

Vysoká škola ekonomická v Praze
Fakulta informatiky a statistiky



**Návrh a testování prostředí pro AI
agenty v softwarovém vývoji**

BAKALÁŘSKÁ PRÁCE

Studijní program: Aplikovaná informatika

Autor: Thanh An Nguyen

Vedoucí práce: Ing. Richard Antonín Novák, Ph.D.

Praha, květen 2026

Prohlášení

Prohlašuji, že jsem bakalářskou práci zpracoval samostatně a že jsem uvedl všechny použité prameny a literaturu, ze kterých jsem čerpal.

V Praze dne 28. ledna 2026

Thanh An Nguyen

Poděkování

Poděkování.

Abstrakt

TODO: Napsat až budou hotové výsledky.

Klíčová slova

AI agenti, softwarový vývoj, prostředí, scaffolding, kvalita kódu

Abstract

TODO: Write after results are complete.

Keywords

AI agents, software development, environment, scaffolding, code quality

Obsah

| | |
|---|-----------|
| Úvod | 9 |
| 1 Vymezení problému a cílů práce | 10 |
| 1.1 Motivace | 10 |
| 1.2 Cíle práce | 10 |
| 1.3 Rozsah práce | 10 |
| 2 Teoretická východiska | 12 |
| 2.1 Softwarové inženýrství | 12 |
| 2.1.1 Definice a vymezení oboru | 12 |
| 2.1.2 Komplexita software | 12 |
| 2.1.3 Historický kontext | 13 |
| 2.2 Životní cyklus a metodiky | 13 |
| 2.2.1 Fáze životního cyklu | 13 |
| 2.2.2 Modely a metodiky | 14 |
| 2.2.3 Nástroje | 14 |
| 2.2.4 Artefakty | 15 |
| 2.2.5 Role a komunikace | 15 |
| 2.3 Agentic coding | 16 |
| 2.4 Scaffolding pro agenty | 17 |
| 3 Metodika | 18 |
| 3.1 Výběr projektu pro case study | 18 |
| 3.2 Referenční implementace | 18 |
| 3.3 Experimenty | 19 |
| 3.4 Analýza | 19 |
| 4 Praktická část | 20 |
| 5 Vyhodnocení a diskuse | 21 |
| Závěr | 22 |
| A Formulář v plném znění | 28 |
| B Zdrojové kódy výpočetních procedur | 29 |

Seznam obrázků

Seznam tabulek

Seznam použitých zkratek

TODO Doplňte zkratky a tento řádek smažte.

Úvod

1. Vymezení problému a cílů práce

1.1 Motivace

[DRAFT]

Studie společnosti METR.ORG ukazuje, že LLM zkušené vývojáře spíše zpomaluje. S rychlým vývojem schopností modelů se situace pravděpodobně mění. Ale to neznamená, že je LLM samo o sobě dostatečné k vypracování dlouhotrvajících úkolů. Není to problém pouze LLM, když projekt roste, bývá těžší jej rozšiřovat jak pro člověka, tak i pro LLM. AI programování tenhle problém ještě více prohlubuje. Vývojáři přicházejí o kontext a hlubokou znalost kódové základny (codebase), zatímco velké jazykové modely (LLM) jsou limitovány omezenou pamětí (context window). Jak nastavit harness/scaffolding tak, aby v tom mohli fungovat agenti a lidé to stále měli pod kontrolou?

1.2 Cíle práce

[DRAFT]

1. Popsat jak se řízení SWE projektů mění v kontextu agentních systémů (teoretický rámec)
2. Navrhnout a implementovat experimentální prostředí (case study: systém upomínek faktur)
3. Prozkoumat vliv různých nastavení scaffoldingu na schopnost agenta provést kvalitní práci
4. Identifikovat jaký kontext je pro agenty klíčový a jak instrukce/procesy ovlivňují schopnost agenta tento kontext vytvářet a využívat

1.3 Rozsah práce

[RAW]

Poznámky k propojení:

- Řízení vyžaduje holistický pohled (vidět celek, ne jen část) - proto celý SDLC, ne jedna fáze
- Billing Reminder Engine jako case study: malý projekt, ale reálné nuance (state machine, business days, edge cases)
- Deterministická logika (stejný vstup = stejný výstup) + jasná pravidla správnosti = objektivně testovatelné

- Hraniční případy (víkendy, svátky, grace periods) = místa kde lze testovat kvalitu agentní práce

TODO: Vysvětlit termíny:

- state machine - ?
- edge cases / hraniční případy - ?
- SDLC - ?

Scope SDLC (k diskuzi):

- Primární plán: celý SDLC včetně deployment/maintenance
- Fallback: zúžit na implementation + testing (hlavní doména coding agents)
- Poznámka: i impl + testing má feedback loop (implementace → testy → chyba → úprava) = stále systémový pohled

[DRAFT]

Práce pokrývá celý SDLC na jednoduchém projektu. Součástí je vlastní implementace, která poskytuje hloubku porozumění potřebnou pro návrh experimentů.

Práce se zaměřuje na:

- Nastavení a použití existujících nástrojů (GitHub, CLI agents)
- Exploratory case study - hledání vzorů a doporučení

Práce se nezaměřuje na:

- Programování nových nástrojů od nuly
- Porovnávání různých LLM modelů
- Porovnávání různých programovacích jazyků
- Vývoj produkčního nástroje/produkту

2. Teoretická východiska

[DRAFT]

Tato kapitola vysvětuje teoretická východiska práce. Nejprve je popsáno softwarové inženýrství jako disciplína, následně životní cyklus a metodiky vývoje software. Poté je zkoumáno, jak se oblast mění díky AI coding agentům, a nakonec jsou představeny prvky scaffoldingu (podpůrných struktur), které agenti využívají.

2.1 Softwarové inženýrství

[RAW]

Úvod sekce 2.1 (1-2 věty): Tato sekce definuje softwarové inženýrství, vysvětuje proč je software inherentně složitý a jak tato složitost vedla ke vzniku oboru.

2.1.1 Definice a vymezení oboru

[RAW]

Co sem patří:

- IEEE/SWEBOK definice softwarového inženýrství
- Vymezení vůči příbuzným oborům (computer science, programování)
- Proč "inženýrství"- systematický, disciplinovaný přístup
- SWE je víc než psaní kódu - zahrnuje procesy, kvalitu, údržbu

Zdroje:

- SWEBOK v4 - definice, knowledge areas
- IEEE Standard Glossary of Software Engineering Terminology

2.1.2 Komplexita software

[RAW]

Co sem patří:

- **Proč je software složitý** (Brooks - No Silver Bullet):
 - Essential complexity - inherentní složitost problému (business logika, requirements)
 - Accidental complexity - složitost kterou si přidáváme (nástroje, technologie)

- Software je neviditelný, snadno měnitelný, nemá fyzická omezení

Zdroje:

- Brooks - No Silver Bullet (1986) - essential vs accidental complexity
- McConnell - Code Complete (cituje Brookse)

2.1.3 Historický kontext

[RAW]

Co sem patří:

- NATO 1968 konference - "software crisis"
- Projekty selhávaly, překračovaly rozpočty, nedodávaly včas
- Potřeba inženýrské disciplíny místo ad-hoc programování
- Vznik SWE jako reakce na komplexitu a krizi

Zdroje:

- NATO 1968 - historický kontext vzniku oboru
- Sommerville - Software Engineering (úvod, historie)

2.2 Životní cyklus a metodiky

[RAW]

Úvod sekce 2.2 (1-2 věty): Tato sekce popisuje jak se software vyvíjí v praxi: co se dělá (fáze), jak se to organizuje (metodiky), čím se to dělá (nástroje), co vzniká (artefakty), a kdo to dělá (role a komunikace).

2.2.1 Fáze životního cyklu

[RAW]

Co sem patří:

- Sommerville - 4 základní aktivity:
 1. Software specification
 2. Software development
 3. Software validation
 4. Software evolution
- Každá fáze produkuje artefakty

- Cyklický charakter - software se neustále vyvíjí

Zdroje:

- Sommerville - Software Engineering (SDLC)
- SWEBOK v4 - Software Engineering Process KA

2.2.2 Modely a metodiky

[RAW]

Co sem patří:

- Waterfall - sekvenční, dokumentace dopředu
- Iterativní/inkrementální - opakované cykly
- Agile - flexibilita, rychlý feedback, spolupráce
- Různé modely = různé způsoby organizace stejných fází
- Trade-offs: prediktabilita vs flexibilita

Zdroje:

- Sommerville - process models
- Boehm - Spiral Model (1988)
- Beck - Extreme Programming Explained
- Agile Manifesto

2.2.3 Nástroje

[RAW]

Co sem patří:

- IDE - integrovaná vývojová prostředí
- Version control (Git) - správa verzí, spolupráce
- CI/CD - automatizace buildů, testů, deploymentu
- Issue tracking, dokumentace
- Nástroje jako součást procesů - umožňují škálování

Proč je to důležité pro BP: Agenti využívají tyto nástroje - musí s nimi umět pracovat. Scaffolding staví na existujících nástrojích.

Zdroje:

- SWEBOK v4 - Software Engineering Tools KA
- Praktické znalosti (Git, GitHub Actions, etc.)

2.2.4 Artefakty

[RAW]

Co sem patří:

- Specifikace a požadavky (requirements)
- Návrhy a architektura (design docs)
- Zdrojový kód
- Testy (unit, integration, e2e)
- Dokumentace (technická, uživatelská)
- Artefakty jako vstup/výstup fází životního cyklu

Proč je to důležité pro BP: Agenti produkují a konzumují tyto artefakty. Kvalita artefaktů ovlivňuje kvalitu výstupu agenta.

Zdroje:

- Sommerville - Software Engineering
- SWEBOK v4

2.2.5 Role a komunikace

[RAW]

Co sem patří:

- SWE je fundamentálně týmová disciplína
- Hodně znalostí je implicitních (v hlavách lidí, ne v dokumentaci)
- Brooks's Law - komunikační režie roste exponenciálně s velikostí týmu
- Procesy slouží ke koordinaci, konzistenci, předávání znalostí
- Nástroje a artefakty jako prostředky komunikace (CSCW perspektiva)

Proč je to důležité pro BP: Když pak v 2.3 řekneme "agenti nepotřebují meetingy ale vše musí být explicitní"- čtenář pochopí co se mění. Implicitní znalosti v týmu → explicitní instrukce pro agenta.

Zdroje:

- Brooks - Mythical Man-Month (Brooks's Law)
- Beck - Extreme Programming Explained (týmová spolupráce)
- CSCW literatura (viz issue #9)

2.3 Agentic coding

[RAW]

Co může sekce obsahovat:

- Co jsou coding agents (definice, typy)
 - Jak agenti mění vývoj (co zůstává, co se mění)
 - Konvergence SDLC - micro-waterfall hypotéza
 - Trade-off agenti vs týmy
-

[K ROZPRACOVÁNÍ] Konvergence SDLC modelů - micro-waterfall hypotéza:

Zajímavý postřeh: nevracíme se s AI agenty zpět k waterfall, jen v menší časové škále?

- **Waterfall (klasický):** Requirements → Design → Implementation → Testing → Deployment [měsíce per fáze]
- **Agile (sprint):** Planning → Dev → Testing → Review [2-4 týdny per cyklus]
- **AI agenti:** Prompt → Generate → Review → Fix [minuty per cyklus]

Ve všech případech máš sekvenční kroky (specifikace → implementace → validace). Agile je nezrušil - jen zmenšil a zrychlil. AI agenti je zmenšují ještě více.

Klíčové rozdíly:

- Batch size: celý projekt → feature → jeden prompt
 - Feedback loop: měsíce → týdny → minuty
 - Kdo specifikuje: analytik (dokument) → PO + tým (stories) → developer (prompt)
 - Kdo validuje: QA na konci → tým průběžně → developer okamžitě
-

[K ROZPRACOVÁNÍ] Trade-off agenti vs týmy:

- Tradiční SWE = hodně o organizaci a lidech (komunikace, koordinace, předávání znalostí)
- Velké týmy = komunikační režie (Brooks's Law)
- Agenti nepotřebují meetingy, ale nerozumí nuancím - vše musí být explicitní
- Žádné implicitní porozumění z konverzace ("řekli jsme si na meetingu")
- Hypotéza: malé týmy s agenty mohou růst rychleji než velké týmy bez nich

Zdroje:

- Surveys o LLM agentech v SE (máme v literatuře)
 - Jin et al. - LLM Agents for SWE Survey
-

- Foundational Pillars of Agentic SE

2.4 Scaffolding pro agenty

[RAW]

Co může sekce obsahovat:

- Co je scaffolding (definice, proč je potřeba)
 - Spec-driven development (BMAD, SpecKit...)
 - Nástroje pro řízení agentů (CLAUDE.md, hooks, agents.md)
 - Metriky a hodnocení (jak měřit úspěšnost agentů)
-

Poznámky k zpracování:

- ITIL/CMMI relevance - ověřit jestli vůbec patří do BP
- Viz handoffs/03-agent-framework-brainstorm.md a 06-agent-framework-consolidated.md

3. Metodika

3.1 Výběr projektu pro case study

[RAW]

Práce se zaměřuje na řízení a scaffolding - jde více do šířky než do hloubky. Proto potřebujeme menší projekt, na kterém můžeme spustit více běhů s různými nastaveními scaffoldingu a měřit výsledky.

Pro experiment potřebujeme projekt který:

- **Hard logic** - jasná business pravidla, ne subjektivní výstupy (např. generování textu)
- **Jasné invarianty** - deterministické chování, matematicky ověřitelné správnost
- **Testovatelné** - lze objektivně měřit kvalitu výstupu
- **Přiměřená velikost** - menší projekt umožňuje více experimentálních běhů
- **Reálný use case** - prakticky využitelné, ne umělý příklad

Systém upomínek faktur

[RAW]

Systém pro automatické odesílání připomínek k nezaplaceným fakturám. Obsahuje:

- Stavový automat pro sledování stavu faktury (nová, po splatnosti, upomínaná, eskalovaná)
- Časové výpočty (pracovní dny, ochranné lhůty)
- Pravidla pro escalaci (kdy poslat další upomínku, kdy předat k vymáhání)
- Plánování odesílání upomínek

3.2 Referenční implementace

[RAW]

Vlastní vývoj systému upomínek faktur se všemi náležitostmi softwarového inženýrství:

- Specifikace a dokumentace
- Implementace (TypeScript)
- Testy (unit, integration)
- Git workflow (issues, commits, PR conventions)
- Quality gates (linting, type checking)

Tato implementace slouží jako "ground truth" pro porovnání.

3.3 Experimenty

[RAW]

Příprava různých nastavení scaffoldingu pro agenty:

- Instrukce a procesy (jak má agent postupovat)
- Git automatizace a skripty
- Kontext který má agent k dispozici a který si vytváří
- Nastavení odvozená z literatury a spec-driven development přístupů

Spuštění agentů s různými nastaveními scaffoldingu na stejném zadání. Zaznamenání průběhu a výstupů.

3.4 Analýza

[RAW]

Měření výstupů proti referenční implementaci:

- **Functional Quality** (dle ISO 25010):
 - Completeness - míra pokrytí požadované funkcionality
 - Correctness - správnost výsledků
- **Compliance** (dodržování procesů):
 - Workflow (issues → branch → PR)
 - Konvence (commit messages, dokumentace)
 - Transparentnost (vysvětluje co dělá a proč)

Identifikace vzorů - která nastavení scaffoldingu vedla k lepším výsledkům.

[RAW]

Poznámka: Konkrétní metriky pro hodnocení výstupu agentů jsou aktivní oblast výzkumu. V rámci práce bude provedena dodatečná rešerše dostupných akademických přístupů.

4. Praktická část

5. Vyhodnocení a diskuse

Závěr

[RAW]

Zdroje k zpracování do BibTeX (původně z notes/sources.md)

1. PRIMÁRNÍ ZDROJE (Frameworky a Metodiky)

GITHUB NEXT. Spec Kit: Spec-Driven Development for AI Agents [online]. 2024. <https://github.com/github/spec-kit> – Klíčový zdroj pro koncept "Spec-Driven Development".

BMAD-CODE-ORG. BMAD METHOD: Breakthrough Method for Agile AI-Driven Development [online]. GitHub, 2025. <https://github.com/bmad-code-org/BMAD-METHOD> – Metodika pro řízení AI agentů v agilním vývoji.

ANTHROPIC. Effective Harnesses for Long-Running Agents [online]. Anthropic Engineering Blog, 2024. <https://www.anthropic.com/engineering/effective-harnesses-for-long-running-agents> – Technický popis problémů s dlouhodobou pamětí agentů.

METR. Model Evaluation and Threat Research [online]. 2024. <https://metr.org/> – Standardy pro hodnocení bezpečnosti a schopnosti modelů.

THEDOTMACK. claude-mem: Persistent Memory System for Claude Code [online]. GitHub, 2025. <https://github.com/thedotmack/claude-mem> – Příklad implementace hooks v CLI agentech - persistentní paměť přes session.

1.2 SYSTÉMOVÉ MYŠLENÍ V SWE

PETKOV, Doncho et al. Information Systems, Software Engineering, and Systems Thinking: Challenges and Opportunities. International Journal of Information Technologies and Systems Approach. <https://www.igi-global.com/gateway/article/2534> – Mapuje historii systémového přístupu v IS a SWE. Propojení systémového myšlení s praxí SWE je stále nedotažené.

MONAT, Jamie a GANNON, Thomas. Systems Thinking: A Review and Bibliometric Analysis. MDPI Systems, 2020. <https://www.mdpi.com/2079-8954/8/3/23> – Přehled co systémové myšlení je a kde se používá. Interdisciplinární - SWE, management, vzdělávání.

ALHARTHI, Sultan et al. A Systems Thinking Approach to Improve Sustainability in Software Engineering. MDPI Sustainability, 2023. <https://www.mdpi.com/2071-1050/15/11/876> – Praktická aplikace - dívají se na vývoj jako systém (developeri, zákazníci, stakeholders).

2. ODBORNÉ STUDIE

2.1 Přehledové studie (Surveys) - LLM agenti v SE

LIU, Junwei et al. Large Language Model-Based Agents for Software Engineering: A Survey. arXiv preprint arXiv:2409.02977. 2024. <https://arxiv.org/abs/2409.02977> – Komplexní přehled 106 prací o LLM agentech v SE, kategorizace z pohledu SE i agentů.

JIN, Haolin et al. From LLMs to LLM-based Agents for Software Engineering: A Survey of Current, Challenges and Future. arXiv preprint arXiv:2408.02479. 2024. <https://arxiv.org/abs/2408.02479> – Pokrývá requirements, code generation, testing, maintenance - celý SDLC.

Comprehensive Survey on Benchmarks and Solutions in Software Engineering of LLM-Empowered Agentic System. arXiv preprint arXiv:2510.09721. 2025. <https://arxiv.org/html/2510.09721> – Přes 150 paperů, taxonomie řešení a benchmarků.

A Survey on Code Generation with LLM-based Agents. arXiv preprint arXiv:2508.00083. 2025. <https://arxiv.org/abs/2508.00083> – Single-agent a multi-agent architektury, aplikace napříč SDLC.

2.2 Agentic Software Engineering - Klíčové práce

Agentic Software Engineering: Foundational Pillars and a Research Roadmap. arXiv preprint arXiv:2509.06216. 2025. <https://arxiv.org/html/2509.06216v2> – Přehodnocení SE pro spolupráci člověk-agent. Framework podobný SAE úrovní autonomie. Rozlišuje SE 2.0 (AI-augmented) vs SE 3.0 (Agentic SE).

AKBAR, Muhammad Azeem et al. Agentic AI in Software Engineering: Practitioner Perspectives Across the Software Development Life Cycle. SSRN. 2025. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=5520159 – Rozhovory s 21 experty, pokrývá celý SDLC. Zjištění: agenti redefinují hranice mezi fázemi SDLC.

Autonomous Agents in Software Development: A Vision Paper. Springer, 2024. https://link.springer.com/chapter/10.1007/978-3-031-72781-8_2 – 12 LLM agentů spolupracujících na celém SDLC.

2.3 Evaluace a produktivita

METR. Measuring the Impact of Early-2025 AI on Experienced Open-Source Developer Productivity. 2025. <https://metr.org/blog/2025-07-10-early-2025-ai-experienced-os-dev-study/> – RCT studie: zkušení vývojáři s AI jsou o 19% pomalejší - překvapivé zjištění.

2.4 Metriky kvality software a AI agentů

ISO/IEC. ISO/IEC 25010:2023 - Systems and software Quality Requirements and Evaluation (SQuaRE) — Product quality model. 2023. <https://www.iso.org/standard/78176.html> – Industry standard pro kvalitu software. 8 charakteristik, Functional Suitability obsahuje Completeness, Correctness, Appropriateness.

ISO 25000. ISO 25010 Software Quality Model. 2023. <https://iso25000.com/index.php/en/iso-25000-standards/iso-25010> – Přehledný popis ISO 25010 modelu kvality.

LXT. AI Agent Evaluation: Comprehensive Framework for Measuring Agent Performance. 2024. <https://www.lxt.ai/blog/ai-agent-evaluation/> – Moderní framework pro evaluaci AI agentů: Task completion, Accuracy, Safety/trust (policy compliance, transparency), Tool

usage.

Weights & Biases. AI Agent Evaluation: Metrics, Strategies, and Best Practices. 2024. <https://wandb.ai/onlineinference/genai-research/reports/AI-agent-evaluation-Metrics-strategies-and-best-practices--VmlldzoxMjM0NjQzMQ> – Praktický průvodce metrikami pro AI agenty.

2.5 Základní LLM studie

JIMENEZ, Carlos E. et al. SWE-bench: Can Language Models Resolve Real-world Github Issues? In: The Twelfth International Conference on Learning Representations (ICLR). 2024. <https://arxiv.org/abs/2310.06770> – Hlavní benchmark pro hodnocení schopností programovacích agentů.

LIU, Nelson F. et al. Lost in the Middle: How Language Models Use Long Contexts. arXiv preprint arXiv:2307.03172. 2023. <https://arxiv.org/abs/2307.03172> – Klíčová studie vysvětující, proč pouhé zvětšení kontextového okna nestačí.

VASWANI, Ashish et al. Attention Is All You Need. Advances in Neural Information Processing Systems, 2017. <https://arxiv.org/abs/1706.03762> – Základní paper definující Transformer architekturu a mechanismus pozornosti (self-attention).

WEI, Jason et al. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. Advances in Neural Information Processing Systems, 2022. <https://arxiv.org/abs/2201.11903> – Základ pro techniky prompt engineeringu používané v práci.

3. TEORIE SOFTWAROVÉHO INŽENÝRSTVÍ A ARCHITEKTURY

RICHARDS, Mark a FORD, Neal. Fundamentals of Software Architecture: An Engineering Approach. O'Reilly Media, 2020. ISBN 978-1492043454. – Moderní přehled architektonických stylů a charakteristik ("ilities").

FOWLER, Martin. Patterns of Enterprise Application Architecture. Addison-Wesley Professional, 2002. ISBN 978-0321127426. – Katalog základních návrhových vzorů pro podnikové aplikace.

KHONONOV, Vlad. Learning Domain-Driven Design: Aligning Software Architecture and Business Strategy. 1. vyd. O'Reilly Media, 2021. ISBN 978-1098100131. – Definuje pojmy jako "Bounded Context" a "Ubiquitous Language", které jsou analogií pro kontext LLM.

ISO/IEC. ISO/IEC/IEEE 42010:2011 Systems and software engineering — Architecture description. 2011. – Mezinárodní standard definující základní pojmy popisu architektury.

IEEE COMPUTER SOCIETY. SWEBOK: Guide to the Software Engineering Body of Knowledge. Version 3.0. IEEE, 2014. <https://www.swebok.org/> – Standardní taxonomie softwarového inženýrství.

BROWN, Simon. The C4 model for visualising software architecture. 2024. <https://c4model.com/> – Metodika pro hierarchický popis architektury, vhodná pro strojové zpracování.

ARC42. arc42 Template for Software Architecture Documentation. 2024. <https://arc42.org/> – Pragmatická šablona pro strukturování architektonické dokumentace.

4. DALŠÍ ZDROJE (Historický kontext a Procesy)

NATO SCIENCE COMMITTEE. Software Engineering: Report on a conference sponsored by the NATO Science Committee. Garmisch, Germany, 1968. – Historický kontext vzniku disciplíny.

KAUR, Rupinder a SENGUPTA, Jyotsna. Software Process Models and Analysis on Failure of Software Development Projects. In: arXiv preprint arXiv:1306.1068. 2013.

Přílohy

A. Formulář v plném znění

B. Zdrojové kódy výpočetních procedur