Business Central v době AI

Příručka pro vývojáře a konzultanty

UKÁZKOVÝ VÝBĚR

Miloš Mikulášek

2026

Business Central v době AI

Příručka pro vývojáře a konzultanty

Miloš Mikulášek

Vydání 2026 · Verze 1.0

Toto je ukázkový výběr

Čtete ukázkový výběr z knihy **Business Central v době AI — Příručka pro vývojáře a konzultanty**.

Kompletní kniha obsahuje **47 kapitol v 6 modulech** pokrývajících vše od prompt engineeringu přes Azure OpenAI integraci v AL kódu až po RAG, AI agenty a MCP servery.

Tato ukázka obsahuje 6 vybraných kapitol — po jedné z každé části knihy — aby vám dala představu o stylu, hloubce a praktičnosti obsahu.

Kompletní obsah knihy:

Kompletní knihu si můžete zakoupit na **leanpub.com/AI4BC-DEV**.

Předmluva

Tato kniha vznikla z jednoduché frustrace: příliš mnoho schopných Business Central specialistů tráví příliš mnoho času na věcech, které by za ně mohla dělat umělá inteligence.

Viděl jsem BC vývojáře, kteří stráví hodiny hledáním správné AL syntaxe pro věc, kterou by Claude Code napsal za třicet vteřin. Konzultanty, kteří ručně přepisují zápisky ze schůzky do funkční specifikace, přestože by AI zvládla první draft za dvě minuty. Freelancery, kteří odmítají projekty, protože na dokumentaci nemají čas — přestože AI by jim ten čas dala zpět.

Zároveň jsem viděl druhou skupinu lidí. Ti, kteří AI přijali a integrovali do svého pracovního postupu. Píší kód 3–5× rychleji. Specifikace, které dřív trvaly den, teď trvají hodinu. Zákazníkům nabízejí věci, které jejich konkurenti ještě ani nevidí.

Tato kniha je pro první skupinu, která se chce stát druhou.

O autorovi

Miloš Mikulášek pracuje s NAV a Business Central více než dvacet let — jako vývojář, školitel a člověk, který prošel každou fází ekosystému od prvních verzí Navision po dnešní cloudový BC. Vedle AL ovládá SQL na úrovni, která mu umožňuje řešit problémy tam, kde standardní BC nástroje nestačí.

Umělou inteligenci přijal jako přirozené rozšíření svého nástrojového arsenálu. Nezajímal ho hype — zajímalo ho, co AI skutečně dokáže v prostředí, které zná do hloubky. Výsledkem je tato kniha — *Business Central v době AI* — a komunita kolem ní.

Jeho přístup je přímý: žádné sliby, které AI nesplní, žádné zkratky tam, kde je potřeba skutečná znalost systému. Jen konkrétní nástroje a reálné příklady z praxe.

Najdeš ho na **ai4bc.dev** nebo na **milos.mikulasek@ai4bc.dev**.

Lekce 0.0: Proč AI mění ERP svět (a proč to není hype)

Před dvěma lety jsem na jedné BC konferenci stál u kávy s kolegou vývojářem. Byl zkušený — 12 let v BC, znalost NAV od verze 3.60. A říkal: „AI? To je jen hype. Za rok o tom nikdo nebude mluvit."

Dnes ten člověk pracuje ve firmě, která mu nařídila „implementovat AI do BC do konce roku." A on stojí před tabulí plnou post-itů a neví kde začít.

Nechci, aby se ti tohle stalo.

Tohle není lekce o tom, jak AI změní vše a jak musíš okamžitě změnit vše, co děláš. Tahle lekce ti dá kontext. Realistický, praktický kontext. Co se skutečně mění, proč to ovlivňuje práci BC vývojářů a konzultantů specificky, a proč je právě teď ten správný čas začít — ne proto, že to říká marketing, ale proto, že data ukazují konkrétní věci.

Pojďme si být upřímní ohledně toho, jak AI obsah na internetu většinou vypadá.

Na jedné straně jsou nadšenci: „AI nahradí vývojáře!" „AI změní vše do 5 let!" „Prompt engineering je nová superschopnost!"

Na druhé straně jsou skeptici: „Jenom hype." „Halucinace jsou problém." „Moji zákazníci AI nechtějí."

A pak se vrátíš k práci, otevřeš VS Code, díváš se na AL kód, na codeunit, který musíš dopsat do pátku — a říkáš si: „Tak co teď? Jak mi tohle konkrétně pomůže?"

Žádný z těch článků ti neřekl, jak napsat codeunit s AI asistencí. Jak AI popsat BC architekturu. Jak pracovat s CLAUDE.md na reálném BC projektu. Jak integrovat Azure OpenAI do AL kódu tak, aby to bylo prakticky nasaditelné.

To je problém, který tato kniha řeší. A tahle lekce je začátek — dám ti framework, který ti pomůže pochopit, kde jsme, kam jdeme a co to znamená konkrétně pro tebe.

Co se skutečně mění — data, ne hype

Začneme čísly. GitHub provedl studii produktivity vývojářů s GitHub Copilotem — zkoumal přes 95 vývojářů na konkrétních úkolech. Zjistil průměrné zrychlení o 55 % u rutinních programátorských

úkolů. 55 procent. To není zanedbatelné číslo.

McKinsey ve své studii z roku 2023 uvádí, že AI může automatizovat 30 až 50 procent času, který vývojáři tráví psaním kódu. A to mluvíme o úkolech jako boilerplate kód, dokumentace, testování, code review.

Ale pozor — tyhle studie jsou na obecném kódu. JavaScript, Python, Java. Jak je to s AL a Business Central?

Proč BC vývojáři mají výhodu i nevýhodu

Pojďme být konkrétní. Business Central má v kontextu AI specifické postavení.

Nevýhoda: AL není v tréninkových datech AI modelů zastoupen tak dobře jako hlavní programovací jazyky. Pokud se zeptáš AI „napiš mi AL codeunit pro X" bez kontextu, dostaneš výsledek, který bude vypadat syntakticky podobně, ale bude mu chybět znalost BC konvencí, event architektury, table relationships. AI nezná specifika BC jako DataPerCompany, jak fungují extensions vs. base app, jak správně subscribovat na eventy.

Výhoda: A tady přichází ta dobrá zpráva. Protože ty víš, co AL a BC znamenají, máš obrovskou výhodu oproti komukoli, kdo AI bez tohoto kontextu používá. Ty víš, co AI vygeneroval správně a co špatně. Ty rozumíš architektuře. Ty jsi expert v doméně — AI je tvůj nástroj.

> **Poznámka:** Dobrý BC vývojář s AI bude produktivnější než průměrný BC vývojář bez AI. A bude výrazně produktivnější než výborný obecný vývojář s AI, ale bez BC znalostí. Vaše doménová znalost je klíčová — AI ji zesiluje, nenahrazuje.

Tři vlny AI v ERP světě

Rád přemýšlím o tom, kde jsme, pomocí metafory tří vln. Pomůže ti pochopit, kde jsme teď a co přichází.

Vlna 1 — AI asistenti pro vývojáře. To jsou nástroje jako ChatGPT, Claude, GitHub Copilot. Pomáhají psát, vysvětlovat a ladit kód, generovat dokumentaci, odpovídat na technické dotazy. Tady jsme teď. Tahle vlna je reálná, dostupná, použitelná dnes. A přesně to pokrývá tato kniha v modulech 1 a 2.

Vlna 2 — AI přímo zabudovaná do produktu. Microsoft Copilot v Business Central, AI-powered vyhledávání, automatické návrhy pro uživatele, inteligentní workflows. Tady začínáme být. Microsoft to aktivně rozvíjí a v posledních verzích BC vidíme první funkce. Modul 3 knihy se na tohle zaměřuje.

Vlna 3 — Autonomní AI agenti. Systémy, které samy monitorují procesy, rozpoznávají anomálie, navrhují a provádějí akce v BC. Tahle vlna přijde v horizontu 2 až 3 let. V modulu 4 se o ní bavíme, protože chceme, abyste měli kontext — i když ji ještě plně implementovat nedokážeme.

Proč ti to říkám? Protože pochopit, kde na timeline jsme, ti pomůže nastavit správná očekávání. Vlna 1 a 2 jsou tady a fungují. Vlna 3 přijde — a ti, kdo se připravují dnes, budou připraveni zítra.

Co to znamená pro tvoji kariéru

Řeknu ti, co si myslím upřímně. Dobrá zpráva: Znalost BC a AL zůstává klíčová. AI ji v dohledné době nenahradí. Zákazník pořád potřebuje někoho, kdo rozumí jeho businessu, kdo mu navrhne řešení, kdo nese zodpovědnost za výsledek.

> ⚠ **Upozornění:** Vývojář, který AI nepoužívá, bude dělat za celý den to, co vývojář s AI udělá za dopoledne. To neznamená, že vás AI nahradí — ale může vás nahradit vývojář, který AI používá lépe.

To neznamená, že tě AI nahradí. Znamená to, že tě může nahradit vývojář, který AI používá lépe než ty. A ten vývojář nemusí být tak zkušený, jak jsi ty — protože AI mu pomáhá překlenout část rozdílu v zkušenostech.

Takže: Nejde o to, jestli AI používat. Jde o to, jak ho používat dobře. A to je přesně to, co se v této knize naučíš.

Shrnutí

Pojďme si shrnout, co jsme si dnes řekli.

Za prvé: AI v BC vývoji není hype. Jsou za tím reálná čísla — 55 % produktivita u GitHub Copilot studie, 30–50 % automatizovatelného času dle McKinsey.

Za druhé: BC vývojáři mají specifickou výhodu. Rozumíme kontextu, který AI sám nezná. Jsme experti v doméně — AI je nástroj, který tuhle expertízu zesiluje.

Za třetí: Jsme ve vlně 1 — AI asistenti jsou zde dnes, dostupní a použitelní. Vlna 2 (AI v BC produktu) začíná. Vlna 3 (autonomní agenti) přichází.

Za čtvrté a nejdůležitější: Nejde o otázku „jestli". Jde o otázku „jak dobře". A to je přesně to, na co se zaměřuje zbytek této knihy.

Konkrétní příklady: AI v BC praxi dnes

Abychom nebyli jen u čísel a studií, pojďme si ukázat konkrétní situace z praxe BC specialistů, kteří AI již používají.

Příklad 1: Vývojář a boilerplate kód

Vývojář dostane zadání: „Potřebujeme tabulku pro evidenci servisních smluv s poli pro zákazníka, datum začátku, datum konce, měsíční poplatek, stav a poznámku. K tomu stránku pro správu a codeunit pro automatickou kontrolu expirace."

Bez AI: Otevře VS Code, založí tabulku, pole po poli. Pak stránku, pole po poli. Pak codeunit. Celkem 45–60 minut rutinní práce, kde se nedělá žádné rozhodnutí — jen se píše standardní struktura.

S AI: Zadá prompt s požadavky, konvencemi (prefix, komentáře) a za 5 minut má kompletní tabulku, stránku i základ codeunitu. Dalších 15 minut stráví review a přizpůsobením. Celkem 20 minut. Úspora: 30–40 minut na jednom úkolu.

Příklad 2: Konzultantka a zákaznický workshop

Konzultantka má zítra discovery workshop u zákazníka. Má téma (úprava expedičního procesu), ale žádnou agendu. Dříve by strávila hodinu přípravou.

S AI za 15 minut: strukturovaná agenda workshopu, klíčové otázky seřazené podle priority, seznam dokumentů, které zákazník má přinést, a přehled typických problémů BC v této oblasti. Přijde na workshop připravená a zákazník to vidí.

Příklad 3: Freelancer a nabídka

Freelancer dostal poptávku na BC extension pro správu nájmů. Potřebuje napsat nabídku s odhadem, rozsahem a cenou.

S AI: zadá popis požadavků a AI rozloží projekt na části, odhadne hodiny pro každou část, identifikuje rizika a navrhne fáze implementace. Freelancer nabídku upraví, doplní svou sazbu a za 30 minut má profesionální dokument — místo 2 hodin přemýšlení a formátování.

Co tyto příklady mají společného

Ve všech třech případech AI nedělá rozhodnutí — člověk rozhoduje. AI dělá rutinní práci: strukturování textu, generování boilerplate kódu, formátování dokumentu. Člověk přidává expertní znalost: BC architektura, zákaznické specifika, cenová strategie.

Tohle je model, který funguje. A přesně tenhle model vás naučí zbytek knihy.

Vyzkoušejte si

Cvičení 1: Měření aktuálního stavu. Tento týden si poznamenejte 3 úkoly, které děláte, a kolik minut na nich strávíte. Na konci týdne budete mít benchmark pro porovnání s AI.

Cvičení 2: První kontakt s AI. Pokud jste AI ještě nezkusili: otevřte claude.ai nebo chatgpt.com a zadejte: „Jsem BC vývojář. Napiš mi jednoduchou AL proceduru pro kontrolu, zda zákazník má vyplněný email.“ Funguje to? Co byste změnili?

Prompt engineering — základy pro vývojáře

Dva vývojáři, stejný nástroj, stejný úkol. Jeden dostane použitelný kód za 2 minuty. Druhý dostane generický výsledek, stráví hodinu opravováním a nakonec řekne „AI nefunguje.“

Rozdíl není v nástroji. Je v tom, jak úkol zadali.

Viděl jsem to desítkykrát. Vývojář si zkusí Claude nebo ChatGPT, napíše „napiš mi codeunit pro faktury“, dostane obecný výsledek, přejde na StackOverflow a řekne si, že AI je přehnaný hype. Přitom jeho kolega ve vedlejší kanceláři — se stejným nástrojem, stejnou BC verzí — dostane přesně to, co potřebuje, na první pokus.

Prompt engineering zní akademicky, ale jde o jednoduchou věc: naučit se mluvit s AI tak, aby rozuměl tvé práci. A pro BC vývojáře existují konkrétní vzory, které fungují opakovaně.

Největší chyba, kterou vidím u vývojářů, kteří si vyzkouší AI a pak ho přestanou používat: zadávají úkoly jako Google search. Krátce, bez kontextu. „Napiš mi AL kód pro fakturu.“ „Jak opravit tuto chybu?“ „Udělej report pro zákazníky.“

AI je jiný nástroj než Google. Google hledá v existujícím obsahu — přesné znění dotazu mu pomáhá najít správnou stránku. AI generuje nový obsah na základě toho, co mu řekneš. Čím méně kontextu mu dáš, tím více si domýšlí — a domýšlí ne tvoji konkrétní situaci, ale průměrnou situaci ze svých tréninkových dat.

Pro BC to znamená konkrétní problém. Průměrná situace v tréninkových datech je Python, JavaScript, webové API. Ne AL, ne Business Central, ne tvůj konkrétní projekt se specifickými tabulkami a naming konvencemi. Proto výsledek bez kontextu vypadá genericky — protože AI nevěděl, že pracuje v BC kontextu, a vygeneroval to, co by vygeneroval pro průměrného vývojáře.

Řešení je přidat kontext.

Zákon č. 1: Kontext je vše

Začněme základním experimentem. Vezmu stejný úkol a zadám ho dvěma způsoby.

Špatný prompt: „Napiš mi codeunit pro kontrolu zboží.“

Dobrý prompt: „Jsem BC vývojář, pracuji v BC v27. Máme extension pro e-shop integraci. Potřebuji codeunit, který zkontroluje dostupnost zboží v tabulce Item (pole Available Inventory) pro seznam Item No. z JSON array. Pokud je dostupnost pod požadovaným množstvím, vrátí JSON response s chybovými položkami. Naming prefix ZZ_, komentáře česky.“

Viděl jsi rozdíl ve výsledcích? Druhý prompt dostane výsledek, který v projektu přímo použiješ. První výsledek budeš přepisovat dvacet minut — přejmenováváš proměnné, měníš naming, přidáváš BC specifické věci, které AI nezná, protože jsi mu o nich neřekl.

> **Tip:** Vždy začni se čtyřmi prvky kontextu: (1) role a prostředí — „Jsem BC vývojář v BC v27”, (2) co přesně chceš — „Potřebuji codeunit, který…”, (3) relevantní tabulky a objekty, (4) konvence projektu — prefix, jazyk komentářů, styl error handlingu. Tyto čtyři prvky dají AI základ pro přesný výsledek.

Zákon č. 2: Rozděl velký úkol na malé

AI je lepší na soustředěné, ohraničené úkoly než na velké monolity. Místo „napiš mi celý modul pro správu projektů s tabulkami, codeunity a pagesami“ dej AI jeden codeunit, jeden report, jeden page extension. Jeden úkol na jeden prompt.

Proč tohle funguje lépe? U malého, ohraničeného úkolu AI lépe pochopí, co chceš a kde jsou hranice výsledku. U velkého úkolu musí domýšlet mnoho věcí najednou — jaký je vztah mezi tabulka-

mi, jaké je rozhraní mezi codeunity, jak se pages provazují s logikou. Každé domýšlení je potenciální zdroj chyby nebo odchylky od toho, co chceš.

Prakticky to znamená: Nejdřív si rozpiš úkol na části. Třeba pět minut s tužkou na papír nebo v poznámkovém bloku. Co jsou samostatné celky? Co může stát samo? Pak každou část zadej AI zvlášť. Výsledek bude přesnější a snáze kontrolovatelný než jeden obří prompt.

Zákon č. 3: Iteruj, ne přepisuj od nuly

Když AI vygeneruje kód, který není přesný, nespouštěj nový prompt od začátku. Místo toho opravuj v té samé konverzaci. AI si pamatuje kontext celé konverzace — každou tvoji zprávu i každou svoji odpověď.

Příklad opravného promptu, který používám: „Tenhle kód vypadá dobře, ale mail sending používá starší přístup. Uprav to tak, aby používal Email codeunit ze standardní BC knihovny místo přímého SMTP."

Nebo: „Logika schválení je správná, ale chybí mi permission check. Přidej ověření, že volající uživatel má oprávnění 'Purchase Manager'."

Nebo ještě konkrétnější: „Přidám zákaznickému objektu nové pole ZZ_CustomField. Jak aktualizuješ proceduru UpdateCustomer, aby s tímto polem pracovala?"

Takhle iteruješ rychle a efektivně. Každá iterace vychází z předchozí práce — nepřepínáš kontext, nepíšeš znovu celý prompt, nemusíš AI znovu vysvětlovat, co projekt dělá. Průměrná konverzace pro nový codeunit trvá tři až pět zpráv, ne jednu. A to je normální — takhle to funguje správně.

Zákon č. 4: Ukaž příklad

Jedna z nejúčinnějších technik se jmenuje few-shot prompting — ukázat AI příklad toho, co chceš dostat. Místo opisování, co chceš, ukáž mu vzor.

Místo „napiš komentář k této proceduře" řekni: „Napiš komentář k téhle proceduře ve stylu tohoto příkladu: [vlož existující komentář z vašeho projektu, který se vám líbí]."

AI okamžitě pochopí styl, formát a úroveň detailu, které chceš. Nemusíš to popisovat slovy — stačí ukázat. Slova jako „stručný, ale informativní" jsou subjektivní. Příklad je objektivní.

Tohle funguje skvěle pro naming konvence, styl error messagí, formát dokumentace — cokoliv, kde „styl" je důležitý a obtížně se popisuje slovy. Vezmi si existující kód z projektu, který se ti líbí, vlož ho do promptu jako vzor, a AI se bude inspirovat jeho stylem.

Zákon č. 5: Ptej se na review, ne jen na generování

Toto je možná nejpodceňovanější způsob využití AI a přitom jeden z nejcennějších. Vývojáři si zvykli AI používat jako generátor kódu. Ale AI je stejně dobrý jako reviewer.

Místo „napiš mi kód" zkus: „Tady je můj kód. Najdi potenciální problémy a navrhni zlepšení."

Nebo před zahájením implementace: „Tady je specifikace od zákazníka. Na co bych si měl dát pozor při implementaci? Jaké edge cases mi mohly uniknout?" Tohle ti dá pohled, který nemáš sám — AI prošel obrovské množství kódu a ví, co se typicky pokazí v podobných situacích.

Nebo po napsání codeunitu: „Zkontroluj, zda jsem neudělal zbytečné databázové dotazy v loopech. Kde by mohl být performance problém?" AI identifikuje N+1 query problémy, chybějící SetLoadFields, místa, kde se stejná data čtou dvakrát.

AI jako reviewer je jiná role než AI jako generátor. Obě jsou velmi cenné. Vývojáři, kteří AI používají jen jako generátor kódu, přicházejí o polovinu hodnoty, kterou jim AI nabízí.

Shrnutí

Pět zákonů prompt engineeringu pro BC vývojáře.

Zákon jedna: Kontext je vše — vždy uveď roli, prostředí, tabulky a konvence. Bez kontextu AI generuje průměr, ne tvoji situaci.

Zákon dva: Rozděl velký úkol na malé části. Jeden úkol na jeden prompt — výsledky jsou přesnější a snáze kontrolovatelné.

Zákon tři: Iteruj ve stejné konverzaci místo začínání znovu. AI si pamatuje kontext — využij ho.

Zákon čtyři: Ukaž příklad — few-shot prompting funguje lépe než slovní popis pro věci jako styl a konvence.

Zákon pět: Používej AI jako reviewera, ne jen jako generátor kódu. Polovina hodnoty je v pohledu na tvůj kód z jiného úhlu.

Tyhle principy platí pro všechny nástroje — Claude, ChatGPT, Copilot. Jsou to principy jak komunikovat s AI, ne specifika jednoho nástroje.

Vyzkoušej si

Cvičení 1: Kontext experiment. Zadej AI úkol „Napiš AL kód pro validaci zákazníka" — nejdřív bez kontextu, pak s plným kontextem (BC verze, tabulky, konvence). Porovnej výsledky. O kolik řádků se liší?

Cvičení 2: Iterace vs. nový prompt. Nech AI napsat codeunit. V odpovědi najdi problém (vždy nějaký je). Zkus ho opravit dvěma způsoby: (a) nový prompt od nuly, (b) opravný prompt ve stejné konverzaci. Který je rychlejší a přesnější?

Cvičení 3: Few-shot prompting. Vezmi existující komentář ze svého projektu, který se ti líbí. Vlož ho do promptu jako vzor a požádej AI o komentáře ke třem dalším procedurám. Jsou konzistentní se vzorem?

AI pro funkční specifikace — od poznámek k dokumentu

Jana je BC konzultantka a každý týden napíše průměrně tři funkční specifikace.

Každá z nich trvá dvě až tři hodiny. Poznámky ze schůzky se zákazníkem, zápisek z emailu, pár telefonátů — a z toho musí vyrůst strukturovaný dokument, který vývojáři pochopí a zákazník podepíše. To jsou hodiny strávené strukturováním textu, formulováním vět a hlídáním konzistentní terminologie — práce, která má jasnou hodnotu, ale nevyžaduje žádnou kreativitu.

Loni v lednu Jana zkusila použít AI pro jednu z těch specifikací. Vzala hrubé poznámky ze schůzky — nebyly ani čisté, jenom bullet pointy — a dala je Claudovi s jednoduchým promptem. Zákazník byl výrobní firma, požadavek se týkal úpravy fakturace pro rámcové smlouvy, a Janiny poznámky nebyly delší než tři odstavce.

Výsledek ji překvapil. Dokument byl z 80 % hotový za 12 minut. Strukturovaný, v čistém Markdownu, se všemi sekcemi na správných místech. Zbývajících 20 % doplnila ona — businessový kontext, zákazníkova specifika, věci, které AI neznal, protože nebyly v poznámkách.

Celkem 25 minut místo dvou hodin. Tři specifikace týdně krát 90 minut úspory na každé — to jsou 4,5 hodiny zpátky každý týden. Hodiny, které Jana teď věnuje zákazníkům, strategickým rozhovorům a přípravě na projekty — ne dokumentaci.

Psaní funkčních specifikací má jeden zásadní problém: je to rutinní práce s vysokou hodnotou, ale nízkou kreativitou.

Struktura specifikace se opakuje stále dokola — záhlaví, popis aktuálního stavu, popis požadovaného stavu, business pravidla, edge cases, testovací scénáře. Tohle AI zvládne perfektně, protože se jedná o strukturované generování textu z daných vstupů.

Co AI nezvládne bez tebe: rozumět zákazníkově businessové kultuře. Vědět, proč zákazník tahle business pravidla má. Chytit neřečené předpoklady, které jsou v organizaci považovány za samozřejmé. To jsi ty.

AI dělá strukturu a formulace. Ty dodáváš businessový kontext a expertní úsudek.

Workflow: Od zákazníkových poznámek k specifikaci

Celý proces má tři kroky. Prvním je příprava vstupního materiálu, druhým použití šablony promptu a třetím iterace na slabých místech výstupu.

Krok 1: Připrav vstupní materiál

Nemusíš psát čistě. AI pracuje dobře i s hrubými poznámkami — fragmenty vět, bullet pointy, dokonce i nepřesné formulace. Důležité je, aby vstup obsahoval všechno podstatné. Zapíš si, co zákazník řekl, co je aktuální stav procesu, co zákazník chce změnit a jakékoli specifické podmínky nebo výjimky, které zmínil. Čím více vstupního materiálu dáš, tím lepší výstup dostaneš. AI z ničeho nevyčaruje kontext, který jsi mu nedal.

V praxi to vypadá tak, že hned po schůzce se zákazníkem vyhradíš pět minut a zapíšeš si klíčové body do textového souboru nebo Notes aplikace. Neformátuj, nepřepisuj — jen vylijíš, co si pamatuješ. Tohle je vstup pro AI. Je lepší mít hodně surových informací než málo čistých vět. Zkušenost z praxe říká, že i 150 slov hrubých poznámek stačí AI k tomu, aby vygeneroval solidní základ specifikace o 500 až 700 slovech.

Krok 2: Použij prompt šablonu pro specifikaci

Základní šablona, která funguje konzistentně dobře, vypadá takto:

Začneš tím, že AI řekneš, kdo jsi a co děláš: „Jsem BC konzultantka. Zákazník mi popsal tento požadavek na úpravu v Business Central." Pak vložíš svoje poznámky nebo zákazníkův email. A pak popíšeš přesně, co chceš na výstupu — strukturu specifikace, formát dokumentu a jazyk.

Konkrétně chceš záhlaví s názvem projektu, zákazníkem, autorem, datem a verzí. Potom popis aktuálního stavu, tedy jak to v BC funguje dnes. Pak popis požadovaného stavu. Business pravidla s podmínkami a logikou. Edge cases a výjimky. Dopad na ostatní části systému. A nakonec otevřené otázky — věci, které ještě musíš vyjasnit se zákazníkem.

Formát chceš Markdown, jazyk česky.

Proč tato šablona funguje: AI dostane jasný rámec. Ví, co chceš, v jakém pořadí a v jakém formátu. Výsledek je pak konzistentní a předvídatelný. Bez šablony dostaneš pěkně napsaný text, ale sekce nebudou vždy v pořadí, které vývojáři nebo zákazníkovi sedí.

Krok 3: Iteruj s AI na slabých místech

AI výstup není nikdy finální dokument. Je to solidní základ, který ty přetváříš do finálního dokumentu. Nejčastěji slabé místo je sekce edge cases — AI generuje obecné příklady, ale nezná specifika zákazníka.

Jak iterovat: Po prvním výstupu zapiš druhý prompt s konkréty: „Sekce edge cases v téhle specifikaci je příliš obecná. Zákazník má tyhle specifika: rámcové smlouvy mohou mít více fakturačních adres, zákazník požaduje separátní faktury pro každou pobočku, příležitostně posílá souhrnné faktury na žádost. Rozšiř sekci edge cases o tyto scénáře."

Výsledek: detailní, zákazníkovi přizpůsobené edge cases za dvě minuty.

Tip: Šablona záhlaví v CLAUDE.md

Pokud pracuješ stále s jedním zákazníkem nebo máš standardizované záhlaví pro celou firmu, přidej šablonu záhlaví do svého CLAUDE.md souboru. Jednoduše si tam poznamenej jméno zákazníka, verzi BC, své jméno a preferovaný formát. AI pak záhlaví vyplní automaticky správnými údaji při každém dalším promptu, aniž bys musel tato data opakovaně zadávat.

Co když AI udělá chybu?

AI občas vymyslí business pravidlo, které zákazník nezmínil. Nebo formuluje požadavek jinak, než jsi zamýšlel. Tyhle chyby odhalíš při čtení výstupu — a to je v pořádku. Proto specifikaci čteš celou, ne jen přijmeš slepě.

AI výstup vždy projdi. Ověř businessová pravidla. Doplň zákazníkova specifika, která nebyla ve vstupním materiálu. Odstraň sekce, které nedávají smysl. Z 80 % hotového dokumentu uděláš 100 % hotový dokument — a to je pořád o 75 % méně práce než psát specifikaci od nuly.

S praxí poznáš, kde AI bývá systematicky slabé — u konkrétního zákazníka to mohou být vždy edge cases, u jiného integrace s třetím systémem. Tyto oblasti začneš kontrolovat jako první a iterace se zkrátí na minuty.

Shrnutí

Workflow pro funkční specifikace s AI je jasný a opakovatelný.

Zaprvé: připrav vstupní materiál. Nemusí být čistý, ale musí být úplný. AI pracuje s tím, co dostane.

Zadruhé: použij šablonu promptu se strukturou specifikace a formátem výstupu. Výsledek je z 80 % hotový.

Zatřetí: iteruj na slabých místech — edge cases, zákazníkova specifika, věci, které AI neznal.

Úspora času je reálná: typická specifikace, která ti dříve trvala dvě hodiny, je s AI hotová za 25 až 30 minut.

Klíčový bod, který si odnes: AI nepíše specifikaci místo tebe. Ty pořád zůstáváš autorem a odpovídáš za obsah. AI se stará o strukturu, formulace a konzistenci — a tím ti uvolňuje kapacitu na to, v čem jako konzultant skutečně vynikáš: porozumění zákazníkovu businessu, odhalení neřečených předpokladů a expertní úsudek o tom, co implementovat a jak.

Praktický tip: hned po schůzce otevři Claude, vlož poznámky a nech ho generovat na pozadí, zatímco píšeš follow-up email zákazníkovi. Specifikace tě čeká hotová, když mail odešleš.

Case study: Konzultantka a specifikace pro výrobní firmu

Jana, BC konzultantka, zpracovává požadavky pro zákazníka — výrobní firmu s 200 zaměstnanci. Dříve psala specifikace 2–3 hodiny, teď s AI průměrně 35 minut.

Jeden konkrétní případ: zákazník chtěl úpravu expedičního procesu — automatické generování dodacích listů s QR kódy pro mobilní sklad. Janiny poznámky ze schůzky měly 12 bullet pointů. AI z nich vygeneroval 4stránkovou specifikaci za 8 minut. Jana doplnila zákazníkova specifika (typy QR kódů, layout dodacího listu) za dalších 15 minut.

Vývojář, který specifikaci implementoval, řekl: „Tohle je nejjasnější zadání, jaké jsem od konzultanta dostal." Protože AI strukturu neimprovizuje — drží se formátu, který vývojáři rozumí.

Vyzkoušej si

Cvičení 1: Vezmi poznámky z poslední schůzky se zákazníkem a vygeneruj specifikaci přes AI. Změř čas. Porovnej se svým obvyklým časem.

Cvičení 2: Nech AI vygenerovat sekci „Otevřené otázky" ze specifikace. Kolik otázek je skutečně relevantních? Kolik bys sám přehlédl?

Demo: Automatická kategorizace položek pomocí AI

Tenhle scénář se opakuje u mnoha zákazníků. Firma přechází na nový BC tenant nebo dělá datovou migraci ze starého systému. A výsledek je vždy stejný: v BC přistane tabulka Item s pěti sty nebo dvěma tisíci položkami bez kategorií. Konzultant dostane zadání: „Prosím kategorizujte to.“ A pak stráví týden ručním procházením položek, čtením názvů, přiřazováním kategorií. Jeden z konzultantů mi říkal, že to trvalo tři dny čistého času.

S AI tenhle proces zvládneš za dvacet minut.

Definice problému

Máme položky v BC. Každá položka má `No.`, `Description`, možná `Description 2`, a prázdné pole `Item Category Code`. Máme definované kategorie v tabulce `Item Category` — třeba ELEKTRO, NABYTEK, KANCELAR, NADRADI atd.

Ruční kategorizace selže ze tří důvodů. První: je to nudná práce, člověk po chvíli dělá chyby. Druhý: různí lidé kategorizují různě — jeden řekne, že tiskárna je ELEKTRONIKA, jiný KANCELAR. Třetí: při migraci je tlak na rychlost, kategorizace jde na konec a pak se nikdy neudělá pořádně.

AI tohle řeší. Model dostane název položky a seznam povolených kategorií, a vrátí nejlepší kategorii. Rychle, konzistentně, bez únavy. Na 80–90 % případů to funguje dobře — zbylých 10 % pak zkontroluj ručně, ale to je pořád obrovská úspora.

Architektura extension

Máme tři části:

Část první: Setup. Potřebujeme někde uložit Azure endpoint URL a API klíč. Vytvoříme setup tabulku `AI Categorization Setup` s těmito poli a stránku pro správce, kde to nastaví. Klíč ukládáme ideálně přes Isolated Storage, ne přímo do tabulky.

Část druhá: Logika volání. Codeunit `Item AI Categorizer`, který: 1. Načte seznam kategorií z tabulky `Item Category` 2. Pro každou nezakategorizovanou položku sestaví prompt 3. Zavolá Azure OpenAI přes `OpenAIHelper` codeunit 4. Zparsuje odpověď a zapíše kategorii zpět do položky

Část třetí: Spuštění. Report nebo page action, který celý proces spustí. Ideálně s progress barem, protože 500 položek může trvat pár minut.

Kód walkthrough

Začínáme s `OpenAIHelper.al`, který se stará o nízkoúrovňové volání API. Není v něm žádná byznys logika — jen čistá komunikace s Azure OpenAI.

Teď se podívejme na jádro aplikace — `ItemAICategorizer.al`:

```
namespace BCDemoAI;

codeunit 50102 "Item AI Categorizer"
{
    procedure CategorizeAllItems()
    var
        Item: Record Item;
        OpenAIHelper: Codeunit "OpenAI Helper";
        CategoryList: Text;
        Prompt: Text;
        SuggestedCategory: Text;
        ProcessedCount: Integer;
        TotalItems: Integer;
        ProgressDialog: Dialog;
    begin
        // Načteme seznam existujících kategorií pro prompt
        CategoryList := GetCategoryList();

        if CategoryList = '' then
            Error('Nejsou definovány žádné kategorie položek. Vytvořte kategorie v BC a zkuste znovu.')

        // Filtrujeme jen nezakategorizované položky
        Item.SetFilter("Item Category Code", '%1', '');
        Item.SetFilter(Description, '<>%1', '');

        if Item.IsEmpty() then begin
            Message('Všechny položky jsou již zakategorizovány.');
            exit;
        end;

        TotalItems := Item.Count();  // Spočítat jednou před smyčkou – volání Count() v každé iteraci b

        ProgressDialog.Open('Kategorizace položek pomocí AI...\Zpracovávám: #1####\Hotovo: #2#### / #3#

        Item.SetLoadFields(Description, "Item Category Code");
        if Item.FindSet() then
            repeat
                ProcessedCount += 1;
                ProgressDialog.Update(1, Item.Description);
                ProgressDialog.Update(2, ProcessedCount);
                ProgressDialog.Update(3, TotalItems);
```

```
                // Sestavíme prompt s názvem položky a seznamem kategorií
                Prompt := BuildCategorizationPrompt(Item.Description, CategoryList);

                // Zavoláme AI
                SuggestedCategory := OpenAIHelper.CallChatCompletionSimple(Prompt);

                // Validujeme, že vrácená kategorie existuje v BC
                SuggestedCategory := CleanAndValidateCategory(SuggestedCategory, CategoryList);

                // Zapíšeme výsledek
                if SuggestedCategory <> '' then begin
                    Item."Item Category Code" := SuggestedCategory;
                    Item.Modify(true);
                end;
            until Item.Next() = 0;

        ProgressDialog.Close();
        Message('Hotovo! Zpracováno %1 položek.', ProcessedCount);
    end;
```

Klíčová část je prompt engineering. Podívejme se na funkci `BuildCategorizationPrompt`:

```
local procedure BuildCategorizationPrompt(ItemDescription: Text; CategoryList: Text): Text
var
    PromptBuilder: TextBuilder;
begin
    PromptBuilder.Append('Jsi asistent pro kategorizaci skladových položek v systému Business Central.'
    PromptBuilder.Append(' Tvůj úkol je přiřadit JEDNU kategorii k následující položce.');
    PromptBuilder.Append(' Odpovídej POUZE kódem kategorie, nic jiného. Žádný text navíc.');
    PromptBuilder.AppendLine();
    PromptBuilder.Append('Dostupné kategorie: ');
    PromptBuilder.Append(CategoryList);
    PromptBuilder.AppendLine();
    PromptBuilder.Append('Položka k zařazení: ');
    PromptBuilder.Append(ItemDescription);
    PromptBuilder.AppendLine();
    PromptBuilder.Append('Kategorie:');

    exit(PromptBuilder.ToText());
end;
```

Pojďme si projít, proč je tento prompt napsaný právě takto.

„Odpovídej POUZE kódem kategorie, nic jiného.“ Tato instrukce je naprosto klíčová. Bez ní

ti AI vrátí věty jako „Na základě názvu položky doporučuji kategorii ELEKTRO, protože…". To pak musíš parsovat. S touto instrukcí vrátí jen „ELEKTRO". Čisté, jednoduché.

„Kategorie:" na konci bez hodnoty. Tohle je klasický completion pattern. AI „doplní" text za dvojtečkou. Funguje konzistentně.

Uvádíme seznam kategorií. Důležité! Pokud AI nezná tvé konkrétní kódy kategorií, vymyslí si vlastní. Vždy v promptu uveď, z čeho může vybírat.

Druhá kritická část je validace odpovědi:

```
local procedure CleanAndValidateCategory(AIResponse: Text; ValidCategories: Text): Text
var
    CleanedResponse: Text;
    ItemCategory: Record "Item Category";
begin
    // Odstraníme přebytečné mezery a nové řádky
    CleanedResponse := AIResponse.Trim();
    CleanedResponse := CleanedResponse.ToUpper();

    // Ověříme, že kategorie existuje v BC
    if ItemCategory.Get(CleanedResponse) then
        exit(CleanedResponse);

    // Pokud ne, zkusíme najít podobnou (AI někdy vrátí trochu jiný formát)
    ItemCategory.SetFilter(Code, '@' + CleanedResponse);
    if ItemCategory.FindFirst() then
        exit(ItemCategory.Code);

    // Pokud nic nevyhovuje, vrátíme prázdný řetězec (položka zůstane nezakategorizovaná)
    exit('');
end;
```

Upozornění: Nikdy slepě nevěř AI odpovědi. Vždy validuj vůči reálným datům v BC. Pokud AI vrátí kategorii, která neexistuje, raději necháme položku nezakategorizovanou, než zapíšeme nesmysl.

Výsledky a produkční aspekty

Z testování na 50 položkách: AI správně kategorizovala 44 — to je 88% úspěšnost bez jakéhokoliv doladění promptu. Zbylých 6 necháme na ruční kontrolu — ale těch 44 jsme nemuseli dělat ručně.

Než tohle nasadíte u zákazníka, pár věcí k zamyšlení:

Rate limiting. Pokud máš 2000 položek, 2000 API volání za sebou pravděpodobně narazí na rate limit Azure OpenAI. Řešení: přidej `Sleep(200)` mezi volání, nebo použij batch approach — pošli 10 položek v jednom promptu a požádej o výsledky jako JSON pole.

Batch prompting. Je to efektivnější. Jeden prompt „Kategorizuj těchto 10 položek, vrať JSON pole s kódy“ je levnější než 10 samostatných volání. Vyžaduje trochu složitější parsování, ale stojí to za to.

Auditní záznamy. Logujte, co AI navrhla a co bylo zapsáno. Zákazník se může chtít vrátit a přezkoumat rozhodnutí. Stačí jednoduchá logovací tabulka s datem, číslem položky a navrženou kategorií.

Lidská kontrola. Nedoporučuji nasadit bez review kroku. Lepší pattern je: AI zapíše kategorie do pomocného pole nebo staging tabulky, konzultant projde výsledky a potvrdí hromadně. Zákazník pak ví, že výsledky prošly lidskou kontrolou.

Kompletní kód extension

Pro úplnost uvádíme kompletní soubory, které potřebuješ pro fungující extension. Tyto soubory nejsou v předchozích sekcích, ale bez nich extension nespustíš.

app.json

```
{
  "id": "a1b2c3d4-e5f6-7890-abcd-ef1234567890",
  "name": "AI Item Categorizer",
  "publisher": "BCDemoAI",
  "version": "1.0.0.0",
  "brief": "Automatická kategorizace položek pomocí Azure OpenAI",
  "description": "Extension pro hromadnou kategorizaci položek pomocí AI.",
  "privacyStatement": "",
  "EULA": "",
  "help": "",
  "url": "",
  "contextSensitiveHelpUrl": "",
  "logo": "",
  "dependencies": [],
  "screenshots": [],
  "platform": "27.0.0.0",
  "application": "27.0.0.0",
  "idRanges": [
    { "from": 50100, "to": 50199 }
  ],
  "resourceExposurePolicy": {
    "allowDebugging": true,
    "allowDownloadingSource": true,
    "includeSourceInSymbolFile": true
  },
```

```
  "runtime": "14.0",
  "allowedExternalUrls": [
    "https://*.openai.azure.com"
  ]
}
```

Setup tabulka — AI Categorization Setup

```
namespace BCDemoAI;

table 50101 "AI Categorization Setup"
{
    Caption = 'Nastavení AI kategorizace';
    DataClassification = SystemMetadata;

    fields
    {
        field(1; "Primary Key"; Code[10])
        {
            Caption = 'Primární klíč';
            DataClassification = SystemMetadata;
        }
        field(10; "Endpoint URL"; Text[250])
        {
            Caption = 'Azure OpenAI Endpoint URL';
            DataClassification = SystemMetadata;

            trigger OnValidate()
            begin
                // Ověříme formát URL
                if ("Endpoint URL" <> '') and (not "Endpoint URL".StartsWith('https://')) then
                    Error('Endpoint URL musí začínat https://');
            end;
        }
        field(20; "Deployment Name"; Text[100])
        {
            Caption = 'Název deployment modelu';
            DataClassification = SystemMetadata;
        }
        field(30; "API Version"; Text[50])
        {
            Caption = 'Verze API';
```

```
            DataClassification = SystemMetadata;
            InitValue = '2025-01-01-preview';
        }
        field(40; "Max Tokens"; Integer)
        {
            Caption = 'Maximální počet tokenů odpovědi';
            DataClassification = SystemMetadata;
            InitValue = 100;
            MinValue = 10;
            MaxValue = 4000;
        }
        field(50; "Temperature"; Decimal)
        {
            Caption = 'Temperature (0.0–2.0)';
            DataClassification = SystemMetadata;
            InitValue = 0.1;
            MinValue = 0;
            MaxValue = 2;
        }
        field(60; "Batch Size"; Integer)
        {
            Caption = 'Počet položek v jednom API volání';
            DataClassification = SystemMetadata;
            InitValue = 1;
            MinValue = 1;
            MaxValue = 20;
        }
        field(70; "Delay Between Calls Ms"; Integer)
        {
            Caption = 'Prodleva mezi voláními (ms)';
            DataClassification = SystemMetadata;
            InitValue = 200;
            MinValue = 0;
            MaxValue = 5000;
        }
    }

    keys
    {
        key(PK; "Primary Key")
        {
            Clustered = true;
```

```
        }
    }

    procedure GetSetup()
    begin
        if not Get() then begin
            Init();
            if not Insert() then
                Get();
        end;
    end;

    procedure StoreApiKey(ApiKey: Text)
    begin
        IsolatedStorage.Set('AzureOpenAIKey', ApiKey, DataScope::Company);
    end;

    procedure GetApiKey(): Text
    var
        ApiKey: Text;
    begin
        if not IsolatedStorage.Get('AzureOpenAIKey', DataScope::Company, ApiKey) then
            Error('API klíč není nastaven. Nastavte ho na stránce Nastavení AI kategorizace.');
        exit(ApiKey);
    end;

    procedure HasApiKey(): Boolean
    begin
        exit(IsolatedStorage.Contains('AzureOpenAIKey', DataScope::Company));
    end;

    procedure GetFullEndpointUrl(): Text
    begin
        GetSetup();
        TestField("Endpoint URL");
        TestField("Deployment Name");
        exit("Endpoint URL" + '/openai/deployments/' + "Deployment Name" +
             '/chat/completions?api-version=' + "API Version");
    end;
}
```

Setup stránka

```
namespace BCDemoAI;

page 50101 "AI Categorization Setup"
{
    PageType = Card;
    SourceTable = "AI Categorization Setup";
    Caption = 'Nastavení AI kategorizace';
    ApplicationArea = All;
    UsageCategory = Administration;
    InsertAllowed = false;
    DeleteAllowed = false;

    layout
    {
        area(Content)
        {
            group(AzureOpenAI)
            {
                Caption = 'Azure OpenAI';

                field("Endpoint URL"; Rec."Endpoint URL")
                {
                    ApplicationArea = All;
                    ToolTip = 'URL vašeho Azure OpenAI resource, např. https://moje-ai.openai.azure.com
                }
                field("Deployment Name"; Rec."Deployment Name")
                {
                    ApplicationArea = All;
                    ToolTip = 'Název modelu deployment v Azure, např. gpt4o-mini-v1';
                }
                field("API Version"; Rec."API Version")
                {
                    ApplicationArea = All;
                    ToolTip = 'Verze Azure OpenAI API';
                }
                field(ApiKeyStatus; ApiKeyStatusTxt)
                {
                    ApplicationArea = All;
                    Caption = 'API klíč';
                    Editable = false;
                    StyleExpr = ApiKeyStyle;
```

```
            }
        }
        group(Parameters)
        {
            Caption = 'Parametry AI';

            field("Max Tokens"; Rec."Max Tokens")
            {
                ApplicationArea = All;
            }
            field(Temperature; Rec.Temperature)
            {
                ApplicationArea = All;
            }
            field("Batch Size"; Rec."Batch Size")
            {
                ApplicationArea = All;
                ToolTip = 'Kolik položek odeslat v jednom API volání (1 = jednotlivě)';
            }
            field("Delay Between Calls Ms"; Rec."Delay Between Calls Ms")
            {
                ApplicationArea = All;
                ToolTip = 'Prodleva mezi voláními v milisekundách (prevence rate limiting)';
            }
        }
    }
}

actions
{
    area(Processing)
    {
        action(SetApiKey)
        {
            Caption = 'Nastavit API klíč';
            Image = EncryptionKeys;
            ApplicationArea = All;

            trigger OnAction()
            var
                ApiKeyInput: Text;
            begin
```

```
                if Page.RunModal(Page::"AI API Key Input", ApiKeyInput) = Action::OK then begin
                    Rec.StoreApiKey(ApiKeyInput);
                    UpdateApiKeyStatus();
                    CurrPage.Update(false);
                end;
            end;
        }
        action(TestConnection)
        {
            Caption = 'Otestovat spojení';
            Image = TestReport;
            ApplicationArea = All;

            trigger OnAction()
            var
                OpenAIHelper: Codeunit "OpenAI Helper";
                Response: Text;
            begin
                Response := OpenAIHelper.CallChatCompletionSimple('Řekni "OK".');
                Message('Spojení funguje. AI odpověděla: %1', Response);
            end;
        }
    }
}

trigger OnOpenPage()
begin
    Rec.GetSetup();
    UpdateApiKeyStatus();
end;

var
    ApiKeyStatusTxt: Text;
    ApiKeyStyle: Text;

local procedure UpdateApiKeyStatus()
begin
    if Rec.HasApiKey() then begin
        ApiKeyStatusTxt := '✓ Nastaven';
        ApiKeyStyle := 'Favorable';
    end else begin
        ApiKeyStatusTxt := '✗ Není nastaven';
```

```
            ApiKeyStyle := 'Unfavorable';
        end;
    end;
}
```

Logovací tabulka pro audit

```
namespace BCDemoAI;

table 50102 "AI Categorization Log"
{
    Caption = 'Log AI kategorizace';
    DataClassification = CustomerContent;

    fields
    {
        field(1; "Entry No."; Integer)
        {
            Caption = 'Číslo záznamu';
            AutoIncrement = true;
        }
        field(10; "Item No."; Code[20])
        {
            Caption = 'Číslo položky';
            TableRelation = Item."No.";
        }
        field(20; "Item Description"; Text[100])
        {
            Caption = 'Popis položky';
        }
        field(30; "Suggested Category"; Code[20])
        {
            Caption = 'Navržená kategorie';
            DataClassification = CustomerContent;
        }
        field(40; "Applied Category"; Code[20])
        {
            Caption = 'Použitá kategorie';
            DataClassification = CustomerContent;
        }
        field(50; "Was Valid"; Boolean)
        {
```

```
            Caption = 'Byla validní';
        }
        field(60; "Processing DateTime"; DateTime)
        {
            Caption = 'Datum a čas zpracování';
        }
        field(70; "Tokens Used"; Integer)
        {
            Caption = 'Spotřebované tokeny';
        }
    }

    keys
    {
        key(PK; "Entry No.")
        {
            Clustered = true;
        }
        key(K1; "Item No.")
        {
        }
        key(K2; "Processing DateTime")
        {
        }
    }
}
```

Vyzkoušej si

Cvičení 1: Spusť kategorizaci. Nainstaluj extension na dev sandbox, vytvoř 10 testovacích položek bez kategorií (např. „Tiskárna HP LaserJet", „Kancelářská židle ergonomická", „Šroubovák křížový PH2"). Nastav Azure OpenAI endpoint v setup stránce a spusť kategorizaci. Kolik položek AI správně zařadila?

Cvičení 2: Uprav prompt. V proceduře `BuildCategorizationPrompt` změň instrukci: místo „Odpovídej POUZE kódem kategorie" zkus „Odpovídej kódem kategorie a jednou větou zdůvodnění ve formátu KATEGORIE: důvod". Jak se změní výstup? Co musíš upravit v parsování?

Cvičení 3: Batch přístup. Uprav codeunit tak, aby posílal 5 položek v jednom promptu a očekával JSON pole odpovědí. Tip: prompt „Kategorizuj těchto 5 položek, vrať JSON array: [{'item': 'popis', 'category': 'KOD'}]". Jak se změní rychlost a cena?

Shrnutí

Za prvé: Prompt engineering rozhoduje. Instrukce „odpovídej pouze kódem kategorie“ a seznam povolených hodnot jsou zásadní pro použitelné výsledky.

Za druhé: Vždy validujte AI odpověď vůči BC datům. Nikdy slepě nepište to, co AI vrátila, bez kontroly, že to dává smysl v kontextu tvého systému.

Za třetí: Zvažte batch approach pro velké objemy — místo jednoho volání na položku pošlete více položek najednou. Levnější a rychlejší.

Za čtvrté: Kompletní extension potřebuje nejen business logiku, ale i setup tabulku s Isolated Storage pro API klíč, konfigurační stránku pro správce a logovací tabulku pro audit. Tyto „nudné“ části jsou v produkci stejně důležité jako samotný AI kód.

Toto je ukazka. V kompletni knize nasleduje dalsich 12 kapitol pokryvajicich RAG, AI agenty, MCP servery, fine-tuning a karrierni roadmapu. Kompletni knihu si muzete zakoupit na leanpub.com/AI4BC-DEV.

30denní plán implementace AI do tvé práce

Nejčastější výsledek online kurzu není špatná recenze ani zklamání. Nejčastější výsledek je nic. Člověk kurz dokončí, cítí se skvěle, řekne si „super, teď to začnu používat“ – a za tři týdny je zpátky u starých návyků. AI mu přijde složitá, čas není, projekty jedou, a ten pocit, že „bych měl začít“, se pomalu mění na výčitky svědomí.

Právě proto je tato kapitola jiná – není to teorie, je to akční plán na 30 dní. Konkrétní, realistický a přizpůsobený tomu, kdo jsi.

Proč lidé po kurzu nic nezmění

Tři hlavní vzorce, které vídám u vývojářů a konzultantů opakovaně.

První je paralýza analýzou. Člověk se naučí tolik nástrojů, tolik možností, že neví, kde začít. Copilot nebo Claude? Mám nejdřív projít všechny moduly znovu? Mám se naučit víc o promptování? Výsledek: neudělá nic.

Druhý je „až budu mít čas“. Projekty jedou na plný výkon, zákazník čeká, šéf tlačí. AI je bonus, ne nutnost, takže jde vždy stranou. Jenže „až budu mít čas“ nenastane nikdy. Čas si musíš vzít, nebo si ho AI musí vydělat tím, že ti ho ušetří.

Třetí je perfekcionismus. „Ještě si počkám, až lépe pochopím RAG.“ „Ještě si musím nastudovat MCP servery.“ „Až to budu umět pořádně, pak to začnu používat.“ Jenže AI se naučíš používat jenom tak, že ji začneš používat. Nedá se to nastudovat dopředu.

Řešení na všechny tři? Malé kroky, konzistentně, na reálných projektech. Ne na cvičných příkladech. Ne na demo datech. Na tvých skutečných úkolech, které máš dnes na stole.

Týden 1 – Základy v praxi

Cíl prvního týdne je jednoduchý: zprovoznit nástroje a použít je na jeden skutečný problém.

Konkrétně – nainstaluj Claude Code, pokud jsi to ještě neudělal. Ne stáhnout, ne si přečíst dokumentaci – nainstalovat a otevřít. Pak si vytvoř CLAUDE.md soubor pro svůj aktuální BC projekt. Nemusí být perfektní, stačí 10 řádků: jaký je projekt, jaká je verze BC, jaké jsou hlavní objekty, jak se jmenují tabulky se kterými pracuješ. To je vše.

> **Tip:** Najdi jeden skutečný bug nebo úkol z tvého backlogu a zkus ho vyřešit s AI. Ne cvičení — skutečný ticket. Pokud AI pomůže, skvěle. Pokud ne, zjistíte proč — a to je taky hodnotná informace.

Petr – vzal by si jeden bug z posledního sprintu: chybnou validaci v codeunitě, špatně fungující report, nebo nesprávně počítaný field. Popis problému dá do Claude Code a sleduje, jak AI navrhuje opravu.

Jana – nevyvíjí, ale má před sebou specifikaci nebo testovací scénář. Vezme jeden reálný proces ze zákaznického projektu a zkusí nechat AI napsat draft specifikace. Neumí AL? Nevadí, tohle AL nepotřebuje.

Martin – jako freelancer má pravděpodobně teď rozdělaný projekt a spoustu komunikace. Vezme jeden email zákazníkovi, který odkládá, protože na něj nemá energii – a nechá AI navrhnout strukturu odpovědi.

Týden 2 – Rutina

Cíl druhého týdne není revoluce. Je to rutina.

Každý pracovní den – každý den, ne přes den – použij AI alespoň na jeden pracovní úkol. Jeden email. Jedna specifikace. Jeden kus kódu. Jeden testovací případ. Cokoliv z tvé reálné práce.

A veď si krátký deník. Opravdu krátký – dvě věty. „Dnes jsem použil AI na X. Zabralo to Y minut místo Z minut.“ Nebo: „AI navrhla řešení, které nefungovalo, protože…“ Tohle není přehnané – je to způsob, jak začít vidět vzorce. Co ti AI ušetří, kde naopak zdržuje, na co se hodí a na co ne.

Proč deník? Protože za tři měsíce, až budeš přesvědčovat šéfa nebo kolegu, nebudeš říkat „mám pocit, že mi to pomáhá”. Budeš říkat „tuto specifikaci jsem napsal za 40 minut místo tří hodin“ nebo „tento bug jsem vyřešil za 20 minut, normálně bych hledal hodinu a půl“. To jsou argumenty, které fungují.

Týden 3 – Prohlubování

Teď jsi dva týdny AI používal. Začínáš cítit, co ti bere čas. Třetí týden je o tom vzít jeden konkrétní opakující se úkol – takový, který děláš alespoň jednou za dva týdny a trvá 30 nebo více minut – a vytvořit pro něj prompt šablonu.

Co je prompt šablona? Je to předpřipravený prompt, který obsahuje kontext, instrukce a strukturu výstupu. Uložíš ho do souboru, příště ho otevřeš, doplníš specifika a jedeš. Nemusíš pokaždé vymýšlet, jak AI požádat o pomoc – máš šablonu.

Petr by vytvořil šablonu pro code review AL objektu: jaká jsou pravidla, na co se dívat, jak má vypadat výstup. **Jana** by vytvořila šablonu pro funkční specifikaci procesu v BC: jaká jsou povinná pole, jak má vypadat popis workflow, co musí obsahovat sekce výjimky. **Martin** by vytvořil šablonu pro nabídku zákazníkovi: jak odhadovat časovou náročnost, jak psát executive summary, jak formulovat rozsah prací.

Jedna šablona. Za třetí týden. To je vše.

Týden 4 – Sdílení

Čtvrtý týden je o tom přejít z „já to dělám“ na „my to děláme“. A to ze tří důvodů.

Zaprvé – když něco vysvětlíš kolegovi, teprve tehdy to skutečně pochopíš. Ukaž jednomu kolegovi, jak AI používáš. Nemusí to být přednáška – stačí pět minut: „Hele, podívej se, jak jsem tohle vyřešil.“ To je vše.

Zadruhé – napiš jeden LinkedIn post nebo interní tip pro tým. Nemusí být dlouhý, nemusí být perfektní. Ale tím, že to napíšeš veřejně nebo sdílíš s týmem, vytvoříš závazek. A sám se přinutíš formulovat, co ses naučil.

Zatřetí – navrhni šéfovi nebo vedoucímu jeden konkrétní AI projekt. Ne „měli bychom implementovat AI“. Konkrétně: „Navrhuji, abychom zkusili automatizovat generování testovacích scénářů pro release. Odhaduju, že nám to ušetří X hodin za release.“ Malý, měřitelný, konkrétní. Jak připravit takový návrh, o tom pojednává následující kapitola.

Jak měřit pokrok

Měření pokroku je klíčové. Nestačí po 30 dnech říkat „cítím se produktivněji”. To je k ničemu – pro tebe i pro tvoje argumenty před šéfem.

Místo toho si veď jednoduchou tabulku. Tři sloupce: úkol, čas bez AI, čas s AI. Každý týden přidej tři až pět řádků. Po 30 dnech budeš mít konkrétní data. Budeš vědět, že specifikace ti trvá 40 minut místo tří hodin, že code review trvá 15 minut místo hodiny, že email zákazníkovi napíšeš za 5 minut místo 25.

To jsou čísla, se kterými něco uděláš. To jsou čísla, která přesvědčí šéfa. To jsou čísla, která ti ukáží, kde je největší hodnota.

Shrnutí

30denní plán není složitý. Týden jedna: nainstalovat, nastavit, použít na jeden skutečný problém. Týden dva: každý den jeden úkol s AI, krátký deník. Týden tři: jedna prompt šablona pro opakující se práci. Týden čtyři: ukázat kolegovi, napsat post, navrhnout projekt.

Klíčové pravidlo: reálné projekty, ne cvičení. Malé kroky, konzistentně. Měřit čas, ne pocity.

Největší chyba, kterou můžeš udělat, je počkat, až budeš mít víc času nebo víc znalostí. Čas a znalosti přijdou tím, že začneš. Ne jinak.

Vyzkoušej si

Cvičení 1: Týden 1 — dnes. Neodkládej. Právě teď: otevři AI nástroj a zadej jeden reálný úkol z tvého dnešního to-do listu. Změř čas. Zapiš výsledek.

Cvičení 2: Tabulka sledování. Vytvoř jednoduchou tabulku (Excel nebo papír): Úkol | Čas bez AI | Čas s AI. Tento týden vyplň alespoň 3 řádky.

Cvičení 3: Prompt šablona. Identifikuj jeden opakující se úkol, který děláš alespoň jednou za 2 týdny. Napiš pro něj prompt šablonu a ulož ji.

Detailní checklist pro každý týden

Týden 1: Checklist

- ☐ Claude Code nainstalovaný a fungující
- ☐ CLAUDE.md soubor vytvořený pro aktuální projekt
- ☐ Jeden reálný úkol vyřešený s AI asistencí
- ☐ Poznamenat si: kolik minut to trvalo vs. odhad bez AI
- ☐ GitHub Copilot nainstalovaný ve VS Code (pokud jsi vývojář)

Týden 2: Checklist

- ☐ Pondělí: první úkol s AI
- ☐ Úterý: druhý úkol s AI
- ☐ Středa: třetí úkol s AI
- ☐ Čtvrtek: čtvrtý úkol s AI
- ☐ Pátek: pátý úkol s AI + zápis do deníku
- ☐ Deník: 5 řádků ve formátu „úkol | čas bez AI | čas s AI“

Týden 3: Checklist

- ☐ Identifikovat jeden opakující se úkol (alespoň 1× za 2 týdny)
- ☐ Napsat prompt šablonu pro tento úkol
- ☐ Uložit šablonu do snadno dostupného místa (Notion/VS Code/soubor)
- ☐ Vyzkoušet šablonu na reálném úkolu
- ☐ Upravit šablonu na základě výsledku

Týden 4: Checklist

- ☐ Ukázat jednomu kolegovi jak AI používáš (5 minut)
- ☐ Napsat jeden LinkedIn post nebo interní tip pro tým
- ☐ Navrhnout šéfovi/kolegům jeden konkrétní AI projekt

- ☐ Spočítat měsíční úsporu z dat z deníku (týden 2)
- ☐ Rozhodnout se: pokračuji? Co přidám v měsíci 2?