedni jsou iritovaní

- Záplava slop-code na GitHubu
- Junioři, kteří neumí debuggovat vlastní kód
- Pull requesty, kterým nikdo nerozumí
- „Ale vždyť to napsal Cursor"

Druzí jsou fascinovaní

- AI napíše celou feature za hodinu
- Prototyp za odpoledne
- „Proč bych psal boilerplate ručně?"
- 10× produktivita, tvrdí Twitter

Kdo má pravdu? Oba. A v tom je problém.

Naše čísla

Za 2 měsíce víc hodnotných features než za 2 roky.

A nenapsali jsme téměř **řádek kódu**.

Není to magie. Je to process.

Co se změnilo?

Před	Teď
Píšeme kód ručně	Agenti píšou, my řídíme
Review = čtení kódu	Review = kontrola výstupu + testy
1 feature / týden	Několik features / den
Bottleneck = psaní	Bottleneck = rozhodování

Hype-coding vs. Vibe-coding

Ne všechno, co se tváří jako AI coding, funguje.

❌ Hype-coding

- „Řeknu AI, ať to napíše, a pushnu"
- Žádné testy, žádné review
- Prompt → code → production
- Funguje na demu, padá v produkci
- **Výsledek:** technický dluh × 10

✅ Vibe-coding

- AI píše, člověk řídí a kontroluje
- PRD → agent → testy → review → merge
- Jasná pravidla, jasné guardrails
- Funguje v produkci, škáluje se
- **Výsledek:** 10× produktivita

Technika #1: PRD-first development

Nepíšete kód. **Píšete zadání.**

Každá feature začíná jako **Product Requirements Document**:

- Co má feature dělat
- Jaké jsou edge cases
- Jaké jsou acceptance criteria
- Co se nesmí rozbít

Agent dostane PRD a pracuje **autonomně**.

```
## Feature: Export do PDF

### Cíl

Uživatel může exportovat prezentaci do PDF.

### Acceptance criteria

- PDF obsahuje všechny slidy
- Funguje na prezentacích s 50+ slidy
- Export trvá < 10s

### Omezení

- Nesmí rozbít stávající HTML export
```

Technika #2: Git jako záchranná síť

Každá změna od agenta jde přes PR. Bez výjimky.

Branch per feature
Agent pracuje na vlastní větvi. Main je vždy čistý.

Atomic commits
Malé, popsané commity. Dají se revertovat jednotlivě.

PR = review gate
Žádný kód nejde do main bez lidského schválení.

> *Git není jen verzování. Je to vaše pojistka proti špatnému AI výstupu.*

Technika #3: Automatické testy

Testy nepíšete proto, že jste pečliví. **Píšete je proto, že agent potřebuje feedback loop.**

- Agent napíše kód **a testy zároveň**
- CI pipeline běží na každém commitu
- Červený build = agent to musí opravit
- Zelený build = review může začít

Bez testů je AI coding ruská ruleta.

Naše CI pipeline:

1. Agent pushne kód
2. Lint + type-check
3. Unit testy
4. Integration testy
5. ✅ Zelená → PR ready for review
6. ❌ Červená → agent dostane error log a opravuje

Technika #4: Pravidla pro agenty

Agent bez pravidel je junior bez mentora. **Dáte mu kontext, dostanete kvalitu.**

```
# .instructions.md

## Coding standards

- TypeScript only, strict mode
- No any types
- All public functions must have JSDoc
- Max file length: 300 lines

## Architecture

- Prefer composition over inheritance
- Use dependency injection
- No circular imports
```

```
## Testing

- Every feature needs unit tests
- Test edge cases explicitly
- Mock external dependencies

## Git

- Conventional commits
- One feature per branch
- Squash before merge

## Forbidden

- No console.log in production
- No hardcoded secrets
- No direct DOM manipulation
```

Technika #5: Code review agentího kódu

Review AI kódu je **jiné** než review lidského kódu.

Na co se díváte:

- Dává to smysl z business pohledu?
- Jsou testy smysluplné, nebo jen „pro zelenou"?
- Nezavedl agent zbytečnou abstrakci?
- Jsou edge cases pokryté?
- Neporušil pravidla z `.instructions.md`?

Na co se nedíváte:

- Formátování (to řeší linter)
- Naming conventions (to řeší pravidla)
- Import order (to řeší tooling)

> *Neřešíte jak to vypadá. Řešíte jestli to funguje a jestli to dává smysl.*

Technika #6: Prioritizace požadavků

Nedávejte agentovi všechno najednou. **Dávejte mu jednu věc po druhé.**

Backlog management

- Každý task má jasnou prioritu
- Agent pracuje vždy na jednom tasku
- Hotový task → review → další task
- Paralelizace = více agentů, ne větší tasky

Velikost tasku

Typ	Velikost	Příklad
Ideální	1–4 hodiny	Nový endpoint + testy
Přijatelný	4–8 hodin	Refactor modulu
Riskantní	8+ hodin	Přepis celé komponenty

Čím menší task, tím lepší výstup.

Workflow: Jak to vypadá v praxi

1. ZADÁNÍ

Napíšete PRD s acceptance criteria.

2. AGENT PRACUJE

Vytvoří branch, píše kód, píše testy, commituje.

3. CI/CD

Automaticky běží testy, lint, type-check.

4. REVIEW

Vy kontrolujete výstup. Business logika, edge cases.

5. MERGE

Schváleno → merge do main. Deploy.

6. ITERACE

Feedback → nový task → opakujte.

Co nefunguje

Buďme upřímní. AI coding není perfektní.

Kde agenti selhávají:

- Velké refactory přes mnoho souborů
- Architektonická rozhodnutí
- Debugging edge cases bez dobrého error logu
- Práce s legacy kódem bez dokumentace
- Cokoliv, co vyžaduje hluboký doménový kontext

Kde agenti excelují:

- Nové features s čistým zadáním
- Boilerplate a CRUD operace
- Testy a test coverage
- Migrace s jasným vzorem
- Dokumentace
- Malé, dobře definované tasky

Shrnutí: 6 pravidel pro AI coding v produkci

1. PRD first
Nepíšete kód, píšete zadání.

2. Git vždy
Branch, PR, review. Bez výjimky.

3. Testy povinně
Bez testů = bez merge.

4. Pravidla pro agenty
Kontext a guardrails v `.instructions.md`.

5. Review = business logika
Ne formátování, ale smysl.

6. Malé tasky
1–4 hodiny. Jeden agent, jeden task.

Pro koho je to určené?

- **Vývojáři** – naučte se řídit agenty místo psaní boilerplate
- **Majitelé firem** – shipujte 10× rychleji se stejným týmem
- **Tech-leadi** – nastavte proces, který škáluje
- **Product owneři** – PRD je teď vaše nejdůležitější práce

> *Budoucnost programování není psaní kódu.*
>
> *Je to řízení agentů, kteří kód píšou za vás.*
>
> *Ale jen pokud máte process, který to udrží pod kontrolou.*

Díky za pozornost!