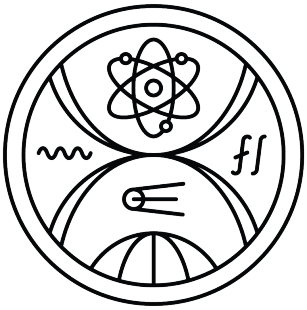
UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

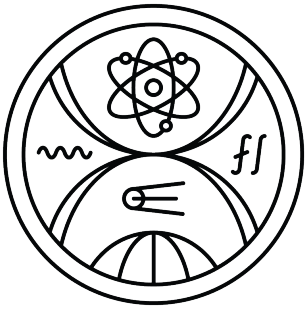


DATASETY V SOFTVÉROVOM INŽINIERSTVE

Diplomová práca

2025 Bc. Jakub Murin

UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY



DATASETY V SOFTVÉROVOM INŽINIERSTVE

Diplomová práca

Študijný program:	Aplikovaná informatika
Študijný odbor:	Informatika
Školiace pracovisko:	Katedra aplikovanej informatiky
Školiteľ:	Ing. Lukáš Radoský

Bratislava, 2025 Bc. Jakub Murin

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Bc. Jakub Murin
Študijný program: aplikovaná informatika (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor: informatika
Typ záverečnej práce: diplomová
Jazyk záverečnej práce: slovenský
Sekundárny jazyk: anglický

Názov: Datasety v softvérovom inžinierstve
Datasets in software engineering

Anotácia: V softvérovom inžinierstve prebieha výskum efektívnych metód transformácie niektorých artefaktov vznikajúcich v rámci vývoja softvéru na iné artefakty vývoja softvéru. Napriek týmto snahám však vo výskumnej komunite často chýbajú vhodné datasety pre evaluáciu týchto metód. Metódy sú tak evaluované na netransparentných dátach v malých objemoch, ak vôbec sú evaluované. Rôzne metódy sú tak ťažko porovnateľné. Príkladom takejto úlohy je transformácia prípadov použitia na UML diagramy alebo zdrojový kód.

Zvoľte si úlohu softvérového inžinierstva a vhodným postupom vytvorte strojovo spracovateľný dataset, na ktorom bude možné rôzne metódy či prístupy riešiace danú úlohu evaluovať. Dataset adekvátne zdokumentujte a opíšte, vrátane štatistických ukazovateľov. Využite vytvorený dataset na evaluáciu niektorej z existujúcich metód na riešenie danej úlohy, prípadne využite vlastnú, hoci aj triviálnu metódu.

Cieľ: Vytvorenie datasetu alebo viacerých datasetov pre vybranú úlohu alebo úlohy softvérového inžinierstva

Literatúra:
Frank Breitinger, Alexandre Jotterand, Sharing datasets for digital forensic: A novel taxonomy and legal concerns, Forensic Science International: Digital Investigation, Volume 45, Supplement,2023,301562, ISSN 2666-2817.
national Conference on Model Driven Engineering Languages and Systems: Companion Proceedings (MODELS '22). Association for Computing Machinery, New York, NY, USA, 396–403.

The Quest for Open Source Projects that Use UML: Mining GitHub; Hebig, R. & Ho-Quang, T. & Robles, G. & Fernandez, M.A. & Chaudron, M.R.V. (2016). In proceedings, ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems, pages 173-183, Saint-Malo, France, October 2-7, 2016.

Alyami A, Pileggi SF, Sohaib O, Hawryszkiewycz I. 2023. Seamless transformation from use case to sequence diagrams. PeerJ Computer Science 9:e1444 https://doi.org/10.7717/peerj-cs.1444

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Song Yang and Houari Sahraoui. 2022. Towards automatically extracting UML class diagrams from natural language specifications. In Proceedings of the 25th Inter

Kľúčové slová: Dataset, Budovanie datasetov, Softvérové inžinierstvo, Vývoj softvéru

Vedúci: Ing. Lukáš Radoský
Katedra: FMFI.KAI - Katedra aplikovanej informatiky
Vedúci katedry: doc. RNDr. Tatiana Jajcayová, PhD.

Spôsob sprístupnenia elektronickej verzie práce:
bez obmedzenia

Dátum zadania: 08.10.2023

Dátum schválenia: 11.11.2023 prof. RNDr. Roman Ďurikovič, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Čestne prehlasujem, že túto diplomovú prácu som vypracoval samostatne len s použitím uvedenej literatúry a za pomoci konzultácií u môjho školiteľa.

. . . . . .

Bratislava, 2025 Bc. Jakub Murin

Poďakovanie: Chcem sa poďakovať ...

Abstrakt

V oblasti softvérového inžinierstva sa výskum zameriava na efektívne metódy transformácie rôznych artefaktov, ktoré vznikajú počas vývoja softvéru, na iné artefakty. Napriek týmto snahám však v tejto oblasti často chýbajú vhodné dátové sady na vyhodnotenie týchto metód, čo vedie k evaluácii metód na netransparentných a malých množstvách dát, prípadne k ich úplnej absencii. Tieto metódy je preto ťažké porovnávať. V našej práci sme tvoríme dátovú sadu pozostávajúcu z dvojíc prípad použitia a sekvenčný diagram. Dáta získavame z repozitárov z GitHub, z dostupných internetových úložísk, generovaním umelou inteligenciou a kombináciou predchádzajúcich metód. Dôraz kladieme na dôkladnú dokumentáciu, vrátane štatistických ukazovateľov, čím sa umožní jej efektívne využitie pre ďalší výskum a porovnávanie rôznych prístupov transformácie.

Kľúčové slová: Dataset, Budovanie datasetov, Softvérové inžinierstvo, Vývoj softvéru, PlantUML

Abstract

In the field of software engineering, research focuses on effective methods for transforming various artifacts created during software development into other artifacts. Despite these efforts, there is often a lack of suitable datasets for evaluating these methods, leading to evaluations based on non-transparent and small data sets, or even their complete absence. As a result, comparing these methods becomes challenging. In our work, we have created a dataset consisting of pairs of use cases and sequence diagrams. The data was collected from repositories on GitHub, available online storage, generated by artificial intelligence, and through a combination of the previous methods. We placed emphasis on thorough documentation, including statistical indicators, which enables the effective use of this dataset for further research and the comparison of different transformation approaches.

Keywords: Dataset, Dataset construction, Software engineering, Software development, PlantUML

Obsah

Úvod **1**

1 Teória **3**
1.1 Umelá inteligencia 3
1.2 Strojové učenie 4
1.3 Neurónové siete 4
1.4 Veľké jazykové modely 5
1.4.1 ChatGPT 5
1.4.2 Gemini 6
1.4.3 Copilot 6
1.4.4 Porovnanie 7

2 Prípad použitia **9**
2.1 Dobrý prípad použitia 10
2.1.1 Postup vytvárania s príkladom 10

3 Sekvenčný diagram **13**
3.1 PlantUML 14
3.1.1 Sekvenčný diagram v PlantUML 15

4 Súvisiace práce **19**
4.1 Seamless transformation from use case to sequence diagrams 19
4.2 Transformation Method from Scenario to Sequence Diagram 20
4.3 An Extensive Dataset of UML Models in GitHub 21
4.4 Data Cards: Purposeful and Transparent Dataset Documentation for Responsible AI 21

5 Návrh **23**

6 Výskum **25**
6.1 Identifikácia PlantUML sekvenčného diagramu 25
6.2 Identifikácia sekvenčného diagramu v .xmi 27

6.3 Získanie súborov z GitHubu 27

7 Diskusia **29**

Záver **31**

Zoznam obrázkov

1.1 Kategorizácia umelej inteligencie 4

3.1 Diagram generovaný z ukážky kódu 3.1 17

4.1 Diagram prípadu použitia z článku 20

6.1 Postupnosť krokov získavania dát 26

Zoznam tabuliek

1.1 Porovnanie parametrov dostupných Chatbotov 7

Zoznam ukážok kódu

3.1 Ukážka prvkov sekvenčného diagramu 16
6.1 Hľadanie súborov so sekvenčným diagramom 25
6.2 Hľadanie výhradne plantuml súborov so sekvenčným diagramom 26
6.3 Jednoduchý sekvenčný diagram 26
6.4 URL na raw súbory UML z daných repozitárov 27

Úvod

Uvádzam túto prácu.

Kapitola 1

Teória

Existujú viaceré spôsoby ako získať väčšie množstvá diagramov, a jedným z nich je využite umelej inteligencie. V ideálnom prípade by sme mali umelú inteligenciu špeciálne trénovanú na vytváranie a spracovanie diagramov, avšak taká verejne dostupná nie je.

1.1 Umelá inteligencia

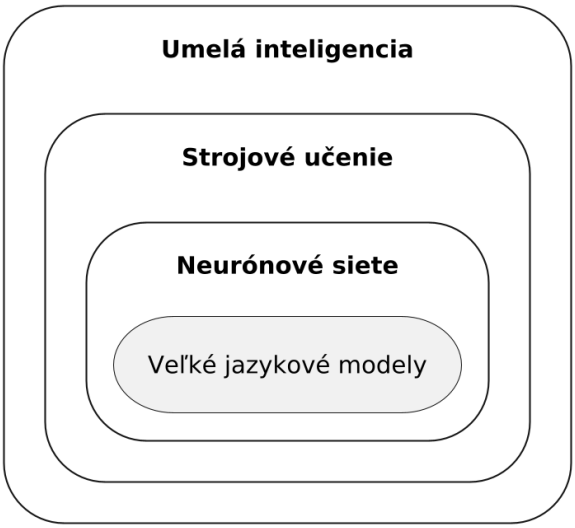
Pojem umelá inteligencia reprezentuje akúkoľvek techniku, ktorá umožňuje strojom napodobňovať inteligenciu podobnú ľuďom, vrátane uvažovania, učenia, plánovania a riešenia problémov. Umelá inteligencia zahŕňa rôzne prístupy, od systémov založených na pravidlách a symbolickej logiky až po zložitejšie modely založené na množstve údajov.

Typy umelej inteligencie [18]:

- Úzka: Systémy navrhnuté na vykonávanie špecifických úloh, ako sú systémy rozpoznávania tváre alebo odporúčaní. Väčšina dnes používanej AI patrí do tejto kategórie.

- Všeobecná: Úroveň AI, ktorá môže vykonávať akúkoľvek intelektuálnu úlohu, ktorú dokáže človek. Nemá konkrétne zameranie a je schopná riešiť úlohy naprieč oblasťami.

- Superinteligentná: Jedná sa o systém, ktorý je plne sebavedomý, prevyšuje ľudskú inteligenciu a mal by mať schopnosť sa neustále zlepšovať. Obohatená ľudskými charakteristikami a podporená výpočtovou a analytickou silou, by mohla výrazne prekonať schopnosti ľudstva.



Obr. 1.1: Kategorizácia umelej inteligencie

1.2 Strojové učenie

Strojové učenie je podmnožinou umelej inteligencie, ktorá sa zameriava na optimalizáciu a predpovedanie na základe dát s cieľom minimalizovať chyby. Klasické strojové učenie vyžaduje ľudský zásah pri rozpoznávaní vzorov a učení sa na základe štruktúrovaných údajov. Hlboké strojové učenie dokáže pracovať aj s neštruktúrovanými dátami a automaticky identifikovať kľúčové vlastnosti. Okrem toho existuje učenie posilnením, kde sa počítač učí prostredníctvom interakcie so svojím okolím a reaguje na získanú spätnú väzbu. Online učenie umožňuje neustále aktualizovanie modelov strojového učenia, na základe nových dostupných dát [4].

1.3 Neurónové siete

Neurónové siete sú podmnožinou strojového učenia a základom hlbokého učenia. Sú vytvorené na princípe fungovania živočíšnej ríše, keď napodobňujú spôsob, akým komunikujú neuróny v mozgu. Skladajú sa z vrstiev uzlov, ktoré spracúvajú dáta na základe váh a prahových hodnôt, pričom výsledkom je číselné vyjadrenie, ktorá možnosť je najpravdepodobnejšia. Tieto siete sa trénujú pomocou množstva dát, aby sa zlepšila ich presnosť a umožnili sa vykonávať náročnejšie úlohy, ako je rozpoznávanie reči a obrázkov. Neurónové siete sa v súčasnosti používajú v mnohých oblastiach, z ktorých je najznámejší Google vyhľadávací algoritmus [4].

1.4 Veľké jazykové modely

Radia sa do kategórie neurónových sietí a teda sa jedná o strojové učenie, ako je znázornené v obrázku 1.1. Veľké jazykové modely (LLM) sú rozsiahle hlboké učebné modely, ktoré používajú architektúru transformera na spracovanie textu. Skladá sa z kódera a dekódera, ktoré pomáhajú porozumieť vzťahom medzi slovami a frázami. Vďaka paralelnému spracovaniu sekvencií s využitím GPU sa tréning modelov LLM výrazne urýchľuje. Tieto modely môžu obsahovať stámiliardy parametrov a sú schopné spracovať veľké množstvá dát, často z internetu, čo im umožňuje pochopiť gramatiku a základné vedomosti z rôznych jazykov [2].

Najväčšie softvérové spoločnosti poskytujú svoje modely verejnosti a ich výhoda spočíva v tom, že sú trénované na množstve zozbieraných dát z rôznych oblastí. Nasledujúcich podkapitolách sa zameriame na niektoré z najpoužívanejších.

Nevýhodou týchto modelov je, že niekedy odpovedajú hodnoverne znejúce, ale neprávne odpovede. Zároveň ich ovplyvňuje formulácia položeného dopytu, kde pri malej zmene môže byť rozdielna odpoveď. Vyplýva to z princípu fungovania neurónových sietí, kde výsledok je odhadovaný.

1.4.1 ChatGPT

ChatGPT[1] je jazykový model, ktorý vyvinula spoločnosť OpenAI ako pridružený model InstructGPT, ktorý je zameraný na dodržiavanie zadaných pokynov. Je navrhnutý na interakciu s používateľmi v prirodzenom jazyku pomocou chatovacieho rozhrania. OpenAI poskytuje prístup k ChatGPT aj cez API rozhranie.

Model bol trénovaný pomocou Reinforcement Learning from Human Feedback (RLHF). Počiatočný model bol dolaďovaný za pomoci simulovaných rozhovorov medzi ľudmi, v ktorých predstavovali obe strany: používateľa aj AI asistenta. Následne školitelia mali možnosť upraviť návrhy odpovedí zostavených umelou inteligenciou, a tak pomohli vytvoriť vzorce pre lepšie odpovede. Takto vzniknutý nový dialógový súbor údajov bol zmiešaný so súborom údajov InstructGPT [12]. Ďalej bol zvolený systém hodnotenia, ktorý z viacerých odpovedí na rovnaký dopyt zvolil tú najlepšiu. Tento cyklus bol viacnásobne opakovaný.

Vzhľadom na jeho simuláciu ľudského vyjadrovania je vhodný na vytváranie prípadov použitia, ktoré sú písané v prirodzenom jazyku. Poskytované API umožňuje automatické generovanie a vykonávanie dopytov pomocou programu.

[1] `https://chatgpt.com/`

1.4.2 Gemini

Jazykový model od spoločnosti Google dostal názov Gemini[2]. Rovnako ako v prípade ChatGPT jedná sa o interakciu prirodzeným jazykom cez chat.

Je trénovaný nad biliónmi slov, na základe ktorých si vytvoril vzorce tvoriace jazyk, ktoré využíva pri odpovediach. Neustále sa učí z dopytov, odpovedí a dostávanej spätnej väzby [6]. Gemini je výsledkom rozsiahleho spoločného úsilia tímov naprieč Google, vrátane tímov z Google Research. Bol vytvorený od základov ako multimodálny, čo znamená, že dokáže zovšeobecniť, bez problémov pochopiť a kombinovať rôzne typy informácií vrátane textu, kódu, zvuku, obrazu a videa [13].

Gemini Ultra je prvý model, ktorý prekonal ľudských expertov pri meraní MMLU (Massive Multitask Language Understanding). Ide o meradlo vedomostí a schopnosti riešiť problémy v 57 odvetviach vrátane matematiky, fyziky, histórie, práva, medicíny, etiky, a ďalších [7]. Toto meradlo je podobné tomu, ako sa vyhodnocujú vedomosti ľudí a pre umelú inteligenciu sa jedná o jeden z náročnejších testov.

Výhodou je, že neplatená verzia je rýchlejšia v porovnaní s konkurenciou a existuje API prístup.

1.4.3 Copilot

Copilot[3] je produkt spoločnosti Microsoft, ale je postavený na modeli GPT-4 od spoločnosti OpenAI.

Tak ako aj iné modely, aj Copilot je nastavený, aby sa naučil všeobecné vzťahy v jazyku, a nie aby si zapamätal konkrétne segmenty tréningových dát. Neukladá si ani nemá prístup k pôvodným tréningovým dátam. Sú podnikané kroky, aby bolo zabránené modelom neúmyselne reprodukovať tréningové údaje, ako je napríklad vytváranie filtrov, ktoré odstránia predtým publikovaný alebo použitý materiál.

Výhodou je, že Copilot vyhľadáva relevantný obsah na webe a potom zosumarizuje informácie, ktoré nájde, aby vytvoril užitočnú odpoveď. Súčasťou odpovede sú aj odkazy na webové stránky, odkiaľ získal informácie [10]. Vďaka tomu sa môžme rýchlejšie dopracovať k zdroju, z ktorého sa dá čerpať.

Cieľom Microsoftu bolo vytvoriť efektívneho asistenta, a preto je maximálny limit dopytov v konverzácii 30. To môže mať za následok, že pri generovaní prípadov použitia a zdrojového zápisu pre sekvenčný diagram, nebudeme dostatočne vedieť poopraviť získanú odpoveď.

[2] `https://gemini.google.com/`
[3] `https://copilot.microsoft.com/`

1.4.4 Porovnanie

Pre väčšiu prehľadnosť sa v tomto porovnaní pozrieme na kľúčové faktory ako sú cenové možnosti, dostupnosť API, limity správ, a iné, aby sme zistili, ktorý model je najvhodnejší pre potreby našej práce.

Základom porovnávania je od každej spoločnosti najlepšie finančne dostupná možnosť. Teda od OpenAI to bude model 4o-mini, od Google Gemini 1.5 Flash a od Microsoftu Copilot.

Všetky modely poskytujú grafické rozhranie chatu na webovej stránke, kde sa dá s nimi viesť bežná konverzácia. Nevýhoda všetkých je, že v prípade zmeny kontextu je potrebné začať novú konverzáciu. Copilot má limit správ na jednu konverzáciu 30. ChatGPT je obmedzený časovo a to na 80 správ za tri hodiny. Jedine Gemini nemá obmedzené množstvo. API rozhranie poskytuje ChatGPT 4o-mini a Gemini 1.5 Flash, zatiaľ čo pre Copilot neexistuje verejne dostupná API, jedine cez prídavné moduly. Pri používaní API je každý dopyt izolovaný, a teda ak chceme nadviazať na odpoveď alebo konverzáciu, je potrebné to poslať v rámci dopytu ako kontext. Veľkosť správ je počítaná na tokeny, pričom jeden token predstavuje približne 3/4 slova. ChatGPT má limit pre kontext 128 tisíc tokenov a Gemini má niekoľkonásobne väčší kontext a to 1 milión tokenov. Copilot nemá špecifický limit, ale 4,096 tokenov je dokopy na kontext aj odpoveď. Z porovnávaných modelov má teda jednoznačne najmenšiu veľkosť kontextu. Veľkosť odpovede pre ChatGPT je 4,096 tokenov a pre Gemini dvojnásobok 8,192 tokenov. Všetky modely sú vhodné na generovanie kódu. Gemini a Copilot môžu využívať informácie z webu ľubovoľne, zatiaľ čo ChatGPT má limitovaný prístup.

Porovnávané atribúty sú zhrnuté v tabuľke 1.1.

Tabuľka 1.1: Porovnanie parametrov dostupných Chatbotov

	ChatGPT 4o-mini	Gemini 1.5 Flash	Copilot
spoločnosť	OpenAI	Google	Microsoft
jadro	GPT-3.5	Gemini 1	GPT-4
najnovší model	GPT-4	Gemini 2.0	Copilot Pro
chat	✓	✓	✓
limit správ	80 za 3 hodiny	neobmedzene	30 na chat
API	✓	✓	✗
veľkosť kontextu	128,000 tokenov	1,000,000 tokenov	4,096 tokenov
veľkosť odpovede	4,096 tokenov	8,192 tokenov	4,096 tokenov
generovanie kódu	✓	✓	✓
prístup k webu	limitovaný	✓	✓

Kapitola 2

Prípad použitia

Prípad použitia je popis možných interakcií medzi systémom a jeho externými aktérmi, ktoré sú zamerané na dosiahnutie konkrétneho cieľa [3]. Hlavným účelom prípadu použitia je predstaviť, ako systém interaguje so svojím okolím, bez zamerania sa na jeho vnútornú štruktúru. Je možné ho využiť aj na popísanie interného správania sa. Prípad použitia zachytáva správanie systému a jeho zodpovednosti, popisujúc rôzne scenáre, ktoré môžu nastať, keď sa aktér usiluje o dosiahnutie daného cieľa. Aktér môže predstavovať osobu, časť softvéru alebo iný systém. Prípad použitia definuje všetky možné sekvencie interakcií, vrátane tých, kde je cieľ dosiahnutý, a aj tých, kde sa cieľ nedá dosiahnuť.

Prípady použitia môžu byť vytvorené kedykoľvek počas vývoja softvéru. Boli úspešne aplikované v rôznych kontextoch, ako je dokumentovanie obchodných procesov, špecifikovanie požiadaviek na softvér alebo navrhovanie hardvérových a softvérových systémov. Sú obzvlášť užitočné na zachytenie správania systému spôsobom, ktorý je jasný a stručný, ukazujúci, ako systém reaguje na externé akcie, bez podrobností o jeho vnútornej implementácii alebo návrhu. Hoci môžu byť reprezentované pomocou rôznych formátov, ako sú diagramy tokov alebo Petriho siete, prípady použitia sú najčastejšie písané v prirodzenom jazyku. Je to jednoduché, prístupné a efektívne na komunikáciu, aj medzi osobami, ktoré nemajú formálne vzdelanie v informatike [3].

Často krát sa prípad použitia zamieňa s diagramom prípadov použitia, ale ten popisuje vzťahy medzi jednotlivými prípadmi použitia. Je akýmsi stručným prehľadom ako jednotlivé prípady použitia súvisia a aký aktéri v nich vystupujú. Je súčasťou jazyka UML.

Ako prvý popísal prípad použitia Ivar Jacobson vo svojej publikácii o objektovo orientovanom vývoji v priemyselnom prostredí. Fungovanie systému nie je možné zhrnúť do jedného popisu, preto je popísané po jednotlivých častiach, ktoré dokopy vytvárajú obraz o celom systéme. Prípad použitia pomáha definovať požiadavky na systém, podporuje komunikáciu medzi vývojármi a netechnickými osobami a slúži ako základ

pre návrh, implementáciu a testovanie systému. Jacobson zdôraznil, že tento prístup umožňuje efektívne riadenie vývoja softvéru v priemyselných projektoch a zaisťuje, že výsledný systém bude spĺňať požiadavky všetkých zúčastnených strán [9].

Na jeho myšlienky naviazalo množstvo autorov ako napríklad Martin Fowler a Alistair Cockburn. Fowler sa venuje vytváraniu príbehu používateľa (user story), ktorý sa zameriava na užšiu oblasť ako prípad použitia a používa sa pri agilnej metodológii vývoja softvéru [5].

Cockburn zas preferuje viac opisný zápis v textovej forme. Základnými princípmi tvorby sú, že chceme písať čo najmenej, čo najjasnejšie a ukázať spôsoby, akými systém reaguje na rôzne situácie.

2.1 Dobrý prípad použitia

V našej práci sa zameriame na Cockburnov prístup. V tejto kapitole sa pozrieme na to, ako vyzerá dobrý prípad použitia podľa Cockburna, a na aké znaky je potrené sa zamerať pri analýze získaných prípadov použitia [3].

2.1.1 Postup vytvárania s príkladom

Pri vytváraní prípadu použitia by sme sa mali zamerať viac na správnosť jednotlivých krokov a postupu, ako na zachytenie každej podrobnosti. Ak by sme mali presne opísať každú skutočnosť, prípad použitia sa stane neprehľadným a stráca sa podstata stručne zachytiť interakciu systému. Cockburn stanovil 12 krokový postup na vytvorenie prípadu použitia, pričom sa zachovávajú opísané koncepty [3]:

1. Nájdite hranice systému (Kontextový diagram, zoznam vstupov/výstupov).
2. Vytvorte zoznam hlavných aktérov.
3. Zostavte zoznamu cieľov hlavných aktérov voči systému. (Zoznam aktér-cieľ)
4. Vypíšte hraničné kľúčové používateľské prípady, ktoré pokrývajú všetky vyššie uvedené body.
5. Znova prehodnoťte a upravte hlavný používateľský prípad. Pridajte, odoberte alebo spojte ciele.
6. Vyberte cieľ, ktorý chcete rozvinúť. Voliteľne napíšte príbeh o systéme v praxi, aby ste sa oboznámili s príbehom používateľského prípadu.
7. Vyplňte zainteresované strany a ich záujmy, predpoklady, úspešné koncové podmienky, ochranu pred zlyhaním. Dvakrát skontrolujte ciele a záujmy v súvislosti s týmito podmienkami.

8. Napíšte hlavný úspešný scenár pre prípad použitia. Dvakrát skontrolujte, či zodpovedá záujmom zainteresovaných strán.

9. Vykonajte brainstorming a zostavte zoznam možných podmienok zlyhania.

10. Napíšte, ako by sa aktéri a systém mali obnoviť z každého zlyhania.

11. Odčleňte akúkoľvek časť, ktorá potrebuje samostatný prípad použitia.

12. Prejdite všetky kroky postupu odznova a pridajte, odoberte alebo spojte prípady použitia. Dvakrát skontrolujte úplnosť, čitateľnosť a podmienky zlyhania.

Tento prístup však nie je vhodný ako postup pre vytvorenie umelou inteligenciou, vzhľadom na to, že obsahuje viaceré kontroly a prehodnotenia, ktoré LLM nevie vykonať. Taktiež by sa dodržanie niektorých bodov ťažko overovalo v už existujúcom prípade použitia. Preto sme vytvorili vlastný zjednodušený postup s menším počtov krokov, ktorý ale značne vychádza z Cockburnovho a zostručňuje ho.

Postup vytvárania:

1. **Aktéri a ciele** - Najskôr je potrebné definovať, kto sa účastní prípadu použitia a čo má byť jeho cieľom. Aktérmi môžu byť ľudia, systémy, objekty alebo skupiny, pričom hlavý aktér je ten ktorý inicializuje akcie a sekundárni aktéri pomáhajú alebo priamo vykonávajú danú akciu.

2. **Hlavný úspešný scenár** - Postup krokov k dosiahnutiu stanoveného cieľa. Každý krok by mal obsahovať, kto vykonáva akciu, akú akcia je vykonávaná a čo je objektom akcie. Zjednodušene podstatné meno, sloveso a predmet.

3. **Alternatívne scenáre** - V prípade, že sa dá k cieľu dopracovať iným postupom ako popisuje hlavný scenár, je potrebné vytvoriť alternatívne scenáre. Pri ich vytváraní postupujeme ako pri hlavnom scenári s tým, že pokračujeme z niektorého dosiahnutého kroku hlavného scenára. V prípade častého vetvenia na alternatívne scenáre je vhodnejšie pre každý vytvoriť vlastný prípad použitia, aby sa zachovala prehľadnosť.

4. **Neúspešné prípady** - Keď nastáva situácia, že sa nepodarí dosiahnuť cieľ, vytvárame neúspešné prípady, ktoré popisujú postup zlyhania. Je to druh alternatívnych scenárov.

5. **Ošetrenie neúspešných prípadov** - V niektorých prípadoch je potrebné popísať čo sa má diať ak nastal neúspešný prípad, a tak sa vrátiť naspäť na hlavný scenár.

Ako príklad vytvorenia si uvedieme výber hotovosti z bankomatu.

- Hlavným aktérom je **zákazník**, a jeho cieľ je **vybrať hotovosť**. Sekundárnymi aktérmi sú **bankomat** a **platobná karta**, ktorú budeme skrátene nazývať karta.
- Vytvárame hlavný scenár:
  1. Zákazník vloží platobnú kartu do bankomatu.
  2. Bankomat načíta informácie z karty.
  3. Zákazník sa autentifikuje PIN kódom pre vloženú kartu.
  4. Bankomat zobrazí informácie o možnostiach výberu.
  5. Zákazník si vyberie obnos hotovosti, ktorý chce vybrať.
  6. Bankomat vykoná transakciu na zníženie zostatku na účte zákazníka o daný obnos.
  7. Bankomat vydá hotovosť zákazníkovi.
  8. Bankomat vysunie kartu.
  9. Zákazník si zoberie hotovosť a kartu.
- Alternatívne scenáre, kde pridávame sekundárneho aktéra **aplikáciu**:
  1. Zákazník si v aplikácii banky zvolí možnosť "Vybrať hotovosť".
  2. Aplikácia vygeneruje kód na identifikáciu.
  3. Zákazník v bankomate zvolí možnosť "Výber pomocou kódu".
  4. Zákazník zadá kód z aplikácie do bankomatu.
  5. Pokračuje sa krokom 4. v hlavnom scenári.

  V kroku 8. hlavného scenára sa môže pridať alternatíva, že bankomat vysunie kartu ak bola vložená.
- Neúspešný prípad môže nastať ak zákazník v hlavnom scenári v kroku 3. zadá zlý PIN.
  1. Zákazník sa neautentifikoval PIN kódom pre vloženú kartu.
- Ošetríme neúspešný prípad z predchádzajúceho bodu tak, že do neho pridáme ďalšie kroky:
  1. Bankomat vyzve zákazníka o znovu zadanie PIN kódu.
  2. Pokračuje sa krokom 3. v hlavnom scenári.

Kapitola 3

Sekvenčný diagram

Sekvenčný diagram je súčasťou skupiny diagramov používaných pri vývoji softvéru. Je tvorený v jazyku UML, čo je modelovací jazyk, ktorý je určený na poskytovanie štandardného spôsobu modelovania a vizualizácie návrhu systému. UML patrí pod správu organizácie OMG, ktorá je zodpovedná za definíciu viacerých štandardov v informatike. Sekvenčný diagram sa radí do kategórie behaviorálnych diagramov, ktoré popisujú správanie sa systému, procesov a interakciu medzi objektami. Ukazuje, ako objekty navzájom komunikujú a v akom poradí sú správy posielané medzi nimi počas určitého scenára alebo procesu. Zvykne zachytávať priebeh počas jedného prípadu použitia. Používa sa pri analýze požiadaviek a návrhu systému, aby bol zachytený tok informácií a interakcie medzi rôznymi komponentmi alebo objektmi. Skladá sa z viacerých prvkov, z ktorých sú hlavné [8]:

- **Objekt** - predstavuje aktéra, systém, konkrétnu inštanciu alebo komponent, ktorý vystupuje v interakcii. V diagrame je reprezentovaný obdĺžnikom, alebo postavičkou s názvom objektu. Poprípade môže byť vyjadrený aj iným symbolom. Objekty sa nachádzajú na vrchu diagramu, a tak má čitateľ rovno prehľad o objektoch vystupujúcich v diagrame. V prípade, že vznikne nový počas akcie, objekt začína od miesta vytvorenia.

- **Správa** - je v diagrame zobrazená vodorovnou šípkou vychádzajúcou z odosielateľa a smerujúcou do príjemcu. Odosielateľ a príjemca môže byť aj ten istý objekt. Typ šípky určuje typ správy, ako napríklad synchrónna správa, asynchrónna alebo odpoveď. Súčasťou je text, ktorý popisuje správu, napríklad aká metóda bola vyvolaná. Správy sú zoradené vertikálne zhora dole, aby ostala zachovaná časová postupnosť.

- **Životná čiara** - je vertikálna prerušovaná čiara predstavujúca objekt z ktorého vychádza. Umožňuje tak zápis správ ako vodorovných šípok, a tým zachováva jednoduchú čitateľnosť sekvencie.

- **Aktivačný blok** - sa nachádza na životnej čiare objektu v podobe obdĺžnika vyjadrujúceho aktiváciu daného objektu. V prípade zavolania metódy samého seba, vzniká ďalší aktivačný blok na rovnakej životnej čiare.

- **Fragment** - je znázornený obdĺžnikom cez časť diagramu. Predstavuje zloženú štruktúru ovplyvňujúcu vnútorné prvky. V ľavom hornom rohu sa nachádza kľúčové slovo, ktoré určuje funkcionalitu fragmentu. Toto sú niektoré z nich:

  - **Podmienka** - kľúčové slovo *opt* určuje či sa má daná oblasť vykonať na základe definovanej podmienky. V prípade viacerých nasledujúcich alternatív z ktorých sa vykoná najviac jedna sa používa *alt* v rámci jedného fragmentu. Vtedy je fragment delený vodorovnými čiarami, kde každá časť predstavuje určitú možnosť.
  - **Cyklus** - podmienka cyklu určuje dokedy sa má daná oblasť opakovať. Je reprezentovaný kľúčovým slovom *loop*.
  - **Paralelné vykonávanie** - v prípade paralelného vykonávania sa vo fragmente používa *par*. Predstavuje paralelné vykonávanie jednotlivých častí oddelených vodorovnou čiarou.

3.1 PlantUML

PlantUML je všestranný nástroj na rýchlu tvorbu diagramov rôzneho druhu. Pôvodný zámer bol tvorba UML diagramov, ktorá je aj najviac rozvinutá. Vychádza z konceptu diagram ako kód, a teda z intuitívneho textového zápisu nástroj vygeneruje diagram bez nutnosti grafického rozhrania. V prípade potreby grafickej reprezentácie je dostupný online server PlantUML, kde sa diagram generuje po každej zmene, alebo viaceré vývojové prostredia majú dispozícii prídavný modul s touto funkcionalitou. Jedná sa teda skôr o vizualizačný nástroj, ktorý zo vstupu vytvorí obrázok, ako kresliaci, v ktorom by sa priamo vytváral graficky diagram. Bol vytvorený, aby zjednodušil vytváranie a aktualizáciu diagramov v dokumentácii. Používateľovi stačí definovať objekty a väzby, nemusí venovať čas usporiadaniu prvkov a ich zobrazeniu, lebo PlantUML obsahuje algoritmy na čitateľné rozloženie. Nástroj je slobodný open-source softvér a je vyvíjaný komunitou, takže je bezpečný a vývoj reflektuje aktuálne potreby používateľov [15].

Okrem svojho špecifického jazyka podporuje aj generovanie z LaTex, AsciiMath a ďalších formátov textu. Postupom času bol rozšírený o viaceré typy diagramov vrátane modelovacieho jazyka ArchiMate. Podporuje aj vizualizáciu dát zo súborov typu yaml a json.

3.1.1 Sekvenčný diagram v PlantUML

PlantUML umožňuje jednoduché a efektívne vytváranie sekvenčných diagramov, pričom jeho syntax je navrhnutá tak, aby bola intuitívna a ľahko zapamätateľná. Hlavné výhody sú [14]:

- **Zhoda textu s grafikou:** Textový zdrojový kód presne zodpovedá grafickým výstupom, čo zaručuje konzistentný a deterministický výsledok bez neočakávaných zmien.
- **Efektívny proces tvorby:** Presná zhoda textu s výstupom zrýchľuje prácu a minimalizuje potrebu zložitých úprav.
- **Vizualizácia pri tvorbe:** Používateľ si dokáže už počas písania textu predstaviť výsledný diagram, čo znižuje chybovosť a zvyšuje produktivitu. Hoci väčšina editorov s podporou PlantUML prekresľuje diagram v reálnom čase, používateľ nemusí sledovať rozdiely v dvoch oblastiach.
- **Jednoduché úpravy:** Diagram sa edituje úpravou textu, čo je presnejšie a rýchlejšie než manuálne úpravy grafických súborov pomocou myši.
- **Automatické generovanie:** Čitateľný zápis umožňuje rýchle automatické generovanie zdrojového súboru pomocou programu. Používateľ si tak vie uľahčiť prácu napísaním jednoduchých skriptov.
- **Zdrojový kód v metadátach:** Ak je výsledný diagram generovaný vo formáte png, PlantUML uloží celý zdrojový kód, z ktorého bol diagram tvorený, do metadát obrázku. Nástroj zároveň umožňuje tieto dáta z metadát extrahovať do textového formátu.

Ukážka 3.1 zobrazuje zdrojový kód sekvenčného diagramu a následne obrázok 3.1 jeho grafickú podobu.

Každý PlantUML diagram začína *@startuml* a končí *@endtuml*. Ak chceme použiť špeciálne formátovanie, odporúča sa ho definovať na začiatok. Na riadku 2 sme aktivovali automatické vytváranie aktivačných blokov a na riadku 3 sme zakázali generovanie názvov objektov na spodnej časti životných čiar.

Na riadkoch 5-7 sú definované zapojené objekty, ktoré sa v diagrame zobrazia v poradí definovania. Každému objektu je možné pridať alias (riadok 7) pre skrátenie zápisu.

Nasleduje samotný diagram, ktorý je reprezentovaný zväčša správami vo formáte *[odosielateľ] [typ šípky] [príjemca] : [text správy]*. Pre synchrónnu správu je typ šípky -> a pre odpoveď - ->. Odpoveď je možné vyjadriť aj pomocou kľúčového slova *return*,

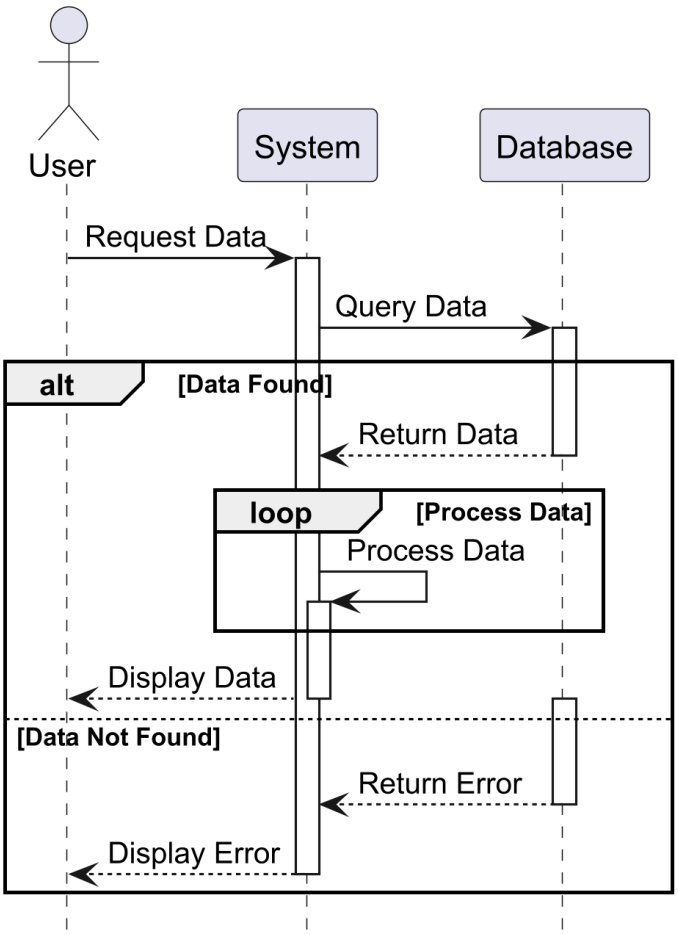
Ukážka kódu 3.1: Ukážka prvkov sekvenčného diagramu

```
1  @startuml
2  autoactivate on
3  hide footbox
4
5  actor User
6  participant System
7  participant "Database" as DB
8
9  User -> System : Request Data
10 System -> DB : Query Data
11 alt Data Found
12      return Return Data
13      loop Process Data
14          System -> System : Process Data
15      end
16      System --> User : Display Data
17 else Data Not Found
18      activate DB
19      System <-- DB : Return Error
20      User <-- System : Display Error
21 end
22 @enduml
```

avšak tá sa vráti na životnú čiaru, ktorá bola posledná aktivovaná (riadok 12). Z dôvodu väčšej čitateľnosti je možné vymeniť poradie odosielateľa a príjemcu, ale vtedy je potrebné otočiť aj smer šípky (riadok 19,20).

Fragmenty sa začínajú svojim kľúčovým slovom (riadok 11, 13), za ktorým nasleduje podmienka. Blok je ukončený slovom *end*. Všetko medzi tým sa vygeneruje do vnútra fragmentu. V prípade *alt* sa zvyšné vetvy začínajú slovom *else* s nasledujúcou podmienkou (riadok 17). Fragmenty je možné ľubovoľne vnárať do seba.

Začiatok aktivácie objektu, a teda jeho aktivačný blok je možné vytvoriť pomocou *activate* s názvom objektu (riadok 18). Podobne je možné ho deaktivovať slovom *deactivate*, vytvoriť objekt *create* a zrušiť objekt *destroy*,



Obr. 3.1: Diagram generovaný z ukážky kódu 3.1

Kapitola 4

Súvisiace práce

V kapitole sa pozrieme na existujúce práce, v ktorých sa zaoberali transformáciami prípadu použitia na sekvenčný diagram a opačne. Zameriame sa na dáta na ktorých boli overované výsledky a aké kritéria spĺňali. Ďalším typom sú články popisujúce získavanie dát, a teda diagramov, z ktorých bude dátová sada pozostávať. Zároveň sú pre náš výskum vhodné práce týkajúce sa tvorby dátových sád a štandardov používaných pri ich publikácii.

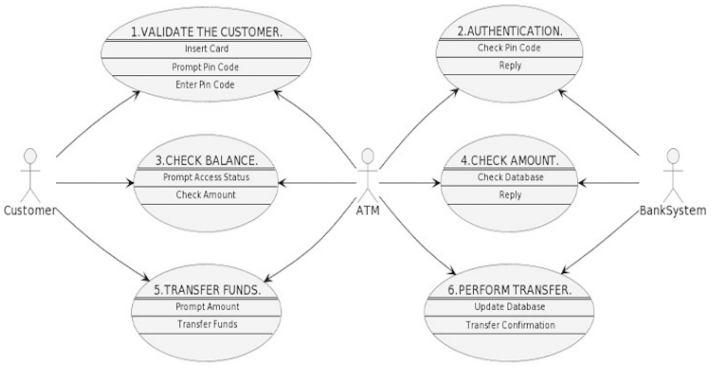
4.1 Seamless transformation from use case to sequence diagrams

V článku autori vytvárajú nástroj, ktorý má pomôcť k väčšej konzistencii návrhu systémov pre študentov a vyučujúcich. Tým, že sú viaceré fázy návrhu prepojené, je možné časti automatizovať, a tak si študenti môžu overiť, či zahrnuli všetky náležitosti z diagramu prípadu použitia do sekvenčného diagramu.

Nástroj dostáva pri vstupe textový diagram prípadu použitia v PlantUML a vygeneruje výstupný sekvenčný diagram, tiež v textovej forme. Súčasťou procesu je aj vytvorenie obrazovej reprezentácie oboch diagramov.

Diagram prípadu použitia v článku je akýmsi hybridom medzi textovým prípadom použitia a diagramom, lebo jednotlivé elementy diagramu obsahujú textový popis usporiadaných krokov ako je na obrázku 4.1. Program na transformáciu prechádza postupne riadok po riadku vstupný súbor, každý analyzuje a extrahuje z neho informácie o aktéroch a akciách, ktoré si následne ukladá. Zaznamenáva si poradie akcii a tiež ich previazanie z aktérmi, pričom je zachované usporiadanie zľava doprava, teda kto vykonáva akciu a komu je smerovaná. Následne zo získaných údajov vygeneruje textový súbor so sekvenčným diagramom v PlantUML tak, že správami, ktoré predstavujú akcie, spája účastných aktérov v zaznamenanom poradí [1].

Výhoda článku spočíva v roku jeho publikácie, a teda jeho aktuálnosť. Avšak hoci



Obr. 4.1: Diagram prípadu použitia z článku

sú v ňom relevantné myšlienky pre našu prácu, je potrebné pristupovať k nemu s rezervou. Autori popisujú triviálne záležitosti ako vývojové prostredie, čo je pre dané riešenie irelevantné. Jednotlivé kroky algoritmu sú popísané povrchne a vstupný súbor musí mať špecifickú formu. Problém transformácie ostáva zachovaný, len sa presunie z vytvorenia sekvenčného diagramu na vytvorenie autormi definovanej formy. Nejedná sa teda o univerzálne riešenie. Taktiež článok neobsahuje dáta o testovaní a uvádza len jeden príklad s prípadom použitia *Výber z bankomatu*.

4.2 Transformation Method from Scenario to Sequence Diagram

Sekvenčný diagram a scenár pri tvorbe softvéru popisujú správanie sa výsledného systému, a tak by mali byť konzistentné. Autori navrhujú metódu transformácie scenára na sekvenčný diagram zapísaný v PlantUML.

Na zapísanie scenára si definovali vlastný jazyk SCEL, ktorý sa podobá na zápis prípadu použitia. Dôvodom je zachovanie vhodnej úrovne abstrakcie krokov, ktorá ak by bola vysoká, je náročné konkretizovať udalosti a jednotlivé akcie v rámci sekvenčného diagramu. Použité podstatné mená a slovesá sú registrované v slovníku a kategorizované podľa konceptu a účelu. Následne sú jednotlivé elementy scenára spárované s elementami sekvenčného diagramu. Akcie sa skladajú zo zdroja, cieľa, objektu a nástroja. Podstatné meno objektu predstavuje správu, zdroj odosielateľa a cieľ príjemcu správy. Ak príjemca nie je špecifikovaný, správu posiela odosielateľ sám sebe. Na základe štruktúry akcie a kľúčových slov v nej, ako napríklad podmienka, sú pridané fragmenty do sekvenčného diagramu. Výsledný diagram v PlantUML je potom možné upraviť používateľom podľa potreby [11].

Článok ponúka myšlienky ako zachovať konzistentnosť medzi prirodzenou rečou a sekvenčným diagramom. V našej práci to môže poslúžiť na overenie, nakoľko sú zachované znaky prípadu použitia v sekvenčnom diagrame a naopak. Výhodou je aj

využitie PlantUML, ktorý používame v našej práci, a teda sledované znaky môžu byť podobné. Síce metóda vyžaduje špecifický formát scenára, ale ten sa do značnej miery podobá s definíciou Cockburna pre dobrý prípad použitia [3].

4.3 An Extensive Dataset of UML Models in GitHub

Cieľom tohto článku bolo vytvoriť dátovú sadu modelov spolu s metadátami o projektoch v ktorých boli použité. Zozbierali viac ako 93 000 UML súborov z vyše 24 000 repozitárov z GitHub. Súbory získavali polo-automatizovanou metódou a sú rôznych typov ako obrázky, xmi súbory alebo plantuml.

Prvým krokom, bolo zozbierať množstvo súborov z GitHub. Zoznam repozitárov, ktoré neboli duplicitné, bol z GHTorrent. Následne v každom z nich identifikovali potenciálne súbory, ktoré mohli obsahovať UML diagram, či už podľa kľúčového slova v názve alebo podľa prípony. V treťom kroku overovali, či skutočne obsahujú UML. Súbory z príponou .uml boli automaticky zaradené do dátovej sady. Súbory s príponou .xml a .xmi kontrolovali na základe použitej referenčnej schémy, či reprezentovala UML. Zvyšné súbory boli väčšinou obrázky a tie, po odstránení ikon a duplicít, určovali ručne. Nakoniec na základe webovej linky repozitára doplnili metadáta o projekte v ktorom boli použité a o autorovi [17].

V čase písania práce sme sa k publikovanej dátovej sade nedopracovali, ale tabuľka s repozitármi a počtom UML súborov v nich bola ešte dostupná. Tú môžme využiť v našej práci, a tak už pracovať s repozitármi, v ktorých je overený UML súbor.

4.4 Data Cards: Purposeful and Transparent Dataset Documentation for Responsible AI

S pribúdajúcim množstvom modelov AI je potrebné zabezpečiť ich bezpečnosť, spoľahlivosť a transparentnosť. To sú niektoré z vlastností zodpovednej AI, ktorá vzniká po dodržaní štandardizovaných princípov. To zahŕňa dobrú dokumentáciu dátovej sady použitej na trénovanie, kde je zahrnutý pôvod dát, spôsob získavania atď. V článku autori predstavujú koncept dátových kariet na podporu transparentnej, účelnej a človeku čitateľnej dokumentácie dátových sád.

Autori spolupracovali s tímami zameranými na prácu s dátami a vývoj AI, na identifikovaní kľúčových poznatkov o dátovej sade pre jej použitie. Vytvorili framework za pomoci Google dokumentov, aby členovia tímu mohli pracovať na dokumentácii paralelne a bola vždy aktuálna. Dátové karty sú postavené na štyroch princípoch: flexibilita, modulárnosť, rozširovateľnosť, prístupnosť. Framework je použiteľný na širokú škálu dátových sád rôznych typov a zameraní. Organizuje dokumentáciu do zmysluplných

sekcií, ktoré sú samostatné a dobre štruktúrované jednotky, schopné poskytnúť kompletný popis jednej oblasti dátovej sady. Komponenty sú rozšíriteľné a prispôsobiteľné pre špecifické potreby. Obsah je predstavený v rôznych úrovniach detailu, a tak je prispôsobený efektívnejšiemu vyhľadávaniu a čitateľnosti [16].

Článok poskytuje body, ktoré by mala dátová sada v metadátach obsahovať, a na aké oblasti sa zamerať pri tvorbe dokumentácie k dátovej sade. Tieto bodu môžu poslúžiť na inšpiráciu v našej práci.

Kapitola 5

Návrh

Kapitola 6

Výskum

Zber dát:

- **Repozitáre na GitHube -** GitHub slúži ako jedna z hlavných platforiem na ukladanie súborov pre developerov, a tak sa vo verejných repozitároch nachádza nespočetne prípadov použitia a sekvenčných diagramov. Navyše GitHub poskytuje API, cez ktorú je možné prehľadávať súbory a repozitáre.
- **Generovanie pomocou LLM (OpenAI - ChatGPT) -** Vzhľadom na to, že obidva formáty (prípadu použitia a sekvenčného diagramu) sú textové, je možnosť použiť generatívnu umelú inteligenciu, a tak simulovať vytváranie človekom.
- **Extrakcia dát z rôznych úložísk na webe -** Okrem GitHubu existujú rôzne úložiská na webe, ktoré sú špecifické pre konkrétnu potrebu. Dáta z nich môžme využiť na doplnenie dátovej sady. Napríklad sme našli úložisko prípadov použitia z praxe, ale sú v rôznych formátoch.
- **Web crawling? -** Na internete v rámci webových stránok sú zverejnené príklady prípadov použitia aj sekvenčných diagramov. Minimálne vzdelávacie inštitúcie s odborom softvérové inžinierstvo ich budú mať zverejnené v rámci učebných materiálov. Pomocou web crawlingu vieme takéto webové stránky prehľadávať a získavať nami hľadané dáta.

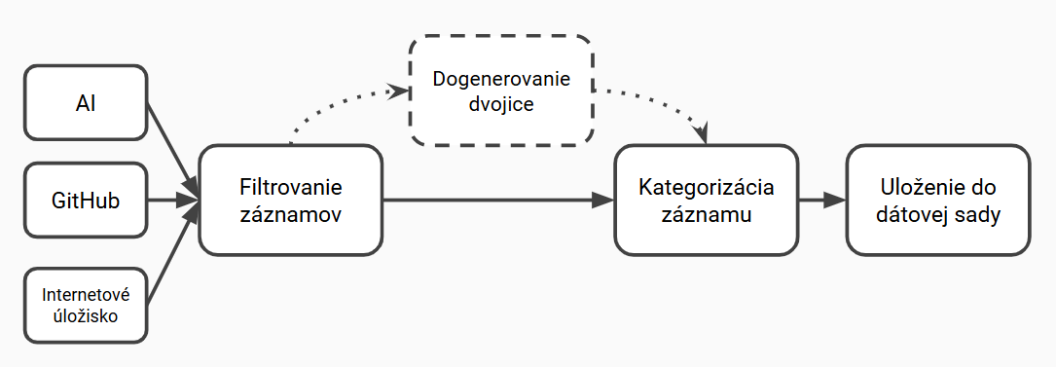
Prípadné vygenerovanie druhej súčasti pomocou AI ak sa získa len jeden z dvojice.

6.1 Identifikácia PlantUML sekvenčného diagramu

Nájdenie súborov s príponou .uml a opsahujú kľučové slovo participant.

Ukážka kódu 6.1: Hľadanie súborov so sekvenčným diagramom

```
1 gh api search/code?q=participant+extension:uml \\
2     --jq ".items[].html_url" --paginate
```



Obr. 6.1: Postupnosť krokov získavania dát

Táto query 6.1 môže nájsť aj xmi súbory.

Ukážka kódu 6.2: Hľadanie výhradne plantuml súborov so sekvenčným diagramom

```
1 gh api search/code?q=participant+@startuml+extension:uml \\
2     --jq ".items[].html_url" --paginate
```

Avšak plantUML môže byť uložené aj pod príponou .puml, respektíve textové .txt, .doc, .docx.

Kľúčové slová, ktoré v plantuml obsahuje len sekvenčný diagram:

- participant
- alt, opt, loop, par, group
- activate, deactivate
- destroy

Môžu byť použité zároveň na kategorizáciu.

Základný diagram pre PlantUML je sekvenčný diagram, čiže aj najzákladnejší bez kľúčových slov je tiež sekvenčný.

Ukážka kódu 6.3: Jednoduchý sekvenčný diagram

```
1 @startuml
2 Bob -> Alice : Authentication Request
3 Bob <- Alice : Authentication Response
4 @enduml
```

Jediný rozdiel oproti diagramu tried by bol v tom, že ten by obsahoval **class Bob**, takže sekvenčný diagram nemôže osahovať slovo class.

6.2 Identifikácia sekvenčného diagramu v .xmi

Všetky obsahujú kľúčové slovo **Interaction**, ale často krát je v jednom súbore množstvo diagramov. Je to spôsobené automatickým vytváraním zo zdrojového kódu. Pre Eclipse je sekvenčný diagram v elemente

<packagedElement xmi:type="uml:Interaction"

Treba určiť level abstrakcie, lebo príliš detailné sekvenčné diagramy nie sú vhodné na vytváranie prípadov použitia.

6.3 Získanie súborov z GitHubu

V url je potrebné vymeniť *https://github.com* za *https://raw.githubusercontent.com* a zmazať */blob*. Následná linka je na raw súbor, ktorý je jednoducho stiahnuteľný cez CURL.

Script 6.4 má za úlohy z daných repozitárov zo súboru *repos.txt* nájsť všetky súbory s príponou .uml a vrátiť URL na raw súbor do súboru *output.txt*.

Ukážka kódu 6.4: URL na raw súbory UML z daných repozitárov

```
1  #!/bin/bash
2
3  while IFS= read -r line; do
4
5      repo=$(echo "$line" | sed 's/[[:space:]]*$//')
6      gh api search/code?q=repo:$repo+extension:uml \
7      --jq ".items[].html_url" | \
8      sed 's|https://github.com␣\
9  ␣␣␣␣|https://raw.githubusercontent.com|' \
10     | sed 's|/blob||' >> output.txt
11
12 done < repos.txt
```

Kapitola 7

Diskusia

Záver

Cieľom práce bolo ...

Literatúra

[1] Abdulrahman Alyami, Salvatore Flavio Pileggi, Osama Sohaib, and Igor Hawryszkiewycz. Seamless transformation from use case to sequence diagrams. *PeerJ. Computer science*, 9, June 2023.

[2] Inc. Amazon Web Services. What are large language models?, 2024. [Citované 2024-11-07] Dostupné z `https://aws.amazon.com/what-is/large-language-model/`.

[3] Alistair Cockburn. *Writing Effective Use Cases*. Addison-Wesley, 2001.

[4] IBM Data and AI Team. Ai vs. machine learning vs. deep learning vs. neural networks: What's the difference?, 2023. [Citované 2024-11-07] Dostupné z `https://www.ibm.com/think/topics/ai-vs-machine-learning-vs-deep-learning-vs-neural-networks`.

[5] Martin Fowler. User story, 2013. [Citované 2024-12-11] Dostupné z `https://martinfowler.com/bliki/UserStory.html`.

[6] Google. Gemini apps faq. [Citované 2024-10-18] Dostupné z `https://gemini.google.com/faq`.

[7] Google. Gemini ultra, 2024. [Citované 2024-10-18] Dostupné z `https://deepmind.google/technologies/gemini/ultra/`.

[8] Object Management Group. Unified modeling language version 2.5.1, 2017. [Citované 0000-00-00] Dostupné z `https://www.omg.org/spec/UML/2.5.1/PDF`.

[9] Ivar Jacobson. Object-oriented development in an industrial environment. *SIGPLAN Not.*, 22(12):183–191, December 1987.

[10] Microsoft. Your everyday ai companion | microsoft copilot, 2024. [Citované 2024-10-18] Dostupné z `https://www.microsoft.com/en-us/microsoft-copilot/learn`.

[11] Yousuke Morikawa, Takayuki Omori, and Atsushi Ohnishi. Transformation method from scenario to sequence diagram. In *International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management*, 2018.

[12] OpenAI. Introducing chatgpt, 2022. [Citované 2024-10-18] Dostupné z `https://openai.com/index/chatgpt`.

[13] Sundar Pichai and Demis Hassabis. Introducing gemini: our largest and most capable ai model. [Citované 2024-10-18] Dostupné z `https://blog.google/technology/ai/google-gemini-ai/#introducing-gemini`.

[14] PlantUML. Drawing uml with plantuml, 2024. [Citované 2024-11-22] Dostupné z `https://plantuml.com/guide`.

[15] PlantUML. Frequently asked questions (faq), 2024. [Citované 2024-12-08] Dostupné z `https://plantuml.com/en/faq`.

[16] Mahima Pushkarna, Andrew Zaldivar, and Oddur Kjartansson. Data cards: Purposeful and transparent dataset documentation for responsible ai. In *Proceedings of the 2022 ACM Conference on Fairness, Accountability, and Transparency*, FAccT '22, page 1776–1826, New York, NY, USA, 2022. Association for Computing Machinery.

[17] Gregorio Robles, Truong Ho-Quang, Regina Hebig, Michel R. V. Chaudron, and Miguel Angel Fernández. An extensive dataset of uml models in github. *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, pages 519–522, 2017.

[18] SAP. Čo je to umelá inteligencia? [Citované 2024-11-07] Dostupné z `https://www.sap.com/sk/products/artificial-intelligence/what-is-artificial-intelligence.html`.