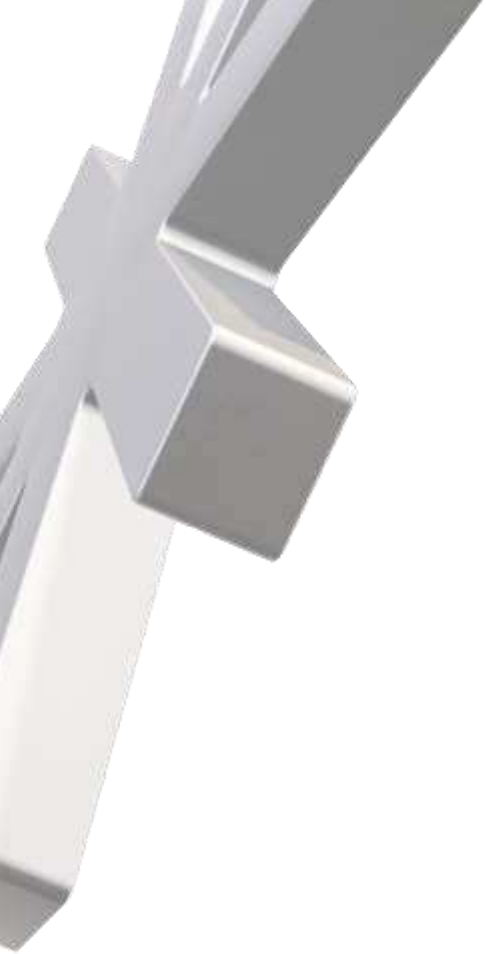
Modernizace legacy core systémů s pomocí AI



180+(40 AI)

Lidí

7

Let na trhu

Digitalizace AI

Strategický partner pro digitalizaci, IT transformaci, implementaci AI

tekies.eu a DNAi dnes

Řešený problém

Historické core systémy >

- IBM i (AS/400) + RPG / CL / DB2, 4GL LANSA nebo Synon/2E
- IBM Z mainframe + COBOL
- PL/I, Assembler (HLASM), Natural/ADABAS (Software AG), IDMS, IMS DB/DC
- CICS / IMS TM
- VB6 / classic ASP / prvotní .NET Framework, Oracle Forms, PowerBuilder, Delphi

Řešený problém

Proč to nejde >

- Lidé, kteří kódu rozumí, odchází do důchodu
- Končí podpora dodavatele
- Mnoho custom vývoje, který posunula původní funkcionalitu jinam
- Desítky let záplat; specifikací je samotný zdrojový kód
- Jakýkoli zásah je riziko — nelze odhadnout rozsah dopadu
- Jakýkoliv vetší rozvoj je drahý a pomalý

Starý software stack je zdaleka nejmenší problém

Přístup

Specification Driven Development >

- Základem je vytvořit 100 procentně kompletní, konzistentní, detailní funkční a technickou specifikaci

Test Driven Development >

- Základem je vytvořit zásadní pokrytí testy a testovacími daty (vstup -> očekávaný výstup) před jakýmkoliv zásahem do kódu

Nutnost dotrénování AI modelu>

- Obecná LLM nejsou dostatečně natrénovaná pro některé jazyky (nedostatek public dat např. pro LANSA)
- Obecně trénovaná pravidla nejsou dostatečná pro hlubokou doménovou znalost a způsob užití. Viz např. gap mezi definicí zákona a jeho výkladem

Přístup k trénování AI

Actor-Critic metoda převzatá z **GAN** (Generative Adversarial Networks). Jeden agent tvoří a druhý agent hodnotí kvalitu a poskytuje slovní hodnocení (**Reflexe**) — z toho hodnocení se první učí, kterým směrem se zlepšit (**TextGrad**).

- **Aktor** (jednající) = ten, kdo vyrobí výstup. U nás: AI, která z legacy kódu navrhne např. business pravidlo
- **Kritik** (hodnotící) = nepřepisuje práci aktora, jen jí dá známku a směr — „tohle je z 70 % dobře, slabina je v té citaci, posuň se sem". Aktor pak při dalším pokus získá vyšší hodnocení

Reflexe = Model po neúspěchu slovně popíše, proč selhal, a uloží si to jako poznámku do paměti. Kritik zamítne pravidlo → systém napíše lekci: „minule jsem vymyslel daňový člen — vždy ověř číselné literály proti zdroji."
TextGrad = Ten slovní zpětnou vazbu použije jako „gradient": konkrétní návod, co v zadání změnit. Z lekce udělá konkrétní úpravu extrakčního promptu: přidá pravidlo - jako učení z chyb v neuronové síti, ale „derivace" je text, ne čísla.

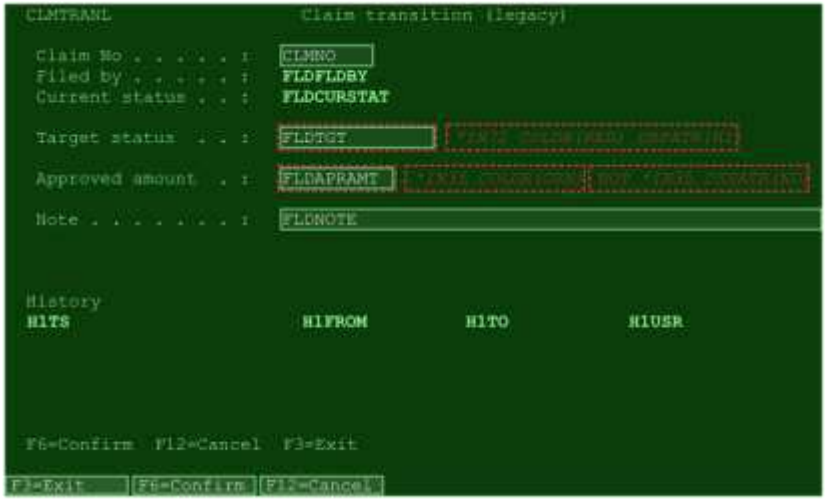
Reverse engineering AS IS Specifikace

DS display files .DSPF
LANSA RDMLX
RPGLE a CL driver

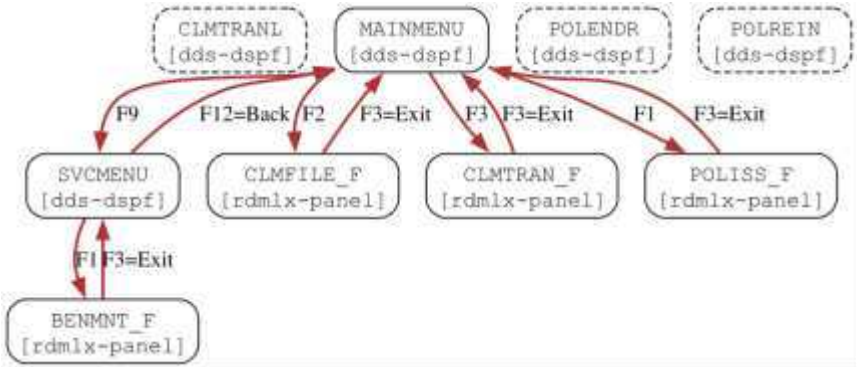
GitHub Copilot
GPT-5.3-Codex

screen mockups
screen flows
GUI logika

Př: Zpětně generované obrazovky ve Figma



Př: Zpětně generované screen flow



Reverse engineering AS IS Specifikace

RPG IV — RPGLE
CL — CLLE
COBOL, Copybooky
DDS — PF / LF / DSPF / PRTF

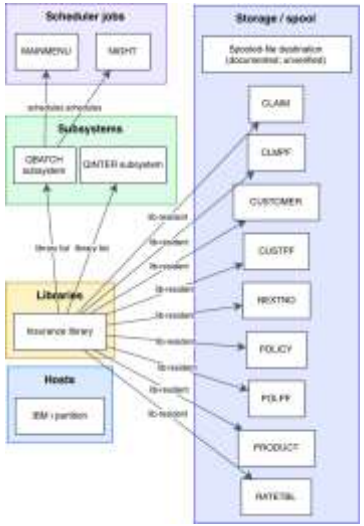
- Funkční požadavky (atomické „systém MUSÍ...")
- Akceptační kritéria / behaviorální spec (Gherkin scénář na pravidlo s konkrétními daty = BDD požadavky)
- Use-case / uživatelské toky (activity diagramy top-level toků + screen-flow)
- Ruly, validace, invarianty, side efekty, edge casy
- Lifecycle požadavky (state machines pro entity)
- Nefunkční požadavky
- ER modely

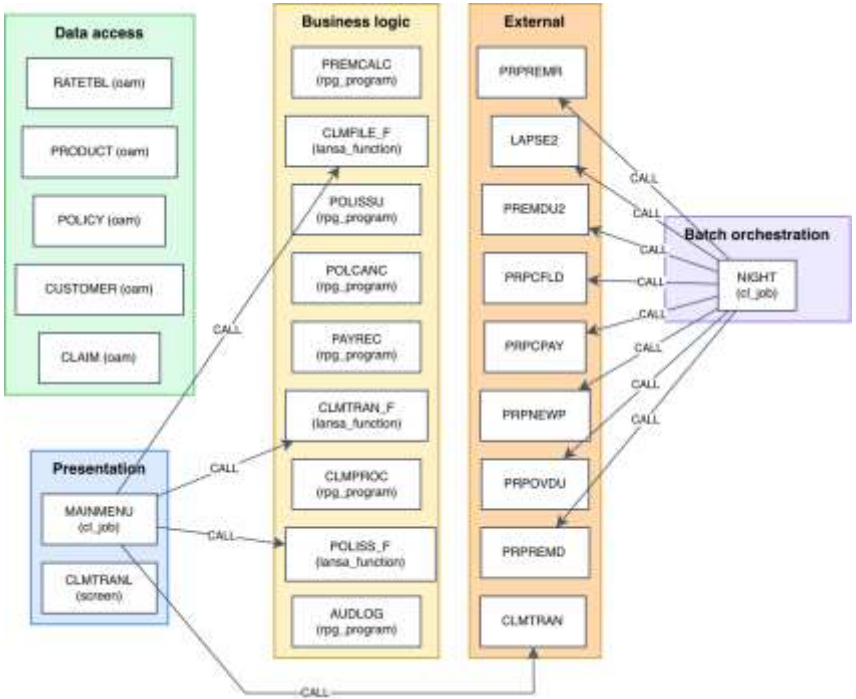
Reverse engineering AS IS Architektury

Řezy AS IS architektury >

GitHub Copilot
GPT-5.3-Codex

- Aplikační/komponentová
- Integrační
- Infrastrukturní
- Informační
- Technologický stack

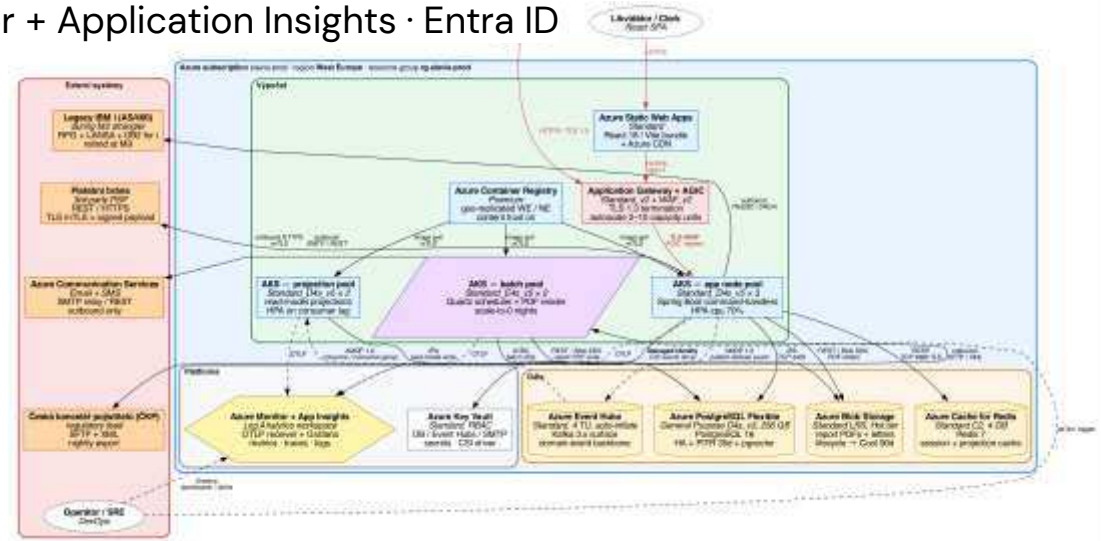




Návrh TO BE Architektury

Příklad migrace>

- Běh: Azure Kubernetes Service 1.30 (app/batch/projection pooly) za
- Data: Azure Database for PostgreSQL Flexible Server (OLTP) ·
- Integrace: Azure Event Hubs (Kafka-compatible) — CQRS event backbone
- Aplikace: Kotlin 2 / Spring Boot 3 služby + Quartz batch; React 18 SPA
- Provoz: Azure Key Vault · Azure Monitor + Application Insights · Entra ID



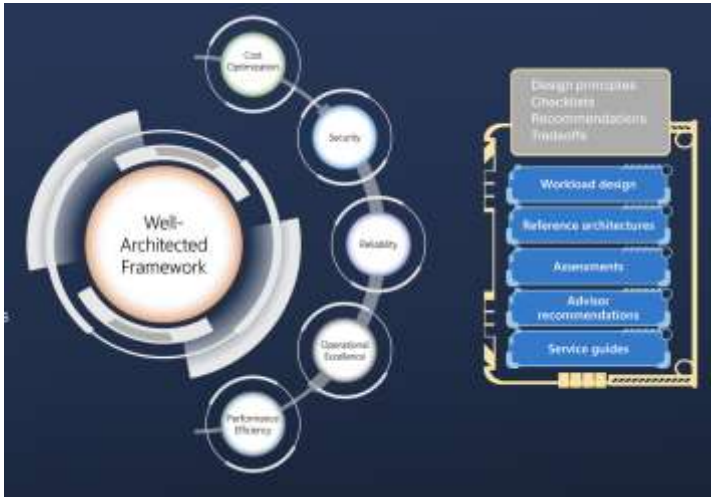
Optimalizace a reengineering

Migrujeme 1:1 nebo upravujeme?

- Nalezení duplicit ve funkcionalitách
- Nalezení chyb ve funkcionalitách, nekonzistencí
- Konsolidace procesů a postupů - různé způsoby dělání stejných věcí (např. v GUI a procesech), komplikované
- Duplicity v datech a službách
- atd ...

Plán Migrace

- Nacenění / TCO: Azure Migrate – Business Case
- Discovery & sizing: Azure Migrate – Discovery + Dependency analysis
- Design gate: Azure Well-Architected Framework
- Plán & kritická cesta: Microsoft Planner

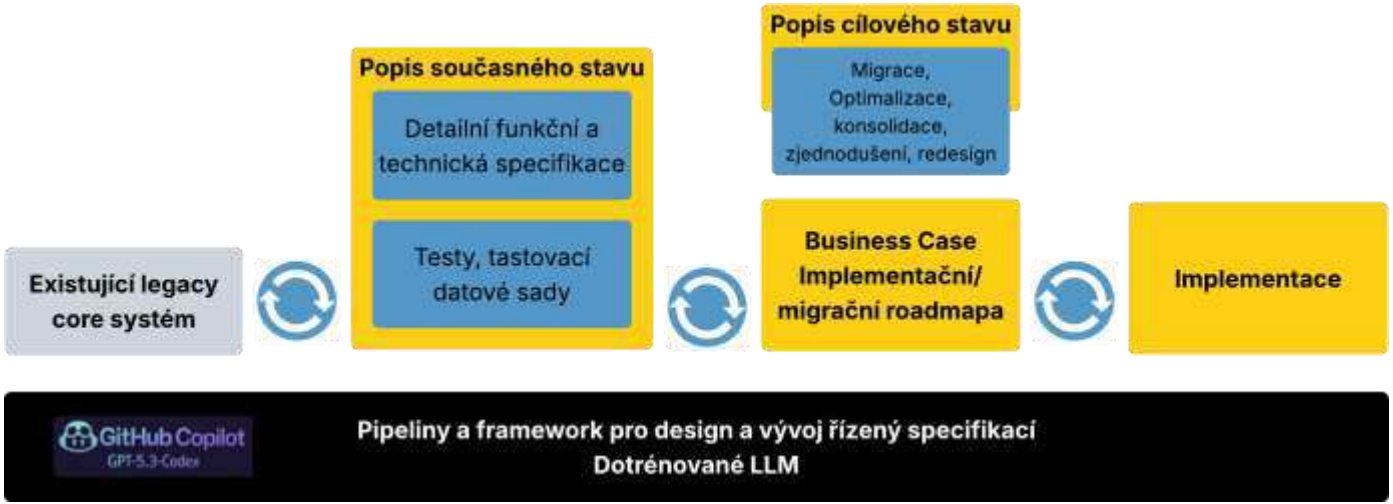


Azure Pricing Calculator → Azure Retail Prices API

Microsoft Azure Migrate

Výsledky

=> První výsledky indikují zásadní úspory v nákladech na migraci (okolo 50-90 procent), výrazné zkrácení času a systém umožňující úsporný a rychlý další provoz



Další kroky: jednodenní workshop, kde se provede první estimace, odhad potenciálních výsledků a prerekvizity pro aplikaci uvedeného přístupu

David Kadlecek, PhD.
CTO & Co-founder

david.kadlecek@dnai.ai
+420 737 264 127