Architektonická analýza a implementační plán pro multi-repozitářový ekosystém MCP Prompts

Shrnutí

Tento dokument předkládá komplexní architektonickou analýzu a detailní implementační plán pro strategický přechod projektu MCP Prompts z monolitického repozitáře na modulární, multi-repozitářový ekosystém. Cílem této transformace je zvýšit modularitu, zavést nezávislé release cykly pro jednotlivé komponenty, zlepšit škálovatelnost a zjednodušit proces onboardingu pro nové jazykové implementace.
Analýza potvrzuje, že navržená topologie, sestávající z centrálního meta-repozitáře pro orchestraci, samostatného repozitáře pro sdílenou kolekci promptů a nezávislých repozitářů pro jednotlivé technologické implementace, je v souladu s osvědčenými postupy v oboru a představuje robustní základ pro budoucí růst.
Klíčová doporučení zahrnují:

1. **Fázovou, inkrementální migraci** namísto jednorázového "big bang" přístupu pro minimalizaci rizik a zajištění kontinuity.
2. Využití nástroje **git-filter-repo** pro rozdělení repozitáře při zachování kompletní historie revizí, což je klíčové pro udržení kontextu vývoje.
3. Implementaci **hybridního CI/CD orchestračního modelu**, který kombinuje znovupoužitelné workflow (workflow_call) pro hlavní orchestrační logiku s kompozitními akcemi pro sdílené pomocné skripty. Tento přístup poskytuje optimální rovnováhu mezi autonomií jednotlivých repozitářů a centralizovanou kontrolou a viditelností celého release procesu.

Realizací tohoto plánu se MCP Prompts transformuje na elegantní, škálovatelný a snadno rozšiřitelný ekosystém, připravený na budoucí výzvy a přispění komunity.

I. Analýza navržené multi-repozitářové topologie

Navržená architektura zavádí logickou dekompozici stávajícího monolitického systému na specializované, spolupracující komponenty. Tento přístup řeší rostoucí komplexitu projektu a připravuje půdu pro jeho další rozvoj.

1.1. Vzor meta-orchestrátoru (mcp-prompts)

Navrhovaný repozitář mcp-prompts představuje centrální uzel ekosystému. Neobsahuje žádný produkční kód, ale slouží jako "maják" pro orchestraci, dokumentaci a správu releasů. Tato strategie centralizuje koordinaci a zmírňuje riziko "koordinační režie", která je častým problémem v multi-repozitářových prostředích.
Srdcem tohoto repozitáře je soubor .github/workflows/dispatch.yml, který naslouchá na události

repository_dispatch a spouští klíčové procesy :

- Sestavení "suite" Docker image, která agreguje nejnovější verze jednotlivých implementací.
- Generování tabulky kompatibility, jež mapuje verze implementací na verze sdílené kolekce promptů.
- Publikaci souhrnných release notes.

Druhý definovaný workflow, docs.yml, automatizuje generování a publikaci dokumentace, čímž posiluje roli meta-repozitáře jako centrálního zdroje informací. Stávající monorepo již obsahuje rozsáhlou dokumentaci a CI/CD workflow , což poskytuje solidní základ pro obsah tohoto nového repozitáře.

1.2. Jediný zdroj pravdy (mcp-prompts-collection)

Plán správně identifikuje adresář prompts/ jako klíčové, sdílené aktivum. Návrh na jeho extrakci do samostatného, nezávisle verzovaného repozitáře mcp-prompts-collection a jeho následnou distribuci jako NPM balíčku (@sparesparrow/mcp-prompts-collection) a Cargo crate (mcp-prompts-collection) je zásadním krokem.

Tato dekompozice odděluje data od implementací, což je fundamentální princip pro dosažení skutečné modularity. Zajišťuje, že všechny implementace (TypeScript, Rust, Python atd.) konzumují identickou, verziovanou a validovanou sadu promptů. Stávající struktura monorepa již naznačuje tento směr existencí adresáře packages/mcp-prompts-catalog , který obsahuje vlastní podadresář prompts. Navrhovaná změna tento přístup formalizuje a povyšuje na úroveň celého ekosystému. Bohatý obsah adresáře prompts/ v kořeni projektu představuje klíčové duševní vlastnictví, což plně ospravedlňuje jeho vyčlenění do samostatné, pečlivě spravované komponenty.

1.3. Nezávislé implementace (mcp-prompts-*)

Migrace jednotlivých jazykově specifických implementací do vlastních repozitářů (mcp-prompts-ts, mcp-prompts-rs, mcp-prompts-pg, mcp-prompts-aidl) poskytne každému týmu nebo komponentě autonomii. Každý repozitář bude mít vlastní CI, verzování a release proces, přičemž bude konzumovat mcp-prompts-collection jako verziovanou závislost. To vede k čistší historii Gitu, menším a rychlejším klonům repozitářů a cílenějším a rychlejším CI pipeline, což jsou klasické výhody multi-repo přístupu.

Stávající monorepo obsahuje jasně oddělitelné části, které jsou pro tuto migraci ideální:

- **TypeScript:** Jádro projektu je TypeScript aplikace s rozsáhlým zdrojovým kódem a testy.
- **Rust:** Plán zmiňuje existující prototyp v adresáři rust/. Architektonická analýza pro Android navíc podrobně popisuje vysoce výkonnou Rust implementaci (mcp-prompts-rs), která je klíčová pro mobilní strategii.
- **PostgreSQL:** Plán zmiňuje adresář storage/postgres/. PostgresAdapter je komplexní komponenta s vlastním databázovým schématem , což plně ospravedlňuje její osamostatnění.
- **Android:** Plán zmiňuje adresář android/. Architektonická analýza odhaluje komplexní komponentu zahrnující AIDL, nativní Rust službu a samotnou Android aplikaci. Tato vysoce specializovaná část bude z oddělení do vlastního repozitáře těžit nejvíce.

1.4. Sdílený kontrakt (mcp-prompts-contracts)

Návrh zmiňuje sdílený kontrakt definovaný pomocí OpenAPI a JSON Schema, který bude pravděpodobně umístěn v balíčku mcp-prompts-contracts . Toto je základní kámen úspěšné multi-repozitářové architektury. Definice API a datových struktur v jazykově agnostickém formátu minimalizuje riziko integračních chyb a dramaticky zjednodušuje onboarding pro nové implementace, jako je plánovaný Python port (mcp-prompts-py) . Existence packages/mcp-prompts-contracts je potvrzena v package-lock.json a packages/README.md , což ukazuje, že projekt již směřuje k formalizaci svých API kontraktů.
Úspěch celého modelu závisí na disciplíně, s jakou budou tyto kontrakty spravovány. Jakákoli zpětně nekompatibilní změna v kontraktu nebo datové kolekci musí být považována za "major" verzi a pečlivě koordinována napříč všemi závislými repozitáři. Role meta-repozitáře tedy není jen orchestrovat buildy, ale především řídit a spravovat evoluci těchto sdílených kontraktů.

Faktor	Současný stav (Monorepo)	Navrhovaný stav (Multi-repo)	Riziko a mitigace	Podpůrný zdroj
Správa závislostí	Jednotný package.json , snadná správa interních závislostí.	Jasně definované externí závislosti, každá implementace spravuje své vlastní.	Riziko: Různé verze sdílených balíčků (version skew). Mitigace: Meta-repo generuje tabulku kompatibility.	
Rychlost CI/CD	Jediný ci.yml spouští všechny testy, což může být pomalé a nákladné.	CI běží pouze pro změněnou komponentu, což vede k rychlejším a levnějším pipeline.	Riziko: Složitější orchestrace cross-repo testů. Mitigace: Meta-repo spouští integrační testy "suite".	
Vlastnictví a přístup	Všichni přispěvatelé mají přístup k celému kódu.	Jemně granulovaná přístupová práva na úrovni jednotlivých repozitářů.	Riziko: Fragmentace znalostí. Mitigace: Centralizovaná dokumentace v meta-repozitáři.	
Atomické změny	Snadné refaktorování napříč komponentami v jediném commitu.	Změny napříč komponentami vyžadují koordinované cross-repo pull requesty.	Riziko: Zvýšená koordinační režie. Mitigace: Navržená orchestrace tento proces částečně zjednodušuje.	
Onboarding (Python)	Přidání nového jazyka komplikuje centrální build systém.	Nový tým může založit repozitář a začít pracovat pouze se závislostmi na collection a contracts.	N/A	

II. Fázový migrační a implementační plán

Pro zajištění hladkého a bezpečného přechodu je doporučen fázový, inkrementální přístup namísto rizikové "big bang" migrace. Tento postup umožňuje postupné ověřování funkčnosti a minimalizuje dopad na probíhající vývoj.

2.1. Fáze 0: Příprava a nástroje

1. **Instalace git-filter-repo:** Nainstalujte a nakonfigurujte nástroj git-filter-repo, který bude použit pro rozdělení historie Gitu.
2. **Vytvoření zrcadlové zálohy:** Vytvořte kompletní zrcadlovou zálohu původního repozitáře sparesparrow/mcp-prompts. Tento krok je klíčový pro bezpečnost a umožňuje opakované spouštění extrakčních skriptů bez rizika poškození originálu.
   ```
   git clone --mirror https://github.com/sparesparrow/mcp-prompts.git mcp-prompts.mirror
   ```

3. **Definice strategie verzování:** Zaveďte jednotnou strategii verzování (např. sémantické verzování) a větvení (např. GitFlow) pro všechny nově vzniklé repozitáře.

2.2. Fáze 1: Vytvoření základních repozitářů

1. **Inicializace mcp-prompts (meta-repo):** Založte na GitHubu prázdný repozitář mcp-prompts. Naplňte jej základními soubory README.md, CONTRIBUTING.md a architektonickou dokumentací z adresáře docs/ původního repozitáře.
2. **Extrakce mcp-prompts-collection:** Vytvořte nový repozitář mcp-prompts-collection. Pomocí git-filter-repo a zrcadlové zálohy extrahujte historii adresářů prompts/ a packages/mcp-prompts-catalog/. Nakonfigurujte CI pro publikaci jako NPM balíček a Cargo crate.
3. **Extrakce mcp-prompts-contracts:** Vytvořte repozitář mcp-prompts-contracts. Extrahujte historii relevantních souborů s TypeScript rozhraními (src/interfaces.ts , src/schemas.ts) a existující struktury packages/mcp-prompts-contracts. Nakonfigurujte CI pro publikaci jako verziovaný NPM balíček.

2.3. Fáze 2: Migrace TypeScript implementace (mcp-prompts-ts)

1. **Přejmenování původního repozitáře:** Přejmenujte původní repozitář sparesparrow/mcp-prompts na mcp-prompts-ts. Tímto krokem zůstane zachována kompletní historie, issues a pull requesty.
2. **Refaktorizace a vyčištění:** Upravte package.json a odstraňte z něj a z adresářové struktury všechny části, které nesouvisí s TypeScript implementací (např. rust/, android/). Odstraňte také adresáře extrahované v Fázi 1.
3. **Aktualizace závislostí:** V package.json nahraďte lokální odkazy novými, verziovanými závislostmi na @sparesparrow/mcp-prompts-collection a @sparesparrow/mcp-prompts-contracts z NPM.
4. **Integrace s meta-repozitářem:** Přidejte do .github/workflows GitHub Action, která po úspěšném release odešle událost repository_dispatch (nebo lépe, zavolá znovupoužitelné workflow) do meta-repozitáře.

2.4. Fáze 3: Extrakce specializovaných implementací

Pro každou z následujících komponent vytvořte nový repozitář a pomocí git-filter-repo extrahujte historii příslušného podadresáře ze zrcadlové zálohy.

- **mcp-prompts-rs (Rust):**
  - Extrahujte adresář rust/.
  - Přidejte mcp-prompts-collection jako Cargo závislost.
  - Nakonfigurujte CI pro build crate, spuštění testů a publikaci Docker image na GHCR.
  - Přidejte workflow pro orchestraci s meta-repozitářem.
- **mcp-prompts-pg (PostgreSQL):**
  - Extrahujte adresář storage/postgres/, kód PostgresAdapter z src/adapters.ts a inicializační skripty z docker/postgres/init/.
  - Tato komponenta bude publikována jako samostatný, verziovaný NPM balíček a bude obsahovat Helm chart pro nasazení v Kubernetes.
- **mcp-prompts-aidl (Android):**
  - Extrahujte adresáře android/ a android_app/.
  - Jedná se o komplexní komponentu s vlastním build procesem (Gradle a Cargo). CI pipeline musí být přizpůsobena pro build Android aplikací a nativních Rust knihoven.

Fáze	Repozitář	Klíčová akce	Ověřovací krok
1	mcp-prompts-collection	Extrahovat historii prompts/ pomocí git-filter-repo.	Ověřit, že historie souboru prompts/sequential-data-analysis.json je kompletní.
1	mcp-prompts-contracts	Extrahovat historii src/interfaces.ts a src/schemas.ts.	Ověřit, že balíček lze publikovat na NPM a importovat.
2	mcp-prompts-ts	Přejmenovat původní repo; nahradit lokální prompts/ závislostí z NPM.	Spustit testy a ověřit, že server funguje s prompty načtenými z NPM balíčku.
3	mcp-prompts-rs	Extrahovat historii rust/.	Sestavit crate a spustit testy, které konzumují mcp-prompts-collection crate.
3.1	mcp-prompts	Implementovat orchestrační workflow.	Spustit release mcp-prompts-rs a ověřit, že meta-repo správně zareagovalo (sestavilo suite image).

III. Strategie pro cross-repozitářovou CI/CD orchestraci

Volba správného mechanismu pro orchestraci CI/CD procesů napříč repozitáři je klíčová pro

úspěch a udržitelnost celého ekosystému.

3.1. Analýza repository_dispatch

Navrhovaný přístup využívá události repository_dispatch, které implementační repozitáře posílají do meta-repozitáře . Tento model je vysoce oddělený (decoupled) – implementační repozitář nemusí vědět, co se stane po odeslání události, což podporuje autonomii.
Hlavní nevýhodou je však **nedostatečná viditelnost**. Vývojář provádějící pull request v mcp-prompts-rs neuvidí stav navazujícího orchestračního workflow v mcp-prompts přímo v záložce "Checks". To komplikuje ladění a snižuje přehlednost celého procesu. Dále je přenos dat omezen na client-payload a bezpečnostní model může být příliš volný.

3.2. Alternativa 1: Znovupoužitelné workflow (workflow_call)

Lepší alternativou je definovat v meta-repozitáři znovupoužitelné workflow a z CI pipeline jednotlivých implementací toto workflow volat. Tento přístup nabízí několik výhod:

- **Plná viditelnost:** Workflow v mcp-prompts-rs zobrazí kompletní graf exekuce, včetně jobů ze znovupoužitelného workflow v mcp-prompts, což poskytuje jednotný a přehledný pohled.
- **Strukturované vstupy a secrets:** Umožňuje předávat silně typované vstupy a bezpečně sdílet secrets pomocí secrets: inherit, což je robustnější než generický JSON payload.

Nevýhodou je těsnější propojení (tighter coupling), kdy je workflow v implementačním repozitáři explicitně závislé na konkrétním souboru v jiném repozitáři.

3.3. Alternativa 2: Kompozitní akce

Pro menší, opakující se sekvence kroků (např. "přihlášení do Docker Hubu" nebo "publikace na NPM") je ideální vytvořit kompozitní akce v meta-repozitáři. Tento přístup redukuje duplikaci kódu v CI souborech jednotlivých implementací, ale není vhodný pro hlavní orchestrační logiku, která vyžaduje více jobů.

Doporučení: Hybridní orchestrační model

Volba mezi repository_dispatch a workflow_call je volbou mezi **událostmi řízenou choreografií** a **centrální orchestrací**. Vzhledem k tomu, že hlavním účelem meta-repozitáře je právě orchestrace, je model workflow_call vhodnější. Nedostatek viditelnosti u repository_dispatch představuje významný problém pro vývojářskou zkušenost.
Doporučuje se proto hybridní přístup:

1. **Znovupoužitelné workflow (workflow_call)** pro hlavní release orchestraci. Release workflow v mcp-prompts-rs zavolá znovupoužitelné workflow v mcp-prompts, které se postará o sestavení "suite" image a generování tabulky kompatibility.
2. **Kompozitní akce** definované v mcp-prompts pro sdílení běžných CI kroků (např. setup Node.js, login do registru) napříč všemi implementačními repozitáři.
3. **repository_dispatch** vyhradit pro skutečně asynchronní, nekritické události, jako je například noční spuštění regenerace dokumentace.

IV. Zachování historie revizí: Strategie s git-filter-repo

Zachování historie commitů při dělení repozitáře je klíčové pro udržení kontextu a sledovatelnosti změn. Nástroj git-filter-repo je pro tento úkol moderní a doporučenou volbou.

4.1. Protokol pro rozdělení repozitáře

Následující kroky popisují bezpečný postup pro extrakci podadresáře do nového repozitáře se zachováním historie.

1. **Klonování zrcadla:** Vytvořte lokální zrcadlovou kopii původního repozitáře. Všechny operace budou prováděny na této kopii, aby byl originál chráněn.
   ```
   git clone --mirror https://github.com/sparesparrow/mcp-prompts.git
   mcp-prompts.mirror
   ```

2. **Filtrování historie:** Pro každý nový repozitář (např. mcp-prompts-rs) vytvořte kopii zrcadla a spusťte git-filter-repo.
   ```
   # Vytvořit kopii pro Rust repozitář
   cp -r mcp-prompts.mirror/ mcp-prompts-rs.git
   cd mcp-prompts-rs.git

   # Spustit filtr pro zachování pouze adresáře 'rust/'
   git-filter-repo --path rust/
   ```

3. **Přepsání cest:** Soubory se stále nacházejí v adresáři rust/. Přepište historii tak, aby se přesunuly do kořenového adresáře.
   ```
   git-filter-repo --path-rename rust/:''
   ```

4. **Vyčištění a push:** Nástroj automaticky odstraní původní remote. Vytvořte nový prázdný repozitář na GitHubu a nahrajte do něj přefiltrovanou historii.
   ```
   git remote add origin
   https://github.com/sparesparrow/mcp-prompts-rs.git
   git push --all --force
   git push --tags --force
   ```

5. **Ověření:** Po nahrání naklonujte nový repozitář a zkontrolujte historii (git log) konkrétního souboru, který byl původně v podadresáři rust/. Historie by měla být kompletní a správná.

V. Analýza rizik a mitigace

Přechod na multi-repozitářovou architekturu s sebou nese specifická rizika, která je nutné identifikovat a řídit.

- **5.1. Riziko: Komplexní správa závislostí**
  - **Problém:** Správa verzí sdíleného balíčku mcp-prompts-collection napříč mnoha repozitáři může vést k tzv. "dependency hell".
  - **Mitigace:** Klíčem je tabulka kompatibility generovaná meta-repozitářem. Ta musí jasně definovat, které verze mcp-prompts-ts, -rs atd. jsou kompatibilní s jakou verzí mcp-prompts-collection. V každém implementačním repozitáři by měl být nakonfigurován nástroj jako Dependabot pro automatické vytváření PR na aktualizaci této závislosti.
- **5.2. Riziko: Složitost nástrojů a build procesů**

    - **Problém:** Každý repozitář bude mít vlastní sadu nástrojů (npm, cargo, gradle), což zvyšuje nároky na údržbu.
    - **Mitigace:** Toto je akceptovaný kompromis za získanou autonomii. Použití kompozitních akcí pro sdílené CI kroky a standardizace na Docker pro buildy, kde je to možné, může tuto zátěž snížit.
- **5.3. Riziko: Nekonzistentní verzování a release proces**
    - **Problém:** Různé repozitáře by mohly přijmout odlišné schémata pro verzování nebo tagování releasů.
    - **Mitigace:** Dokumentace v meta-repozitáři musí jasně definovat jednotný release proces pro celý ekosystém. Skript release.sh z původního repozitáře by měl být adaptován a zapouzdřen do kompozitní akce, aby se vynutil konzistentní postup.
- **5.4. Riziko: Objevitelnost kódu a onboarding**
    - **Problém:** Noví přispěvatelé nemusí vědět, ve kterém repozitáři mají pracovat.
    - **Mitigace:** README.md v meta-repozitáři musí sloužit jako kanonický vstupní bod. Musí obsahovat přehledný architektonický diagram a tabulku, která mapuje funkcionality na příslušné repozitáře s odkazy.

VI. Strategický závěr a doporučení

Navrhovaná migrace na multi-repozitářovou architekturu je správným a nezbytným strategickým krokem, který připraví ekosystém MCP Prompts na budoucí růst, škálovatelnost a přispění komunity. Plán správně identifikuje klíčové komponenty pro oddělení a definuje logickou cílovou strukturu.
Pro úspěšnou realizaci se doporučuje následující upřesněný postup:

1. **Přijmout inkrementální migrační strategii:** Postupujte fázově, abyste snížili rizika a doručovali hodnotu průběžně. Začněte extrakcí collection a contracts, ověřte funkčnost a teprve poté pokračujte s dalšími komponentami.
2. **Použít git-filter-repo podle detailního protokolu:** Zajistěte bezpečné rozdělení repozitáře se zachováním kompletní a neporušené historie revizí.
3. **Implementovat hybridní CI/CD orchestrační model:** Využijte znovupoužitelné workflow (workflow_call) pro hlavní orchestraci a kompozitní akce pro sdílené kroky. Tento model poskytuje nejlepší rovnováhu mezi centralizovanou kontrolou, viditelností a autonomií jednotlivých týmů.

Tato nová architektura vytváří "plug-and-play" ekosystém. Přidání nové jazykové implementace, jako je navrhovaný mcp-prompts-py , se stává přímočarým procesem: založení nového repozitáře, přidání závislostí na mcp-prompts-collection a mcp-prompts-contracts a implementace definovaného API kontraktu. Právě tato modularita je klíčem k dlouhodobému úspěchu a škálovatelnosti projektu.

Works cited

1. sparesparrow/mcp-prompts: Model Context Protocol server for managing, storing, and providing prompts and prompt templates for LLM interactions. - GitHub, https://github.com/sparesparrow/mcp-prompts 2. Mono Repo vs. Multi Repo in Git: Unravelling the key differences - Coforge, https://www.coforge.com/what-we-know/blog/mono-repo-vs.-multi-repo-in-git-unravelling-the-key-differences 3. MCP Prompts Server - Glama, https://glama.ai/mcp/servers/@sparesparrow/mcp-prompts 4. Git Filter-repo: Rewrite Git History

Like a Pro (Beginner's Guide) - YouTube, https://www.youtube.com/watch?v=_EcmY7_zlv0 5. A Comparison of Calling vs. Dispatching Workflows in GitHub Actions - Juraj's blog, https://jurajsim.hashnode.dev/a-comparison-of-calling-vs-dispatching-workflows-in-github-actions?source=more_articles_bottom_blogs 6. Reusing workflows - GitHub Docs, https://docs.github.com/en/actions/sharing-automations/reusing-workflows 7. Avoiding duplication - GitHub Docs, https://docs.github.com/en/actions/sharing-automations/avoiding-duplication 8. The Ultimate Guide to GitHub Reusable Workflows: Maximize Efficiency and Collaboration, https://www.dhiwise.com/post/the-ultimate-guide-to-github-reusable-workflows-maximize-efficiency-and-collaboration 9. Boost Your CI/CD with GitHub Actions: Composite Actions vs Reusable Workflows, https://www.ewere.tech/blog/boost-your-ci-cd-with-github-actions-composite-actions-vs-reusable-workflows/ 10. Composite Actions vs Reusable Workflows: what is the difference? [GitHub Actions], https://dev.to/n3wt0n/composite-actions-vs-reusable-workflows-what-is-the-difference-github-actions-11kd 11. Creating a composite action - GitHub Docs, https://docs.github.com/en/actions/sharing-automations/creating-actions/creating-a-composite-action 12. How to change git history with git-filter-repo - Krzysztof Marczewski, https://selfformat.com/blog/2025/02/26/how-to-change-git-history-with-git-filter-repo/ 13. Using the git filter-repo tool - Graphite, https://graphite.dev/guides/git-filter-repo 14. www.git-tower.com, https://www.git-tower.com/learn/git/faq/git-filter-repo#:~:text=Instead%20of%20starting%20a%20new,of%20your%20commit%20history%20intact.&text=This%20method%20safely%20eliminates%20sensitive,recorded%20commit%20in%20the%20repository. 15. Git Filter-Repo: The Best Way to Rewrite Git History, https://www.git-tower.com/learn/git/faq/git-filter-repo 16. Filter and copy files from one git repo to another and keep git history | Alex Hollis, https://alexanderhollis.com/git/2023/09/11/filter-and-copy-files-from-one-git-repo-to-another-and-keep-git-history.html 17. git - Selectively move files and directories to a new repository (preserving history), https://stackoverflow.com/questions/69946651/selectively-move-files-and-directories-to-a-new-repository-preserving-history 18. Monorepo vs. multi-repo: Different strategies for organizing repositories - Thoughtworks, https://www.thoughtworks.com/en-us/insights/blog/agile-engineering-practices/monorepo-vs-multirepo 19. modelcontextprotocol/python-sdk: The official Python SDK ... - GitHub, https://github.com/modelcontextprotocol/python-sdk