Analýza a návrh prototypu nástroje pro odhadování rozsahu software

Filip Frýdl

Bakalářská práce
2024

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav informatiky a umělé inteligence

Akademický rok: 2023/2024

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Filip Frýdl**
Osobní číslo:	**A21325**
Studijní program:	**B0613A140020 Softwarové inženýrství**
Forma studia:	**Prezenční**
Téma práce:	**Analýza a návrh prototypu nástroje pro odhadování rozsahu software**
Téma práce anglicky:	**Analysis and Design of a Prototype Software Size Estimation Tool**

Zásady pro vypracování

1. Proveďte rešerši možných přístupů k odhadování rozsahu software.
2. Pro zvolenou metodu analyzujte možnosti výpočtového nástroje.
3. Navrhněte funkcionalitu výpočetního nástroje.
4. Proveďte analýzu a implementaci prototypu aplikace vzorových příkladů.
5. Uveďte možný další rozvoj aplikace.

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam doporučené literatury:

1. TRENDOWITZ, Adam a Ross JEFFERY. Software Project Effort Estimation. Cham: Springer International Publishing, 2014. ISBN 9783319036281.
2. SILHAVY, Radek; SILHAVY, Petr; PROKOPOVA, Zdenka. Analysis and selection of a regression model for the use case points method using a stepwise approach. Journal of Systems and Software, 2017, 125: 1-14.
3. BUNDSCHUH, M. AND C. DEKKERS The IT measurement compendium : estimating and benchmarking success with functional size measurement. Edtion ed. Berlin: Springer, 2008. xxxv, 643 p. p. ISBN 9783540681878
4. ARLOW, Jim and NEUSTADT, Ila, 2001. UML and the Unified Process: Practical object-oriented analysis and design. . Boston, MA: Addison Wesley. ISBN 9780201770605.
5. PILONE, Dan, UML 2.0 pocket reference. Beijing ; Farnham : O'Reilly, c2006. ISBN 0596102089.
6. Unhelkar, Bhuvan, Software engineering with UML. Boca Raton : CRC Press : Auerbach Publications, c2006. ISBN 9781351235167.

Vedoucí bakalářské práce: **doc. Ing. Radek Šilhavý, Ph.D.**
Ústav počítačových a komunikačních systémů

Datum zadání bakalářské práce: **5. listopadu 2023**
Termín odevzdání bakalářské práce: **13. května 2024**

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky
DĚKAN
L.S.

doc. Ing. Jiří Vojtěšek, Ph.D. v.r.
děkan

prof. Mgr. Roman Jašek, Ph.D., DBA v.r.
ředitel ústavu

Ve Zlíně dne 5. ledna 2024

Prohlašuji, že

- beru na vědomí, že odevzdáním bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně;
- byl/a jsem seznámen/a s tím, že na moji bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – bakalářskou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na bakalářské práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne 13.5.2024 Filip Frýdl, v.r.

ABSTRAKT

Cílem bakalářské práce je navrhnout a implementovat prototyp nástroje pro odhad rozsahu software, který bude schopný vypočítat odhad pro vybranou metodu use case points. Součástí práce bude analýza požadavků, funkční a datová analýza s vytvořením různých modelů pomocí grafického jazyka UML. Dále bude vytvořen prototyp webové aplikace, který bude vytvořen podle návrhu. Posledním bodem bude zamyšlení nad budoucím rozvojem aplikace.

Klíčová slova: návrh prototypu, odhad rozsahu software, implementace prototypu, metoda use case points, Flask aplikace

ABSTRACT

The aim of bachelor thesis is to design and implement prototype of software size estimate tool, which will be able to calculate an estimate for the selected use case points method. The thesis will include requirements analysis, functional and data analysis with the creation of different models using the UML graphics language. Furthermore, a prototype of web application will be created according to the design. The last point will be a reflection on the future development of the application.

Keywords: prototype design, software size estimate, prototype implementation, use case points method, Flask application

Rád bych poděkoval vedoucímu práce panu doc. Ing. Radku Šilhavému Ph.D. za poskytnutí cenných rad při psaní bakalářské práce.

Prohlašuji, že odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.

OBSAH

ÚVOD

V době stále rychlejšího technologického rozvoje a neustálého zvyšování poptávky po software se stává odhad rozsahu software stále více klíčovým a nezbytným při vývoji. Tato bakalářská práce se zabývá analýzou, návrhem a implementací prototypu aplikace pro odhad rozsahu software.

Cílem teoretické části je představit různé metody a přístupy, které se zabývají problematikou odhadu software. Zaměří se na v dnešní době používané praktiky v praxi a jejich teoretické základy, které tyto metody podporují. Důraz bude kladen na pochopení komplexitě řešení jednotlivých problémů a vysvětlení klíčových faktorů ovlivňujících přesnost a spolehlivost odhadů. Další částí bude popis nástrojů pro vývoj aplikace pro odhad. Rozbor použitých nástrojů se zaměří na technologie použité při vývoji prototypu. Technologií, která bude použita na vývoj, je programovací jazyk Python, kde se bude využívat několik zásadních knihoven pro zhotovení prototypu. Pro analýzu a návrh bude použit grafický jazyk UML, který je běžně používaný ve fázi analýzy a návrhu.

Praktická část bakalářské práce bude zaměřena na analýzu a návrh aplikace, který se bude zabývat slovním popisem návrhu aplikace. Dále bude obsahovat analýzu požadavků aplikace s následným vytvořením modelu případů užití a dalšími důležitými prvky analýzy. Obsahem návrhu bude ER diagram s modelem tříd a sekvenční diagramy pro nejdůležitější části prototypu aplikace. Dále bude následovat samotná implementace navrhnuté aplikace. Bude se jednat o webovou aplikaci vytvořenou pomocí mikro frameworku Flask, jenž je vhodným nástrojem pro vývoj webových aplikací za použití programovacího jazyka Python. Poslední částí praktické části bude zamyšlení nad budoucím vývojem aplikace.

I. TEORETICKÁ ČÁST

1 ROZBOR PROBLEMATIKY

Odhadování software je proces stanovení množství práce, která je nutná k dokončení softwarového projektu. Je to důležitá součást plánování projektu, protože umožňuje týmu určit kolik času, zdrojů a peněz bude potřeba. Smyslem je tedy vytvoření časového plánu. Prvotní odhady jsou vytvářeny v inicializační fázi projektu.

1.1 Definice některých pojmů

Na začátek je potřeba si stanovit některé pojmy, které souvisí s problémem odhadu software. Nejčastější pojmy, se kterými se můžeme potkat, jsou:

Software = je v informatice pojem představující všechny programy v počítačích, které vykonávají nějakou činnost.

Odhad = proces, u kterého předpovídáme kolik času, úsilí atp. bude potřeba pro vývoj software

Člověkohodina = hodnota odpovídající času práce průměrného člověka za jednu hodinu

Pracnost = množství člověkohodin, které je potřeba pro vykonání určitého úkolu či projektu

Rozsah = velikost v ucp bodech

2 METODY ODHADU SOFTWARE

Tato kapitola je zaměřena na představení různých metod pro odhad software. Budou zde představeny přístupy odhadu spolu s nejpoužívanějšími metodami.

2.1 Expertní odhad

Metoda odhadování, která je založena na zkušenosti experta. Kvalita odhadu je přímo úměrná kvalitě experta. Lepší odhad bude v případě, pokud se na odhadu podílí více expertů. [3]

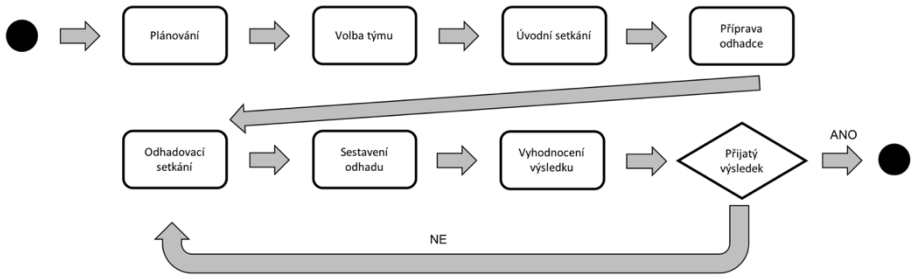
2.1.1 Metoda Delphi

Je založena na získání odhadů více expertů. Jsou vytvořeny týmy složené jak z projektových manažerů, tak i z expertů v oblastech týkajících se daného projektu (u vývoje softwaru jsou to například programátoři, analytici atp.). Jedná se o častou metodu odhadů pracností a používá se především u projektů, kde nelze předložit měřitelná data. Nejdříve je skupině expertů popsán cíl projektu. Ti následně představí svůj odhad. Výsledek je anonymně zpracován a zpětně zaslán odborníkům. Postup je iteračně opakován, až výsledky odhadů konvergují k jednomu odhadu. Nevýhodou metody může být zaujatost jednotlivých expertů podílejících se na odhadu. Tento problém je možné částečně vyřešit zvýšením počtu odhadujících expertů [1, 3].

V této metodě se využívá 7 základních kroků [2]. Tyto kroky jsou:

- Plánování
- Volba týmu
- Úvodní setkání
- Příprava odhadce
- Odhadovací setkání
- Sestavení odhadu
- Vyhodnocení výsledku

Iterační znázornění těchto kroků je na obrázku (Obr. 1)



Obrázek 1 – Ukázka Delphi metody [3]

2.1.2 Metoda tří bodů

Tato metoda je příkladem spojení technik založených na zkušenostech a modelech. Při použití metody tří bodů se vyžaduje využití tří odhadů na každou úlohu. Poté se na základě statistiky určí, kolik úsilí bude potřeba s danou pravděpodobností k úspěchu v plánu testů. Mezi tři zmíněné odhady patří:

- Nejhorší odhad – odhad, který počítá s nejhoršími podmínkami a neplánovanými obtížemi
- Nejlepší odhad – odhad, který počítá s ideálním průběhem a nezahrnuje neplánované obtíže
- Realistický odhad – odhad, který pracuje s obvyklým množstvím neplánovaných obtíží

Při použití této metody se nadále pracuje se čtvrtinou nejhoršího odhadu, čtvrtinou nejlepšího odhadu a polovinou realistického odhadu.

V praxi se můžeme potkat i s jednodušší variantou, která pracuje pouze se dvěma body. Práce s touto variantou vyžaduje použití pouze dvou odhadů, a to nejhoršího a nejlepšího odhadu. Ty se následně zprůměrují a vznikne výsledek odhadu. [3]

2.2 Odhad pomocí analogie

Je metoda založená na získaných vědomostech z projektů, které již existují a obsahují prvky podobného charakteru. Předpokládá se tedy, že odhad software bude podobný. Jedná se o použití zkušeností a při použití se identifikují rozdíly mezi novým a již řešeným projektem. Důležitou součástí metody je správná analýza předchozího projektu a důkladné zjištění, zda se jedná o podobné řešení problému. V případě špatného zpodobnění projektů by analogická

metoda mohla negativně ovlivnit celý odhad aktuálního projektu. A proto se tato metoda používá spíše jako doplňková, například jako externí odhad. [3]

2.3 Metody založené na modelu

Odhady založené na modelu používají data z již uskutečněných a zrealizovaných projektů. Opírají se o matematické a statistické modely, které analyzují různé aspekty softwarových projektů. Tyto modely se vytvářejí na základě dat a znalostí o daném problému. Využitím těchto modelů můžeme odhadnout náklady, čas a zdroje potřebné pro vývoj. [3]

2.3.1 Metoda constuctive cost model - COCOMO

Jedná se o jeden z nejznámějších a nejpopulárnějších modelů pro odhad nákladů. První varianta modelu této metody se skládá z hierarchie tří modelů s rostoucím významem detailů – základní model COCOMO, střední model COCOMO a pokročilý mode COCOMO. Důvodem vyvinutí těchto modelů bylo odhadování projektů zákaznického software a software stavěného na míru. Při výpočtu odhadu je potřeba znát rozsah softwaru, tedy odhad řádků zdrojového kódu. Tato varianta poskytuje bodové odhady pracnosti a časový plán. Vhodné použití této varianty je v raných fázích projektu, kdy je pouze hrubý odhad velikosti a složitosti software. [3,4]

Novější druhá verze metody COCOMO byla vytvořena z důvodu neschopnosti první verze odhadovat objektově-orientovaný software, evoluční modely a software pro komerční účely na sklad. Druhá možnost nabízí alternativní způsob výpočtu pracnosti projektu, a to s ohledem na jednotlivé fáze projektu. Výsledkem je pravděpodobnostní rozsah odhadů, který se pohybuje v rozmezí jedné standardní odchylky k nejpravděpodobnějšímu odhadu. Varianta je vhodná pro použití v pozdějších fázích projektu, kdy jsou známy podrobnější informace o požadavcích a architektuře softwaru. [3,4]

Výpočet metody COCOMO využívá několik vzorců při řešení odhadu. První vzorec (1) slouží pro výpočet pracnosti v člověkohodinách (MM). Druhý (2) vzorec se využívá pro výpočet doby v měsících (D). A třetí (3) vzorec slouží pro výpočet doporučeného počtu vývojářů (N). [3,4]

$$\boldsymbol{MM = a * (KLOC) * b} \quad (1)$$

$$D = c * (MM) * d \quad (2)$$

$$E = MM/D \quad (3)$$

Kde KLOC je ekvivalent počtu řádků zdrojového kódu a písmena a, b, d jsou volena podle typu projektu, jak je ukázáno v tabulce (Tab. 1). Písmeno c má hodnotu 2,5. [3,4]

Tabulka 1 – Hodnoty pro výpočet metody COCOMO

Typ projektu	a	b	d
Organický	2,4	1,05	0,38
Přechodový	3,0	1,12	0,35
Uzavřený	3,6	1,20	0,32

2.3.2 Functional points (FP)

Metoda funkčních bodů (dále jen FP) je založena na odhadu složitosti interakce s okolím. Můžeme ji uplatnit v momentě, kdy jsou již specifikovány transakce v budovaném systému a je jasná datová struktura, tedy na konci detailního návrhu. FP se dělí na dva kroky, přičemž u prvního kroku se jedná o specifikaci složitosti systému a v druhém kroku je vykonán převod tzv. funkčních bodů definujících složitost systému na člověkodny pracnosti. Odhad se v této metodě provádí na úrovni celého projektu. Dále je třeba odhadnutou hodnotu celkové pracnosti rozdělit mezi jednotlivé etapy, fáze a činnosti. Při práci s FP se můžeme setkat s určitými typy funkcí, které se hodnotí v prvním kroku metody. Mezi typy patří:

- Externí vstupy (EI) – požadavky, které definují počet externích vstupů. Pro představu se mezi externí vstupy řadí funkce, proces a transakce, které umožňují uživateli aplikace provádět změny ve vnitřní struktuře dat, a to přidáním, odebráním nebo editací
- Externí výstupy (EO) – definuje počet externích výstupů. Pro představu sem patří všechna unikátní uživatelská nebo řídící data, která opouští hranice aplikace.
- Externí dotazy (EQ) – definuje počet externích dotazů. Lze to chápat jako externí vstupy i výstupy (spíše externí výstupy), které jsou bez vlastního zpracování dat.

- Vnitřní logické soubory (ILF) – definují počet vnitřních logických souborů. Jedná se o soubory dat, které musí být generovány, používány a udržovány právě vyvíjeným systémem. V podstatě se sem řadí pracovní datové soubory, na kterých bude systém postavený a ze kterých bude čerpat data.
- Soubory vnějších rozhraní (EIF) – definuje počet souborů vnějších rozhraní. Jedná se o vnitřní logické soubory v jiné aplikaci. Patří sem různé „odkazy“ na rozhraní, odkud čerpáme data pro náš systém.

FP primárně sloužila právě k měření složitosti systému. Vychází se z toho, že čím je systém složitější, tím je také pracnější. Z toho se odvozuje i jeho pracnost. Metoda je poměrně objektivní. [4,6]

2.3.3 Use Case Points (UCP)

Oproti výše zmíněným metodám je UCP výhodná k použití v prvních fázích vývoje. Jejím autorem je Gustav Karner, a proto se metoda někdy označuje jako Karnerova metoda. Základní metoda UCP je založena na přiřazení vah jednotlivým aktérům a případům užití, které se obvykle používají jako funkční popisy navrhovaných systémů nebo software. Využívají se tři typy shluků [5]. Prvním krokem při použití této metody je zjištění počtu aktérů a případů užití. Aktéři se rozdělí do skupin, které mají rozdílnou váhu. Tyto skupiny jsou:

- Jednoduché – např. využití API, aktérem je tedy jiný systém
- Průměrné – např. systém komunikující přes protokol TCP/IP
- Komplexní – např. uživatelé využívají GUI, webové stránky apod.

V dalším kroku se rozdělí případy užití do kategorií dle významů. Stejně jako aktéři jsou případy užití tři typy, a to jednoduchý, průměrný, komplexní. Příkladem dělení může být počet kroků ve scénáři, a to následovně: <=3, 4-7, >= 8, kde váhy jsou 5, 10, 15. Nebo lze kroky nahradit transakcemi. Celkový počet UCP je tedy roven součtu kroku jedna a dva. Ve třetím kroku se řeší technické faktory a faktory prostředí, kde dochází k výpočtu těchto faktorů. Důležitou součástí výpočtu technického faktoru je ohodnocení váhy vybraným technickým faktorům, které souvisejí s technickou stránkou vytvářeného software. Hodnotící stupnice zahrnuje hodnoty 0 až 5. Pokud bude faktoru přidělena 0, pak je jeho vliv bezvýznamný, v případě 5 je velmi významný. Hodnoty přidělené každému faktoru se roznásobí přidělenou váhou a dále se provede jejich součet [5]. Tím vznikne tzv. technický faktor

(TFaktor), který se nadále dosadí do vzorce (4) pro výpočet technického faktoru složitosti (TCF). [5, 7]

$$TCF = 0,6 + (0,01 * TFaktor) \quad (4)$$

Stejný postup se použije i u faktoru prostředí, kde se také přiřadí váhy vybraným faktorům. Tentokrát nám vznikne tzv. faktor prostředí (EFaktor), který se použije ve vzorci (5) pro výpočet složitosti faktoru prostředí (EF). [5]

$$EF = 1,4 + (-0,03 * EFaktor) \quad (5)$$

Výpočet pomocí těchto dvou vzorců se využije v dalším kroku, a to pro výpočet upraveného počtu UCP. Ten zjistíme tak, že se vynásobí celkový počet UCP s hodnotami TCF a EF. Závěrečným krokem v metodě UCP je výpočet člověkohodin. Ten se vykoná vynásobením celkové hodnoty UCP s hodnotou průměrné produktivity. [5]

3 POUŽITÉ TECHNOLOGIE

Při vývoji software je důležitou součástí návrh, který se dále zpracovává a vzniká z něj hotová aplikace. Na tvorbu takové aplikace existují různé technologie, které jsou stále vyvíjeny a modernizovány. Pro samotný návrh se používá jazyk UML, který je využit i v této práci. Pro implementaci návrhu je možno využít mnoho technologií a programovacích jazyků. Pro tuhle práci je využit programovací jazyk Python, jenž nabízí možnost vyvinutí aplikací pro web, počítač a další.

3.1 Jazyk UML

Jazyk UML (Unified Modeling Language) vzniká v 90. letech minulého století jako sjednocení několika různých standardů, a to z důvodu rozšíření objektově orientovaného programování. Dalším důvodem vzniku bylo vyvinutí takového nástroje, který by dokázal vizuálně modelovat všechny fáze vývoje. Jak již bylo zmíněno, UML se využívá na návrh a analýzu software. Vzhledem k tomu, že se jedná o grafický jazyk [8], je zcela nezávislý na dalším programovacím jazyku, tedy na další vývoj podle návrhu může být použit libovolný jazyk. Základní myšlenkou UML je pochopení systému jako souboru, kde jsou vzájemně propojené objekty. Objekt zde reprezentuje jednotlivé entity v systému jako je například auto nebo zákazník. Každý objekt má určité vlastnosti (např. značka auta nebo jméno zákazníka) a operace (např. řízení auta nebo přihlášení zákazníka). [8,9]

3.2 Python

Python je programovací jazyk poprvé zveřejněn roku 1991. Jedná se o vysokoúrovňový, univerzální jazyk, který se používá v různých oblastech vývoje softwaru. Patří mezi interpretované jazyky, což znamená, že se daný zdrojový kód překládá do sady instrukcí. Ta se jinak nazývá bajtkód, který je detailem jazyka CPython, což byl původní implementační jazyk Pythonu. Sada instrukcí je pro procesor srozumitelná a ten je schopný ji vykonat [10]. Mezi jeho hlavní vlastnosti patří jednoduchost. Syntaxe programovacího jazyka Python je oproti jiným programovacím jazykům poměrně jednoduchá a pochopitelná. Na to navazuje čitelnost kódu. Kód je obvykle čitelný a pochopitelný, a to i pro ty, kteří s programováním teprve začínají. Jedná se o tzv. open-source jazyk, tedy volně přístupný, který je díky svým nástrojům a knihovnám vhodný pro vývoj mnoha odvětví, jako jsou webové stránky, umělé inteligence nebo třeba automatizace různých úloh na počítači uživatelů. Velice oblíbenou vlastností tohoto programovacího jazyka je efektivnost. A to díky tomu, že neexistuje krok

kompilace, tedy převod zdrojového kódu na strojový kód. Tím je umožněn rychlejší cyklus editace, testování a ladění, které je v python programech velmi snadné. Oproti jiným jazykům se v momentě nastání chyby při ladění nebo vložení špatného vstupu zavolá výjimka, což znamená, že program pouze upozorní na chybu a nezhroutí se. V případě nezachycení výjimky se vytiskne obsah zásobníku, který nastane před nastáním chyby. Tyto vlastnosti umožňují rychlý a efektivní vývoj. Python je velmi oblíbený i mezi nejznámějšími firmami, jako jsou Netflix, Google nebo NASA. [11]

3.2.1 Flask

V programovacím jazyku Python existuje více způsobů, jak vytvořit webové stránky. Jedním z nich je použití backhandového microframeworku Flask. Označení microframework si získal tím, že nevyžaduje specifické knihovny ani nástroje. Je založený na knihovnách Werkzeug, Jinja a Click. Tyto knihovny nabízí základní části, které jsou potřebné pro tvorbu webových stránek. Flask je navržen tak, aby byl co nejjednodušší. Je tedy vhodný k vývoji jednodušších webových stránek, ale může být použit i na vývoj komplexnějších webů. Jeho velkou výhodou je rozšiřitelnost. V základu sice neobsahuje knihovny, které nabízejí jiné frameworky, ale existuje zde možnost přidání právě chybějících knihoven třetích stran. [12]

II. PRAKTICKÁ ČÁST

4 ANALÝZA A NÁVRH APLIKACE

Tato kapitola je zaměřena na analýzu a návrh aplikace pro odhad rozsahu software. Velká část návrhu bude probíhat v grafickém jazyku UML, který je vhodný k přehledné realizaci a ukázce analyzovaného software.

4.1 Možnosti výpočetního nástroje

V praxi existuje několik nástrojů, které je vhodné použít na výpočetní operace nebo práci s daty. Jsou to nástroje, které se hojně využívají pro každodenní práci. Takovými nástroji mohou být například Microsoft Excel nebo Sparx Enterprise Architect.

4.1.1 Microsoft Excel

Jedná se o aplikaci sloužící pro přehlednou práci s daty. Nabízí spoustu možností, jak data zpracovat a znázornit, například do tabulky nebo grafu. V souvislosti s odhadem software se dá říci, že je to nástroj, kde je možnost vypočítat odhad. Nevýhodou použití takového nástroje je vložení přesných číselných hodnot. To znamená, že se musí vložit do výpočtu přesný počet všech prvků a není zde možnost vložení obecných informací, ze kterých by si aplikace sama vzala potřebné vstupy. Je tedy nutnost vše psát ručně a pro každý takový odhad vytvořit novou tabulku. Proto tato aplikace není úplně vhodná na použití při vykonání odhadu, ale mohla by být vhodná pro jiné účely. To znamená, že aplikace sloužící na odhad by neměla sloužit jako zápis dat do tabulek nebo počítání s přesně danými čísly, ale měla by zvládnout analyzovat informace ze vstupů, přehledněji uchovávat historii odhadů a bezpečně ukládat data, aby byla přístupná právě danému uživateli.

4.1.2 Sparx Enterprise Architect

Jedná se o aplikaci, která je hojně využívaná při modelování a designu software. Je to aplikace, ve které je možnost vypracování různých diagramů, které jsou potřebné pro lepší orientaci ve vyvíjeném software. Je velice oblíbená mezi architekty software, protože podporuje několik různých standardů, mezi které patří UML standard, ve kterém bude probíhat analýza a návrh prototypu. Co se týče odhadu, hraje tento software velkou roli. Sice nedokáže vykonat odhad sám, ale je schopný vizualizovat všechna data tak, že vznikne přehledný model, jenž se dá převést do vstupních hodnot do odhadu. Další výhodou je uložení jednotlivých diagramů či modelů do obrázku, který by mohl sloužit jako vstup do odhadu. Nevýhodou této aplikace je její cena a složitost. Jedná se totiž o placenou aplikaci, která nabízí

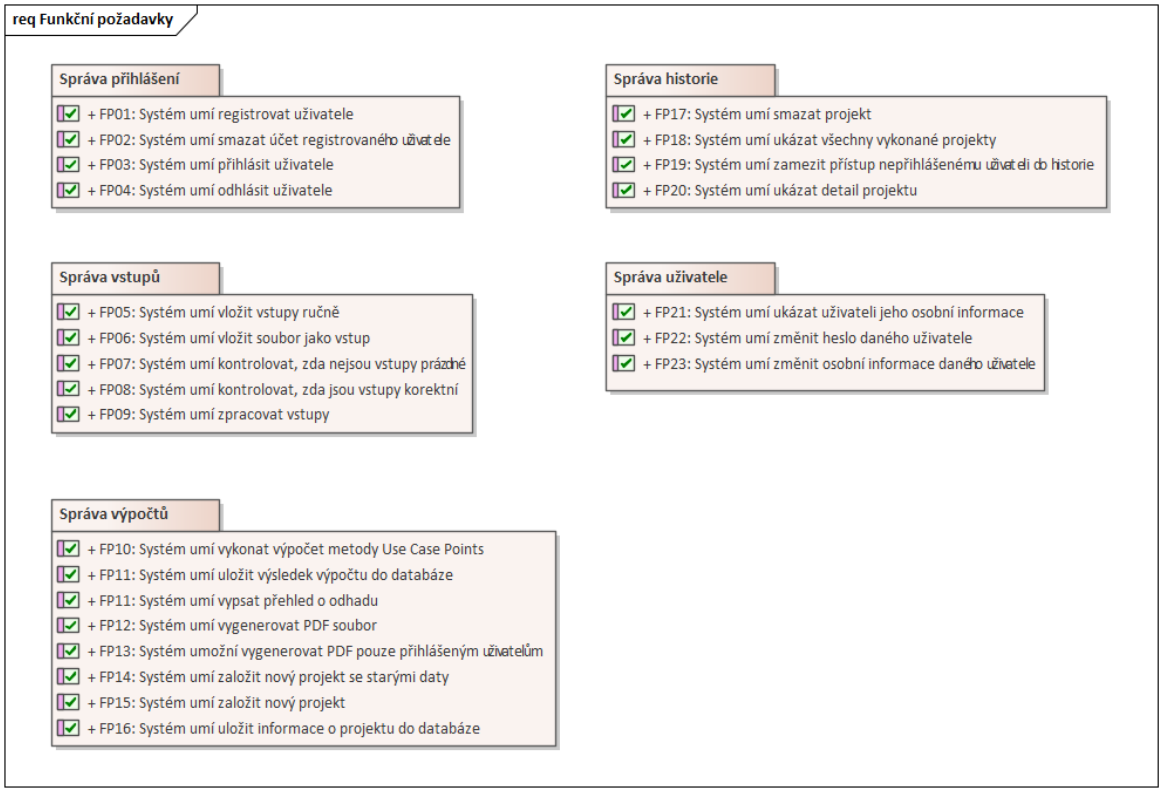
spoustu možností, jak něco popsat nebo vizualizovat, ale je potřeba projít školením, jenž stojí další peníze. Celkové použití při odhadu je velmi užitečné a skoro nezbytné, ovšem pro samotný odhad je potřeba jiného software, který by byl dalším krokem po vytvoření všech diagramů a modelů.

4.2 Požadavky

Důležitou součástí každého návrhu je návrh detailních požadavků, které popisují, co by měl systém umět a jak by měl pracovat. Je to základní krok návrhu, bez kterého by byl další postup velice náročný a nepřehledný. Hlavním důvodem návrhu požadavků je sjednocení informací mezi vývojáři a zákazníkem. Každý má jinou představu o software, a proto se sjednotí představy obou stran a zajistí se větší přesnost při vývoji. Pravidelná komunikace mezi těmito dvěma stranami zajistí, že se software nebude muset celý předělávat, a tak bude i méně nákladný.

4.2.1 Funkční požadavky

Zde budou specifikovány detailní funkční požadavky, které mají za úkol detailně popsat klíčovou funkcionalitu systému. Bude se jednat o přesný popis chování systému sloužícího pro výpočet rozsahu software. Požadavky musí poskytnout specifikaci pro správu přihlášení, uživatele, výpočtů odhadů, historie. [13]



Obrázek 2 – Funkční požadavky

Požadavky, které jsou na obrázku (Obr. 2), specifikují různé aspekty systému. Zahrnují schopnosti systému práce s uživatelem, vytvoření, výpočet a smazání odhadu, možnost nahlédnutí a práci s historií výpočtů a další různé možnosti využití systému.

1. Správa přihlášení
    - **FP01: Systém umí registrovat uživatele –** Systém by měl umět registrovat uživatele do webové aplikace. Registrace bude probíhat zadáním důležitých informací o uživateli a uložením do databáze. Registrací uživatel dostane možnost vykonávat odhady a pracovat s nimi.
    - **FP02: Systém umí smazat účet registrovaného uživatele –** Systém by měl poskytovat možnost vymazat všechny informace o účtu uživatele a jeho vykonaných odhadech. Také zamezí následného přihlášení zadáním stejných informací a daný uživatel nebude mít možnost vidět obsah, jenž vyžaduje platný účet.

- **FP03: Systém umí přihlásit uživatele –** Systém by měl poskytovat možnost uživateli přihlásit se do systému v případě, kdy uživatel zadá platné informace o registrovaném účtu. Po přihlášení dostane uživatel možnost vidět historii výsledků a další obsah.
- **FP04: Systém umí odhlásit uživatele –** Systém by měl poskytovat možnost uživateli odhlásit se od účtu, pokud byl uživatel předtím přihlášen. Odhlášením systém zamezí přístup do účtu jiným uživatelům. Všechna vykonaná práce bude uložena v databázi a bez přihlášení nepřístupná.

2. Správa vstupů
    - **FP05: Systém umí vložit vstupy ručně –** Systém by měl poskytovat uživateli možnost vyplnit informace o jednotlivých prvcích systému, jako jsou informace o projektu, aktérech, případech užití a číselných hodnotách faktorů.
    - **FP06: Systém umí vložit soubor jako vstup –** Systém by měl poskytovat možnost vložení souboru, který bude použit jako vstup pro výpočet. Soubor, který bude vložen, musí mít předem danou příponu, a to .xml. Soubory končící na jinou příponu nebudou zpracovány.
    - **FP07: Systém umí kontrolovat, zda nejsou vstupy prázdné –** Systém by měl zjistit, zda vložené vstupy obsahují nějakou hodnotu nebo jsou prázdné. Kontrolovat by se měly vstupy v obou případech vložení. Při zjištění prázdného vstupu by měl systém oznámit chybu uživateli.
    - **FP08: Systém umí kontrolovat, zda jsou vstupy korektní –** Systém by měl zjistit, zda jsou zadané vstupy správné a mohou být použity do dalších výpočtů. V případě, kdy vstupy neodpovídají očekávané hodnotě, systém oznámí danou chybu uživateli, a to například v případě, kdy systém očekává číslo a uživatel zadá řetězec znaků.
    - **FP09: Systém umí zpracovat vstupy –** Systém by měl zpracovat vstupy z databáze do takové podoby, aby byla možnost vypočítat metodu. Systém by měl ze slovního zadání některých vstupů správně rozdělit jednotlivé prvky do určitých skupin a přiřadit jim číselnou hodnotu.
3. Správa výpočtů
    - **FP10: Systém umí vykonat výpočet metody Use Case Points –** Systém by měl uživateli poskytnout přesný výpočet metody use case points. Systém

přebere zadané vstupy a následně pomocí předem stanovených vzorců vykoná výpočet.

- **FP11: Systém umí vypsat přehled o odhadu –** Systém by měl poskytnout uživateli přehled o vykonaném odhadu, a to vypsáním všech informací o daném projektu včetně všech aktérů a dalších prvků. Pokud je projekt dokončený, systém by měl vypsat i výslednou hodnotu odhadu.
- **FP12: Systém umí vygenerovat PDF soubor –** Systém by měl poskytnout uživateli možnost vygenerovat PDF soubor, který bude obsahovat celkový přehled o projektu.
- **FP13: Systém umožní vygenerovat PDF pouze přihlášeným uživatelům –** Systém by měl poskytnout možnost vygenerovat PDF pouze uživatelům, kteří jsou uloženi v systému a současně přihlášeni. Pro nepřihlášené uživatele není přístupné generování PDF souboru.
- **FP14: Systém umí založit nový projekt se starými daty –** Systém by měl umožnit uživateli založit nový projekt s již použitými daty. Projekt by měl být úplně stejný s tím rozdílem, že bude umožněno pracovat s daty jako s novým projektem a bude vykonán nový odhad.
- **FP15: Systém umí založit nový projekt –** Systém by měl poskytnout přihlášenému uživateli možnost založit nový projekt. Projekt by měl mít jedinečný název. Systém by měl umožnit výpočet odhadu pro daný projekt.
- **FP16: Systém umí uložit informace o projektu do databáze –** Systém by měl poskytnout uživateli možnost uložit informace o projektu. Ukládání jednotlivých částí by mělo probíhat automaticky po přidání nějaké části.

4. Správa historie

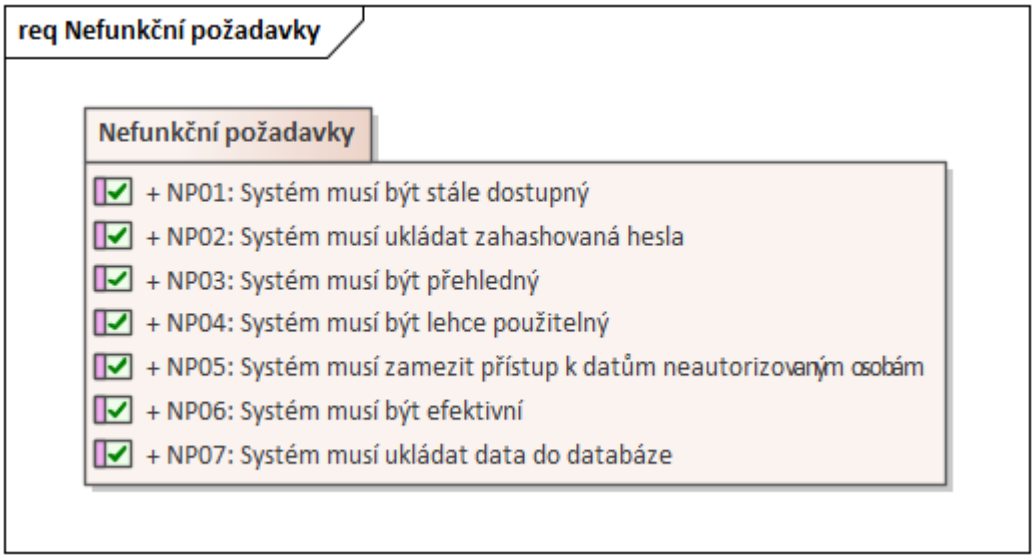
- **FP17: Systém umí smazat projekt –** Systém by měl poskytnout uživateli možnost smazání jednotlivých projektů z historie. Smazat by se měli všechny informace o projektu z databáze. Smazání by mělo být úplné, tedy by se měly smazat všechny dílčí části projektu.
- **FP18: Systém umí ukázat všechny vykonané projekty –** Systém by měl poskytnout uživateli nahlédnout na stránku historie, kde se nachází základní informace o vykonaných projektech. Na stránce historie dostane možnost detailnějšího náhledu na vybraný projekt.

- **FP19: Systém umí zamezit přístup nepřihlášenému uživateli do historie –** Systém by neměl poskytnout nepřihlášenému uživateli přístup na stránku historie. Pro přístup na tuhle stránku je podmínkou založený účet, a to i z důvodu ukládání výsledků k danému uživateli.
- **FP20: Systém umí ukázat detail projektu –** Systém by měl poskytnout uživateli možnost detailního popisu vybraného projektu. Všechny části projektu by měly být viditelné. Detail by měl být přístupný ihned po vykonání odhadu, ale i přístupný z historie výpočtů

5. Správa uživatele
    - **FP21: Systém umí ukázat uživateli jeho osobní informace –** Systém by měl uživateli poskytnout přístup na stránku účtu, kde by měly být ukázány jeho osobní informace, které zadal při registraci. Tyto informace jsou viditelné pouze danému uživateli.
    - **FP22: Systém umí změnit heslo daného uživatele –** Systém by měl uživateli poskytnout možnost změny hesla v případě, kdy bude uživatel chtít. Změna hesla by neměla nijak ovlivnit účet uživatele, ani nijak změnit uložené výpočty. Pouze změnit heslo při přihlášení do systému.
    - **FP23: Systém umí změnit osobní informace daného uživatele –** Systém by měl poskytnout uživateli možnost na změnu osobních informací, které jsou uloženy v systému.

4.2.2 Nefunkční požadavky

Stejně jako funkční požadavky jsou nefunkční velmi důležité. Nefunkční požadavky definují charakteristiku a vlastnosti systému. V některých případech mohou vést k vytvoření funkčních požadavků [13]. Výběr požadavků, které se hodí pro aplikaci na odhad rozsahu software můžeme vidět na obrázku (Obr. 3). Jako ve většině návrzích i zde jde o to, aby byla výsledná aplikace co nejvíce bezpečná a spolehlivá.



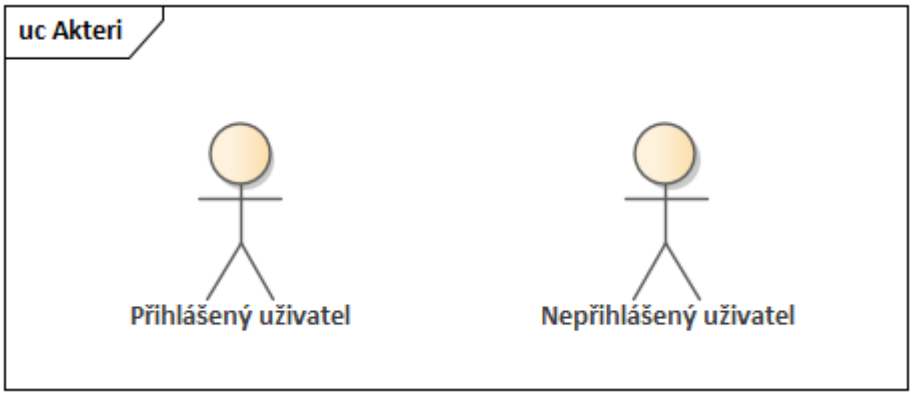
Obrázek 3 – Nefunkční požadavky

- **NP01: Systém musí být stále dostupný –** Systém by měl být dostupný v jakémkoliv čase. To znamená, že uživatel bude moci používat systém v takovou dobu, jakou uzná za vhodnou, bez ohledu na to, kolik je hodin. Systém musí být dále dostupný po celý čas, po který je daným uživatelem používán.
- **NP02: Systém musí ukládat zahashovaná hesla –** Systém by měl v každém případě, kdy se registruje nějaký uživatel, ukládat zahashovaná hesla. Tedy by měl uložit jakékoliv heslo takovým způsobem, aby nebylo možné z použitého tvaru vyčíst, o jaké heslo se jedná. Použití kódování hash je nevratné, a tedy bezpečné.
- **NP03: Systém musí být přehledný –** Systém by měl být v takovém tvaru, aby se v něm dokázal vyznat i uživatel, který ho poprvé navštíví a nemá s ním dřívější zkušenost. Hlavní funkcionalita systému by měla být lehce k nalezení a jasně znázorněna. Systém by neměl působit odpudivě a náročně.
- **NP04: Systém musí být lehce použitelný –** Systém by měl být lehce použitelný i pro uživatele, který není s podobnými systémy tak zkušený. Všechny prvky v systému by měly dávat smysl a být umístěny tak, aby bylo na první dojem jasné na co slouží. Popřípadě by měl být použit nějaký jednoduchý návod, jenž by pomohl se složitějšími částmi systému.
- **NP05: Systém musí zamezit přístup k datům neautorizovaným osobám –** Systém by měl fungovat takovým způsobem, aby se k datům sloužícím jednomu uživateli nedostal druhý uživatel. Jednotlivá data musí být správně přiřazena přihlášenému uživateli.

- **NP06: Systém musí být efektivní –** Systém by měl být efektivní. To znamená, že musí v co nejkratším čase splnit všechny přiřazené úlohy, a to bez nečekaných problémů.
- **NP07: Systém musí ukládat data do databáze –** Systém musí uložit všechna data o uživateli nebo uživatelem vytvořená data do databáze. Neměla by nastat situace, kdy budou nějaká důležitá data neuložena a po přepnutí do jiné části systému nadobro ztracena, pokud k tomu není důvod. Databáze by měla uchovávat některá data zašifrovaná, a to z důvodu úniku osobních dat.

4.3 Aktéři

Nedílnou součástí analýzy software jsou aktéři. Představují totiž abstraktní entitu, která přímo interaguje se systémem a stojí mimo hranice systému. Jako aktéra si můžeme představit například uživatele, extérní systémy nebo třeba hardware. Využití aktérů najdeme i v návrhu aplikace pro odhad software. Analyzovaný systém obsahuje dva aktéry, a to přihlášeného uživatele a nepřihlášeného uživatele. Každý aktér se v systému bude chovat jiným způsobem a dostane se do různých částí systému. Hlavním aktérem, který bude plnit většinu funkcí, které budou zpracovány, bude přihlášený aktér. Ten díky registraci dostane přístup k důležitým částem systému. Nepřihlášený aktér bude sloužit hlavně k přihlášení a registraci do systému. Oba tito aktéři jsou znázorněni graficky na obrázku (Obr. 4). [14]



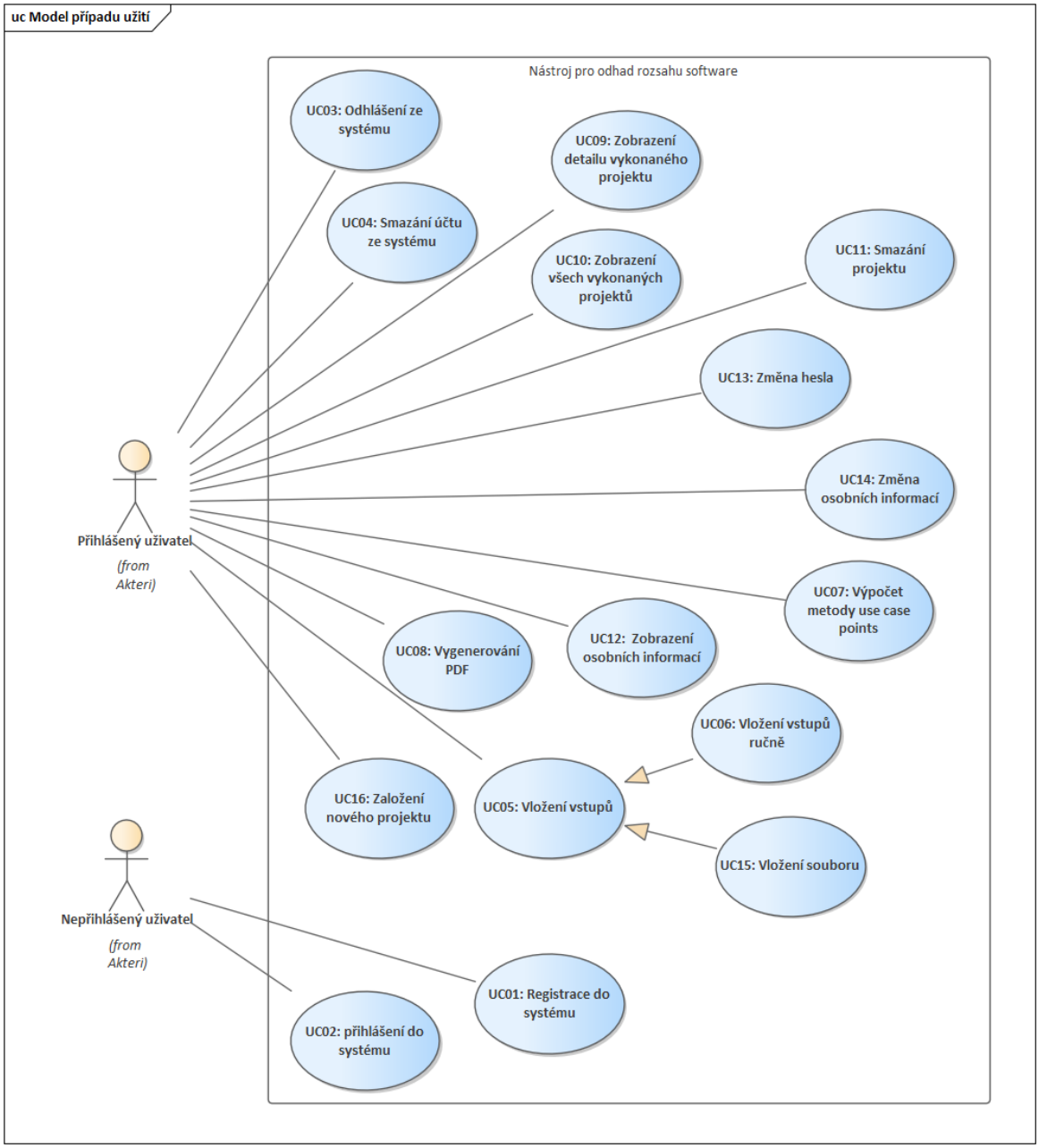
Obrázek 4 – Aktéři v systému

- **Přihlášený uživatel** – Aktér představující uživatele, který je zaregistrován a zároveň přihlášen. Tento aktér bude interagovat se systémem pouze v případě validního přihlášení do systému. Tento aktér bude moct vykonávat všechny výpočty a dále s nimi pracovat.

- **Nepřihlášený uživatel** – Aktér představující uživatele, jenž není v systému zaregistrován. Využití tohoto aktéra je v případě, kdy se daný uživatel potřebuje přihlásit nebo registrovat do systému.

4.4 Model případů užití

Jedná se o dynamický model popisující chování systému. Využívají se k realizaci požadavků. Cílem modelu případů užití je přiřazení jednotlivých případů užití jednotlivým požadavkům. To znamená, že každý požadavek by měl být zastoupen případem užití. Jedná se zde pouze o funkční požadavky. Případ užití popisuje chování systému z pohledu uživatele a pomocí kroků definuje přesný postup uživatele k tomu, aby dosáhl konkrétního cíle. [3]



Obrázek 5 – Model případů užití pro prototyp aplikace

Pro realizaci požadavků zmíněných výše slouží obrázek (Obr. 5), na kterém je znázorněn model případů užití. Na modelu jsou zastoupeni oba aktéři, jenž nějakým způsobem budou vstupovat do aplikace. Každý z těchto aktérů má přiřazené různé případy užití, a to podle toho, jak bude každý aktér pracovat s navrhnutým systémem. Z obrázku je více jasné, jaký bude rozdíl, zda je uživatel přihlášený či nikoliv. Přihlášený uživatel dostane možnost využití velkého počtu případů užití, ve výsledné aplikaci to znamená, že bude moct využít všech funkcí, oproti tomu nepřihlášený uživatel nebude moct vykonat téměř nic.

4.5 Specifikace případů užití

Dalším krokem v návrhu prototypu aplikace je specifikace případů užití. V praxi to znamená, že ke každému případu užití je vytvořen tzv. scénář. Jedná se o posloupnost kroků, kterou by uživatel, či jiná entita, vykonal při splnění daného problému. Primární scénář obsahuje ideální průchod případem užití. Myslí se tím to, že při vykonání nenastanou žádné problémy, ani žádné nečekané události, které by vedly k nesplnění cíle v dané situaci. Ovšem neexistuje situace, která by byla vždy ideální, a to ani zde. K tomu slouží tzv. alternativní scénář. Jedná se zde o situaci, kdy nastane nějaká chyba a výsledná situace se ubírá jiným směrem, než se očekávalo. To vše musí být očekáváno a ošetřeno. Alternativní scénář se využívá i v situacích, kdy má například uživatel více možností, jak pokračovat. Primární vykoná jednu možnost a alternativní druhou. V tabulce (Tab. 2) je znázorněn primární scénář případů užití UDC01, kde dochází k registraci uživatele do systému, a to bez sebemenších problémů. [3]

Tabulka 2 – Scénář pro registraci do systému

Název: Registrace do systému		
ID: UC001		
Charakteristika:		
Popisuje registraci uživatele do systému		
Primární aktér:		
Nepřihlášený uživatel		
Vedlejší aktéři:		
Nejsou		
Vstupní podmínky:		
Uživatel se musí nacházet v systému		
Výstupní podmínky:		
Uživatel musí mít možnost se přihlásit		
Hlavní scénář:		
Krok	Aktér/Systém	Popis
1	Nepřihlášený uživatel	Klikne na tlačítko "Registrovat"
2	Systém	Zobrazí prázdný formulář registrace
3	Nepřihlášený uživatel	Vyplní platné informace do formuláře
4	Nepřihlášený uživatel	Klikne na tlačítko "Registrovat"
5	Systém	Zkontroluje, zda jsou validní vstupy
6	Systém	Potvrdí správné vstupy
7	Systém	Vytvoří účet uživateli
8	-	Tímto případ užití končí

Alternativní scénáře:
UC001a – Nevalidní nebo prázdné vstupy UC001b – Email je již zaregistrovaný

V tabulce (Tab. 2) je zobrazen primární scénář pro registraci do systému. Je v něm znázorněn přesný postup, jak uživatel postupuje při registraci do systému a jak systém reaguje na jeho pokyny. Ve scénáři vystupuje pouze nepřihlášený uživatel, který jako jediný má možnost registrace, jak je znázorněno v modelu případů užití (Obr. 2).

Tabulka 3 – Alternativní scénář pro registraci do systému: Nevalidní vstup

Název – Alternativní scénář: Nevalidní nebo prázdné vstupy		
ID: UC001a		
Charakteristika:		
Zachycení alternativního toku případu užití		
Alternativní scénář:		
Krok	Aktér/Systém	Popis
1	Systém	Nalezne chybu ve vstupech
2	Systém	Vypíše chybovou hlášku

V tabulce (Tab. 3) je znázorněn alternativní scénář pro registraci do systému (Tab. 2). Jedná se o scénář zachycující výjimku v zadání vstupů od uživatele. Případ může nastat v situaci, kdy uživatel zadá špatná data (např. číslo v kolonce „Jméno“) nebo nezadá žádná. Systém zareaguje vypsáním chybové zprávy jako je vidět ve scénáři.

Tabulka 4 – Alternativní scénář pro registraci do systému: Neplatný email

Název – Alternativní scénář: Email je již zaregistrovaný		
ID: UC001b		
Charakteristika:		
Zachycení alternativního toku případu užití		
Alternativní scénář:		
Krok	Aktér/Systém	Popis
1	Systém	Nalezne email, který je již zaregistrovaný v systému
2	Systém	Vypíše chybovou hlášku

V tabulce (Tab. 4) je druhý alternativní scénář pro registraci do systému. Tentokrát zachycuje výjimku, kdy se bude chtít registrovat uživatel, který se již registroval a daný email se již nachází v databázi registrovaných uživatelů.

Tabulka 5 – Scénář pro vložení vstupů ručně

Název: Vložení vstupů ručně		
ID: UC002		
Charakteristika:		
Popisuje vkládání vstupů ručně od uživatele		
Primární aktér:		
Přihlášený uživatel		
Vedlejší aktéři:		
Nejsou		
Vstupní podmínky:		
Nejsou		
Výstupní podmínky:		
Nejsou		
Hlavní scénář:		
Krok	Aktér/Systém	Popis
1	Přihlášený uživatel	Vloží informace o aktérech
2	Přihlášený uživatel	Vloží informace o případech užití
3	Přihlášený uživatel	Vloží číselné hodnoty technických faktorů
4	Přihlášený uživatel	Vloží číselné hodnoty faktorů prostředí
5	Systém	Je připraven na další kroky
6	-	Tímto případ užití končí
Alternativní scénáře:		
Nejsou		

Další scénář (Tab. 5) se týká vložení vstupů ručně. Uživatel by měl mít možnost vložit všechny vstupy, a to bez žádných vstupů od systému.

Tabulka 6 – Scénář pro vložení souboru jako vstupu

Název: Vložení souboru jako vstupu
ID: UC003
Charakteristika:
Popisuje vkládání souboru jako vstupu
Primární aktér:
Přihlášený uživatel

Vedlejší aktéři:		
Nejsou		
Vstupní podmínky:		
Uživatel musí být přihlášený v systému		
Výstupní podmínky:		
Nejsou		
Hlavní scénář:		
Krok	Aktér/Systém	Popis
1	Přihlášený uživatel	Vloží soubor
2	Systém	Zkontroluje soubor, zda je povolený
3	Systém	Je připraven na další kroky
4	-	Tímto případ užití končí
Alternativní scénáře:		
UC003a - Špatný formát souboru		

Tabulka 7 – Alternativní scénář pro vložení souboru jako vstupu

Název – Alternativní scénář: Špatný formát souboru		
ID: UC003a		
Charakteristika:		
Zachycení alternativního toku případu užití		
Alternativní scénář:		
Krok	Aktér/Systém	Popis
1	Systém	Detekuje soubor, který není povolený
2	Systém	Vypíše chybovou hlášku

V další tabulce (Tab. 6) se nachází scénář pro vložení souboru, který slouží jako vstup. Dále je v tabulce (Tab. 7) jeho alternativní scénář, který detekoval špatný formát vkládaného souboru.

Tabulka 8 – Scénář pro výpočet metody use case points

Název: Výpočet metody use case points
ID: UC004
Charakteristika:
Popisuje průběh výpočtu metody
Primární aktér:
Přihlášený uživatel
Vedlejší aktéři:

Nejsou		
Vstupní podmínky:		
Uživatel musí být přihlášený v systému		
Výstupní podmínky:		
Nejsou		
Hlavní scénář:		
Krok	Aktér/Systém	Popis
1	Přihlášený uživatel	Klikne na tlačítko vypočítat
2	Systém	Zkontroluje, zda jsou všechny vstupy validní
3	Systém	Zkontroluje, zda se nenacházejí prázdné vstupy
4	Systém	Dosadí všechny vstupy do vzorců
5	Systém	Provede výpočet
6	Systém	Vypíše výsledek výpočtu
7	Systém	Uloží výsledek do databáze
8	-	Tímto případ užití končí
Alternativní scénáře:		
UC004a - Zpracování vstupů ze souboru UC004b – Nalezení nevalidních vstupů UC004c – Nalezení prázdných vstupů		

Asi nejdůležitější scénář se nachází v tabulce (Tab. 8) a popisuje průběh výpočtu vybrané metody use case points. Tento primární scénář počítá se vstupy, které jsou vloženy ručně a jsou bez problémů.

Tabulka 9 – Alternativní scénář pro výpočet: Zpracování souboru

Název – Alternativní scénář: Zpracování vstupů ze souboru		
ID: UC004a		
Charakteristika:		
Zachycení alternativního toku případu užití		
Alternativní scénář:		
Krok	Aktér/Systém	Popis
1	Systém	Otevře soubor
2	Systém	Zpracuje soubor
3	_	Pokračuje se krokem 2

V tabulce (Tab. 9) je první alternativní scénář pro výpočet, a to scénář pro zpracování souboru. Zde dochází k přidání zpracování souboru a napojení na další kroky do primárního scénáře.

Tabulka 10 - Alternativní scénář pro výpočet: Nevalidní vstup

Název – Alternativní scénář: Nalezení nevalidních vstupů		
ID: UC004b		
Charakteristika:		
Zachycení alternativního toku případu užití		
Alternativní scénář:		
Krok	Aktér/Systém	Popis
1	Systém	Nalezne nevalidní vstup
2	Systém	Zobrazí chybovou hlášku

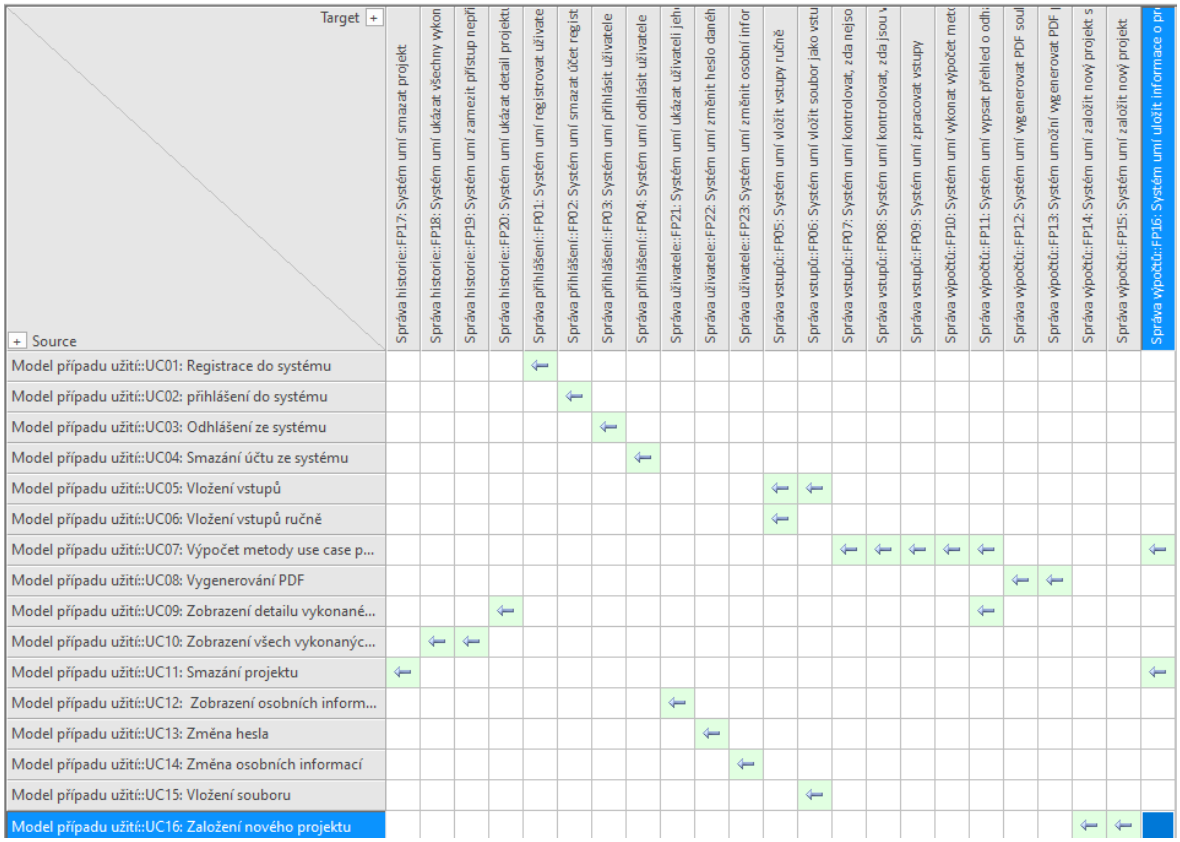
Tabulka 11 – Alternativní scénář pro výpočet: Prázdný vstup

Název – Alternativní scénář: Nalezení prázdného vstupu		
ID: UC004c		
Charakteristika:		
Zachycení alternativního toku případu užití		
Alternativní scénář:		
Krok	Aktér/Systém	Popis
1	Systém	Nalezne prázdný vstup
2	Systém	Zobrazí chybovou hlášku

Tabulky (Tab. 11, Tab. 12) obsahují alternativní scénáře pro výpočet, a to pro nalezení prázdných vstupů a pro nalezení nevalidního vstupu, kterým se myslí např. slovo místo číselné hodnoty.

4.6 Realizace případů užití

Poslední zmíněnou částí analýzy bude realizace případů užití. Jedná se o přehled vytvořených případů užití, které mají pokrýt požadavky. Jak již bylo zmíněno, každý požadavek by měl být pokrytý minimálně jedním případem užití. Ke grafickému znázornění přehledu realizace slouží tzv. matice vztahů. Jedná se o matici, kde jsou zapsány všechny požadavky spolu se všemi případy užití. Pokud některý případ užití pokrývá některý požadavek, najde se jeho průsečík a označí se. V matici lze snadno zobrazit případ, kdy jeden případ užití pokrývá více požadavků najednou.

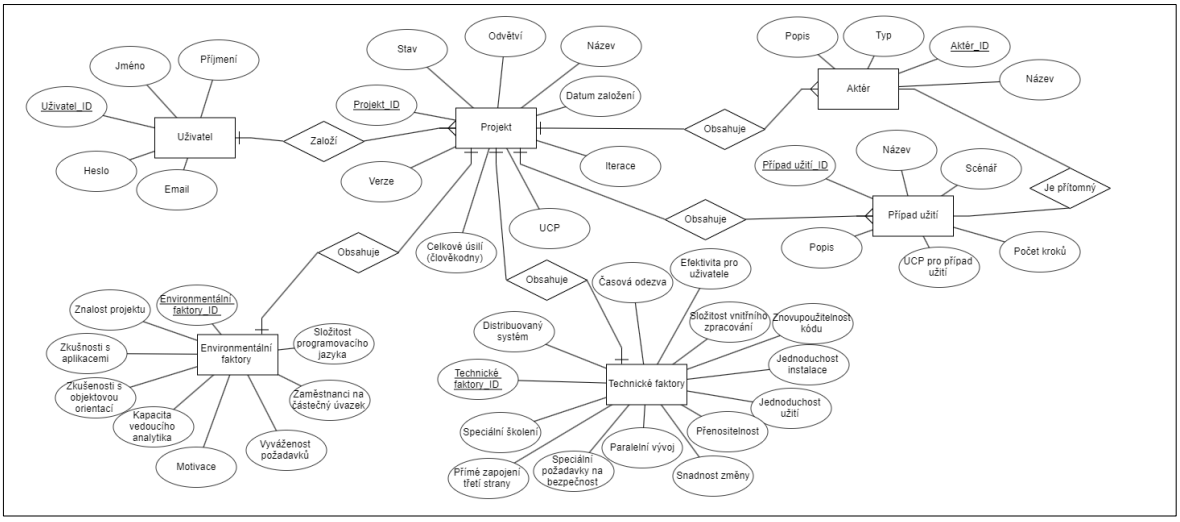


Obrázek 6 – Matice vtahů mezi požadavky a případy užití

Na obrázku (Obr. 6) je zřetelně viditelné, že je pokryt každý funkční požadavek. Nachází se zde více případů, kdy jeden případ užití zpracovává více požadavků. Tímto grafickým zobrazením je potvrzeno, že nedochází k případu, kdy by navrhnutý požadavek nebyl zpracován.

4.7 Datový model

Prvním diagramem, který se týká samotného návrhu aplikace, bude ERD (entity-relationship diagram). Tento diagram slouží pro grafický návrh databázového systému. Jak jeho název napovídá, obsahuje entity a vztahy mezi entitami. Entita představuje tabulku v databázi. K entitám by měly být vždy přiřazeny atributy, což jsou vlastnosti dané entity (např. jméno nebo věk).



Obrázek 7 – ER diagram

Na obrázku (Obr. 7) je návrh tabulek pro řešený prototyp, který obsahuje šest entit a vztahy mezi nimi. Všechny tyto entity se v pozdější fázi vývoje promění na tabulky, do kterých se budou ukládat všechna důležitá data, jak o uživatelích, tak o informacích o odhadu.

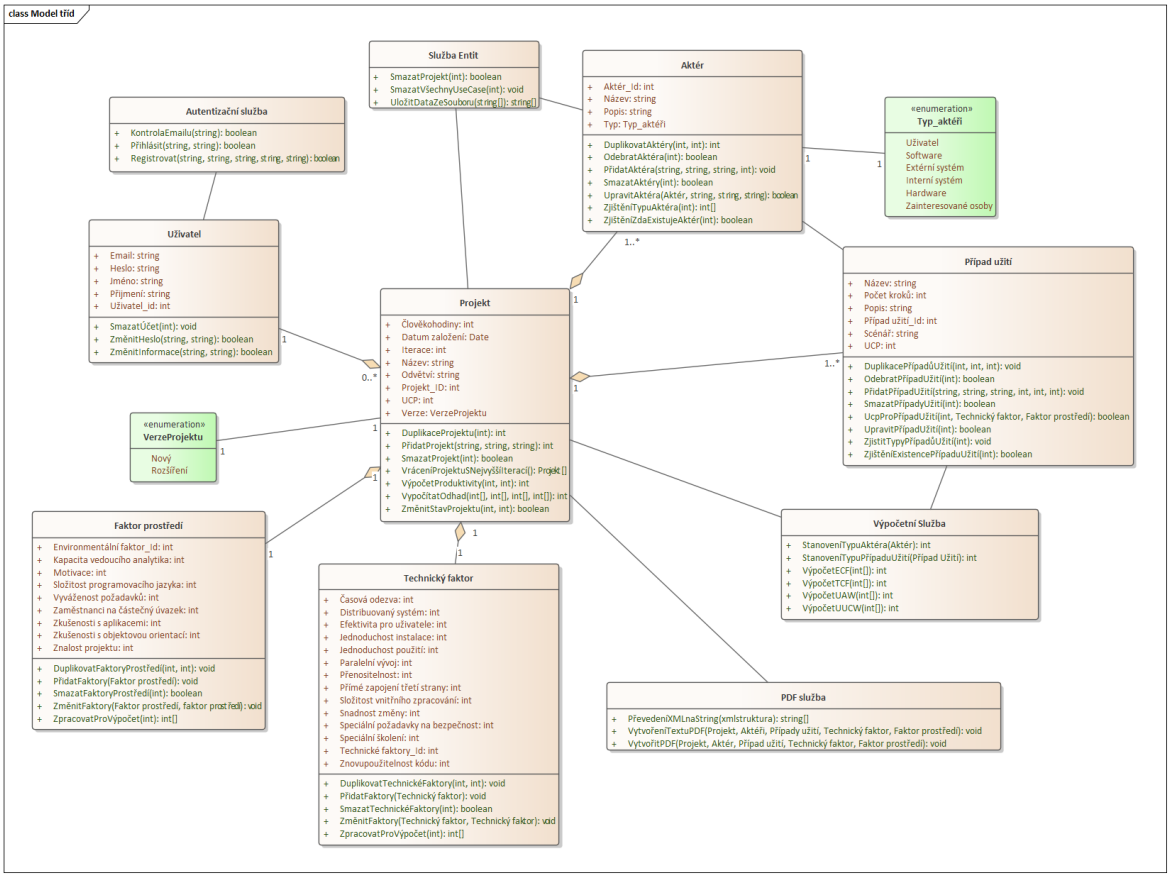
- **Uživatel –** Jedná se o entitu, která bude představovat všechny přihlášené uživatele. Po registraci do systému se vytvoří záznam v tabulce, který zpřístupní přihlášení uživateli. Atributy této entity představují základní informace o uživateli včetně hesla a přihlašovacího emailu.
- **Projekt –** Jedná se o entitu, která bude představovat základní informace o nově vytvořeném projektu spolu s výslednou hodnotou odhadu. Na tuto entitu budou navázány ostatní entity včetně přihlášeného uživatele. Při smazání záznamu v tabulce budou smazány i ostatní záznamy vázané na tuhle entitu, kromě uživatele.
- **Aktér –** Jedná se o entitu, která bude představovat jednotlivé aktéry vystupující v odhadu. Každý aktér bude mít vlastní záznam v tabulce, aby byl volně měnitelný nebo smazatelný v případě, kdy není potřeba. Atributy této entity představují informace jako název nebo stručný popis. Nejdůležitějším atributem je typ aktéra, jenž bude velmi důležitý v samotném odhadu.
- **Případ užití –** Jedná se o entitu, která bude představovat jednotlivé případy užití vystupující v odhadu. Stejně jako u entity aktér, atributy představují základní informace o případu užití. Každý záznam je vázaný na záznam v tabulce aktér, to znamená, že v případě, kdy nebude v projektu žádný záznam aktéra, nebude moci být vytvořený záznam o případu užití.

- **Technické faktory –** Jedná se o entitu, která bude uchovávat informace o technických faktorech vystupujících v odhadu. Pro každý projekt bude vytvořen jeden záznam v tabulce, kde budou zahrnuty všechny faktory, které budou volně měnitelné.
- **Environmentální faktory –** Jedná se o entitu, která bude uchovávat informace o environmentálních faktorech vstupujících do odhadu. Princip fungování této entity je stejný, jako u entity technické faktory.

Kromě zmíněných entit se v diagramu nachází i vztahy, které dodávají lepší představu o tom, jakým způsobem budou tabulky v prototypu propojeny. Většina entit je vázána pouze na entitu projekt, která je v návrhu zásadní, ale existuje zde i vztah mezi entitami aktér a případ užití, který je stejně důležitý a bez kterého nebude finální aplikace fungovat správně.

4.8 Model tříd

Dalším krokem návrhu je grafické znázornění modelu tříd. Jedná se o návrh tříd, který obsahuje všechny třídy, jež by měly figurovat v konečné aplikaci. V modelu jsou znázorněny třídy, které obsahují atributy, podobně jako v ERD. Rozdíl je ovšem ten, že třídy v modelu tříd obsahují i operace, což jsou navrhnuté metody, které by měly být použity v implementaci. Díky návrhu modelu tříd máme lepší představu o použitých třídách a jednotlivých metodách. V mém případě je model tříd udělaný podle ER diagramu, s tím rozdílem, že jsou navrhnuté metody. V samotné implementaci bude každou entitu představovat jedna třída, která bude mít stejný název i atributy. Na obrázku (Obr. 8) je návrh modelu tříd. [15]



Obrázek 8 – Model tříd

- **Uživatel** – Třída, která představuje uživatele. Atributy popisují vlastnosti uživatele.
  - *SmazatÚčet(int)* – Jedná se o metodu, která má za úkol smazat daného uživatele a všechny projekty, které jsou s daným uživatelem spojené. Vstupním parametrem je id aktuálního uživatele, jenž má být smazán. Toto id je v metodě využito i na zjištění projektů založených uživatelem.
- **Projekt –** Třída, která představuje založený projekt. Atributy popisují vlastnosti jednotlivého projektu.
  - *VrácenìProjektuSNejvyššíIterací()* – Jedná se o metodu, která má za úkol vrátit seznam všech projektů spojených s aktuálním uživatelem. Pokud se v databázi nachází více projektů se stejným názvem, vrátí vždy ten, co má vetší a zároveň novější hodnotu iterace. Nemá vstupní parametr a vrací vybrané projekty, které jsou připravené na další práci.
  - *VypočítatOdhad(int[],int[],int[], int[])* – Jedná se o metodu, která má za úkol provést odhad metody use case points. Vstupními parametry jsou všechny potřebné hodnoty pro provedení odhadu. Výstupem z této metody je výsledná hodnota UCP, tedy hodnota odhadu.

- *ZměnitStavProjektu(int, int)* – Jedná se o metodu, která má za úkol změnit stav projektu z nedokončeného na dokončený. Druhým úkolem této metody je uložení výsledné hodnoty UCP do databáze. Vstupními parametry je id projektu a hodnota UCP.
- *VýpočetProduktivity(int,int)* – Jedná se o metodu, která má za úkol vypočítat produktivitu v člověkohodinách. Vstupními parametry je UCP hodnota odhadu a id projektu.

- **Aktér –** Třída, která představuje jednotlivého aktéra vystupujícího v odhadu. Atributy této třídy představují vlastnosti aktéra.
  - *ZjištěníTypuAktéra(int)* – Jedná se o metodu, která má zjistit typ všech aktérů, jenž mají být použiti v odhadu. Metoda sama rozhodne, do jaké skupiny zařadí aktéra, a to pomocí předem zvoleným pravidlům. Vstupním parametrem je id projektu a výstupem pole roztříděných aktérů připravených vstoupit do odhadu.
  - *ZjištěníZdaExistujeAktér(int)* – Jedná se o metodu, která má rozhodnout, zda se v databázi nachází alespoň jeden záznam aktéra u daného projektu. Tato kontrola je z důvodu přiřazení aktéra k případu užití. Vstupním parametrem je id projektu a výstupem pravda nebo nepravda.
- **Případ užití –** Třída, která představuje jednotlivý případ užití v odhadu. Atributy třídy popisují vlastnosti jednotlivých případů užití.
  - *UcpProPřípadUžití(int, Technický faktor, Faktor prostředí)* – Jedná se o metodu, která má vypočítat hodnotu ucp pro jednotlivé případy užití. Vstupními parametry jsou id projektu, technické faktory a faktory prostředí. Metoda vypočítá hodnoty pro všechny případy užití najednou. Po vypočítání hodnoty pro jednotlivý případ užití, metoda uloží záznam do databáze.
  - *ZjistitTypyPřípadůUžití(int)* – Jedná se o metodu, která má zjistit typ všech případů užití, jenž mají být použiti v odhadu. Princip stejný jako v podobné metodě u entity aktér.
  - *ZjištěníExistencePřípaduUžití()* – Jedná se o metodu, která má rozhodnout, zda se v databázi nachází alespoň jeden záznam případu užití u daného projektu.
- **Technický faktor –** Třída, která představuje technické faktory vstupující do odhadu. Atributy třídy představují jednotlivé faktory.

- **Faktor prostředí –** Třída, která představuje faktory prostředí vstupující do odhadu. Atributy třídy představují jednotlivé faktory.

V návrhu modelu tříd jsou navrhnuty pro jednotlivé třídy metody, které jsou si podobné a mají stejný účel s tím rozdílem, že jsou navrhnuty pro různé prvky odhadu, ale mají stejný nebo podobný princip. Takovými metodami jsou:

- **Přidání prvků –** Jedná se o skupinu metod, která má za úkol přidat do odhadu nějaký prvek, který je zadán. Přidáním daného prvku je myšleno přidání záznamu do jednotlivých tabulek a další možná práce s nimi. Vstupními parametry jsou potřebné informace o dané entitě, v některých případech doplněné o id projektu.
- **Odebrání prvků –** Jedná se o dvě skupiny metod. První skupina odebírá pouze jeden záznam. Příkladem je odebrání jednoho aktéra při vkládání informací o aktérech. Druhou skupinou je smazání všech záznamů, které jsou spojeny s daným projektem. Vstupním parametrem je vždy id projektu nebo id daného prvku.
- **Úprava prvků –** Skupina metod, která upravuje daný záznam v databázi. Dostupné jen u některých tříd. Vstupními parametry jsou proměnné, které mohou být změněny.
- **Duplikace prvků –** Jedná se o skupinu metod, kde dochází k duplikaci všech prvků, a to z důvodu nové iterace projektu. Starý dokončený projekt zůstane beze změny a nový může být rozšířený nebo změněný.

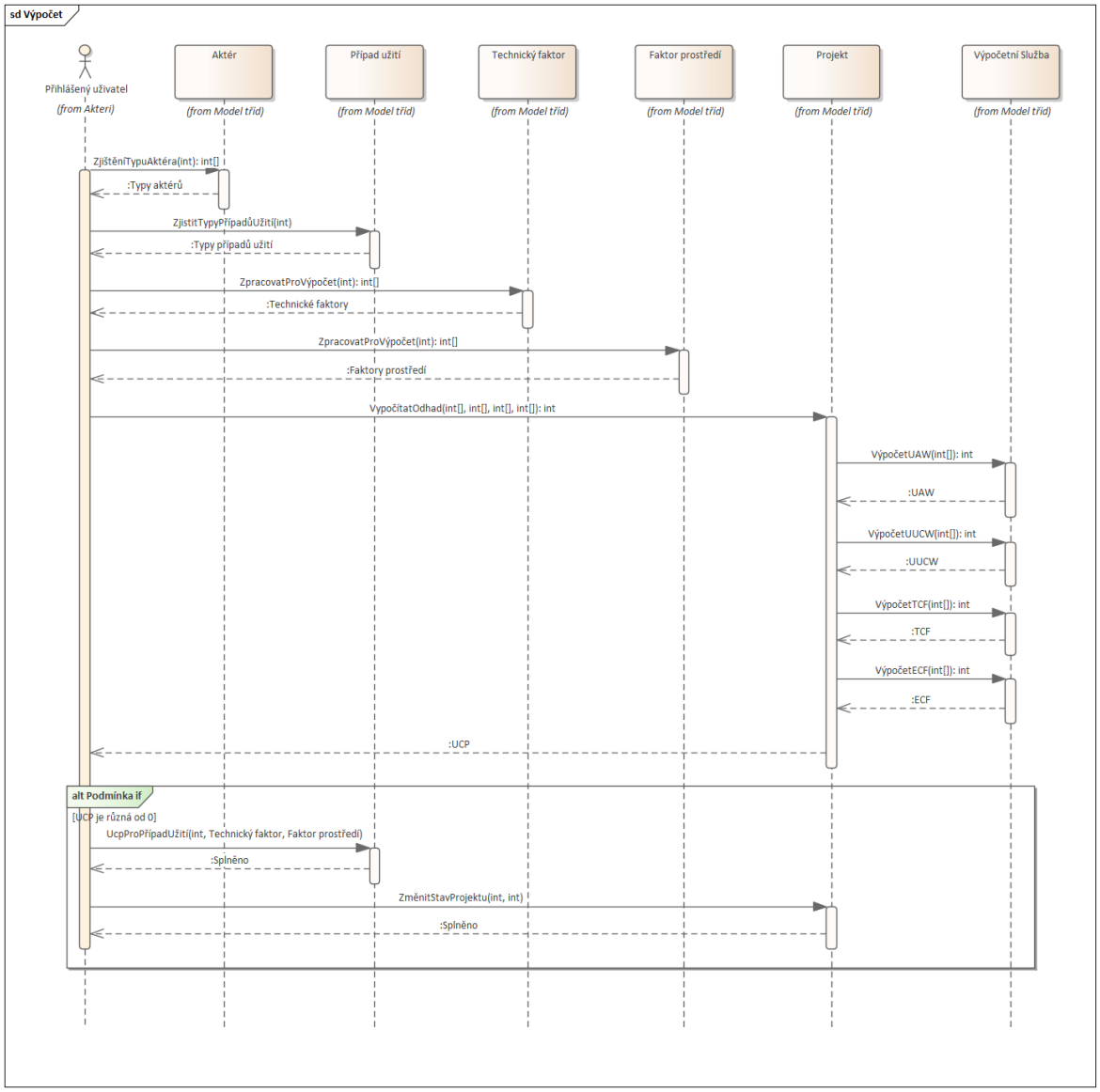
V návrhu modelu tříd se kromě tříd reprezentujících entity z ER diagramu nachází i rozšiřující třídy, které jsou označeny jako služby. Každá služba je vytvořena na vykonávání určité činnosti. Důvodem vytvoření takových služeb je zachování přehlednosti a znovu použitelnosti. Tyto třídy jsou:

- **Výpočetní služba –** Třída, která slouží k výpočetním operacím. Odehrávají se zde všechny dílčí výpočty, jež jsou důležité pro výpočet odhadu. Kromě výpočtů zde probíhá i stanovení typů aktéra a případu užití pro výpočet hodnot UCP pro jednotlivé případy užití.
- **Autentizační služba –** Třída, která slouží k autentizaci uživatele. Vykonává se zde registrace a přihlášení do systému. Dále je zde metoda pro kontrolu, zda daný email není již použit v systému.
- **PDF služba –** Třída, která slouží k práci se soubory PDF. Dochází zde ke generování souboru, kde se uchovávají všechny informace o vybraném projektu. Kromě tohoto zde probíhá i zpracování XML struktury a její převod na textový řetězec.

- **Služba entit –** Třída, která slouží k práci více entit zároveň, tedy třída, kde se potkávají všechny entity a nedochází zde ke kolizi importů. Nachází se tu metoda pro smazání celého projektu a metoda pro smazání všech případů užití, které jsou v projektu vázány na určitého aktéra. Dále je tu metoda pro uložení všech dat ze souboru do databáze.

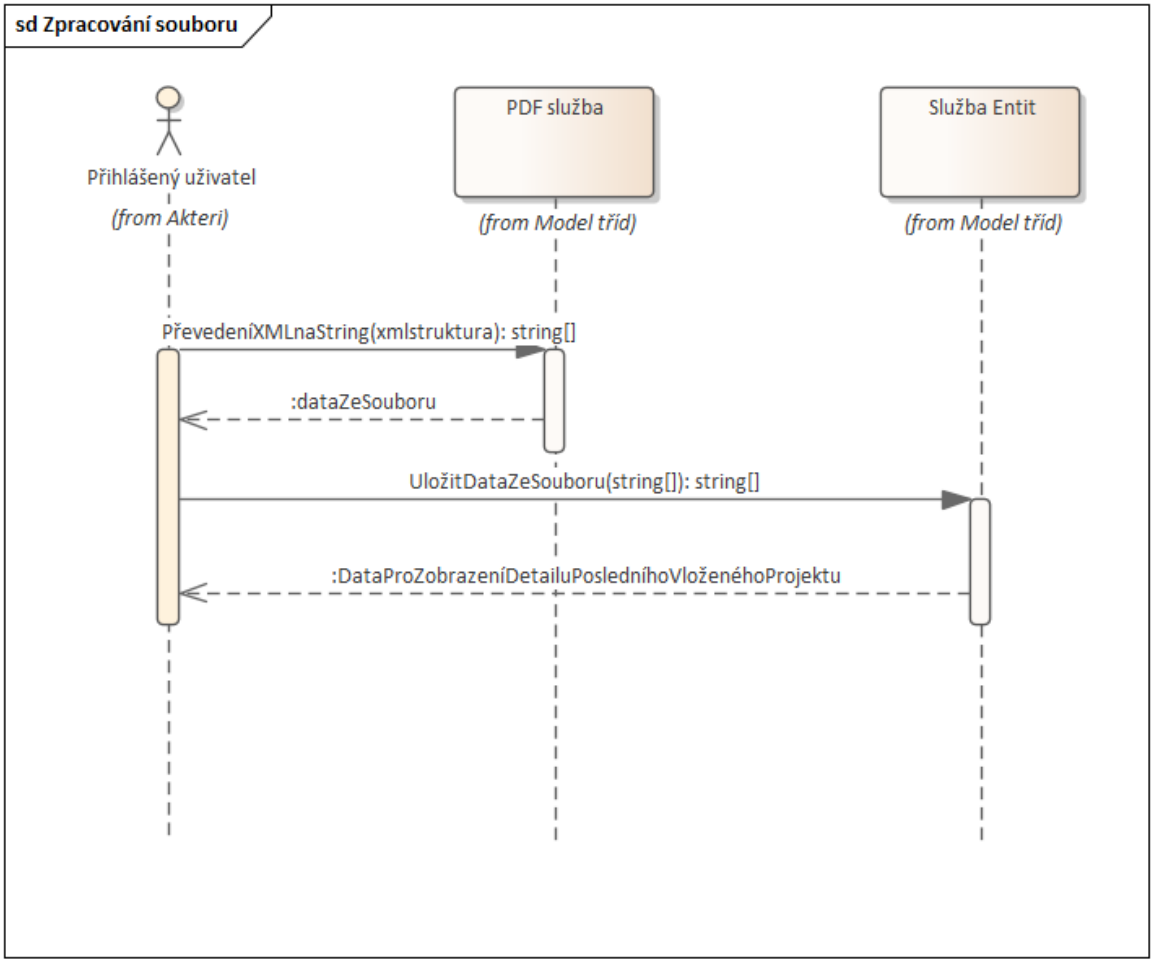
4.9 Realizace UC

Posledním krokem v návrhu prototypu aplikace bude realizace případu užití. K tomu se využije sekvenční diagram, který názorně ukáže, jak budou probíhat jednotlivé kroky za sebou. Tedy bude o něco málo zřetelnější, co se vykoná dříve a co později. I zde budou ukázány sekvenční diagramy nejdůležitějších případů užití, podobně jako u scénářů. [15]



Obrázek 9 – Sekvenční diagram: Výpočet metody use case points

Prvním sekvenčním diagramem je diagram ukazující průběh výpočtu vybrané metody, který je na obrázku (Obr. 9). Před samotným výpočtem proběhne nejdříve příprava, kde dojde ke zpracování vstupů. Jakmile jsou vstupy v požadovaném formátu, dojde k zavolání metody na výpočet. Jak již bylo zmíněno, pro jednotlivé výpočty je vytvořena třída výpočetní služba, kde dojde k výpočtu dílčích částí. Posledním krokem je podmínka, zda je výsledná hodnota větší než nula a poté se vypočítají hodnoty UCP pro jednotlivé případy užití a změní se stav projektu na dokončený.



Obrázek 10 – Sekvenční diagram: Zpracování souboru

Na obrázku (Obr. 10) je znázorněný sekvenční diagram pro zpracování souboru. Pro zpracování je použita PDF služba, kde je převedena XML struktura s daty na jednotlivé prvky, které se následně ve službě entit uloží do databáze.

5 IMPLEMENTACE PROTOTYPU APLIKACE

V této kapitole jsou ukázány všechny fáze vývoje prototypu aplikace. Jsou zde popsány fáze problematiky od založení projektu až po finální vzhled a funkčnost prototypu. Celý postup implementace včetně názvů proměnných a metod bude v anglickém jazyku, tedy doporučeném jazyku pro vývoj aplikací.

5.1 Úvod do implementace

V první fázi se nejdříve musí objasnit čeho se má dosáhnout a jakým způsobem se má práce vykonávat.

5.1.1 Cíl implementace

Cílem implementace je dosáhnout funkčního prototypu aplikace pro odhad rozsahu software. Aplikace by měla splňovat všechny vlastnosti, které byly znázorněny a popsány dříve v analýze a návrhu. Měly by být splněny všechny požadavky, a i způsob použití by měl odpovídat návrhu. V ideálním případě by měl být uživatel schopen při zadání validních vstupů, nebo při vložení souboru, dostat správný odhad rozsahu software. Dále by měly být splněny podmínky, které jasně říkají, kde se může uživatel pohybovat, v závislosti na tom, jestli je přihlášený či nikoliv. Splněna by měla být i funkčnost databáze, která by měla uchovávat všechny důležité informace a data, jak už o uživatelích, tak o provedených výpočtech. Vzhledem k tomu, že se jedná o prototyp, konečná aplikace by měla být rozšířena o další výpočty, ale o tom v poslední kapitole.

5.1.2 Vývojové prostředí

Pro prototyp aplikace s microframeworkem Flask může být použito několik vývojových prostředí (dále jen IDE – *Integrated Developnent Enviroment*). Neexistuje žádné ideální, záleží čistě na preferencích vývojáře. Pro vývoj bude použito vývojové prostředí Visual Studio Code, které má označení open-source, tedy je zdarma přístupné všem uživatelům. Jedná se o IDE, ve kterém lze vytvořit různé typy aplikací, a to v různých programovacích jazycích. Pro vývoj webových aplikací se hojně využívá i díky své jednoduchosti a přehlednosti.

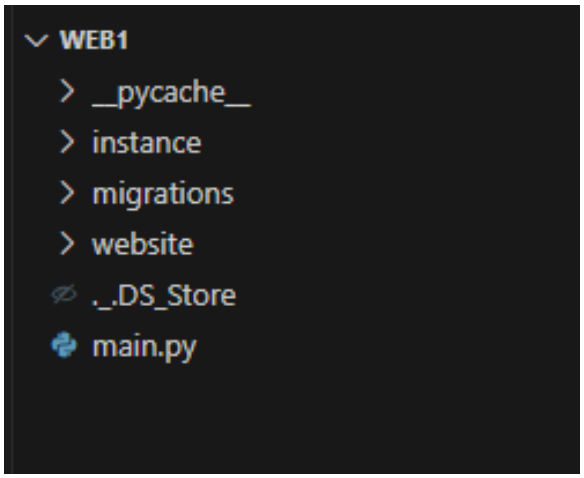
5.1.3 Založení projektu

Prvním krokem při založení nového projektu je stažení Pythonu do počítače. Bez tohoto stažení by následný projekt, ani žádné jiné projekty v programovacím jazyku Python

nefungovali. Dalším krokem je stažení a instalace balíčku Flask, který obsahuje důležité knihovny pro tvorbu webových stránek. Stažení a instalace probíhá v příkazovém řádku vybraného vývojového prostředí, a to po zadání příkazu „*pip install Flask*“ [16]. Tímto způsobem budou později staženy a nainstalovány další důležité balíčky s knihovnami, jež budou důležité ke správné funkčnosti a běhu aplikace.

5.2 Struktura projektu

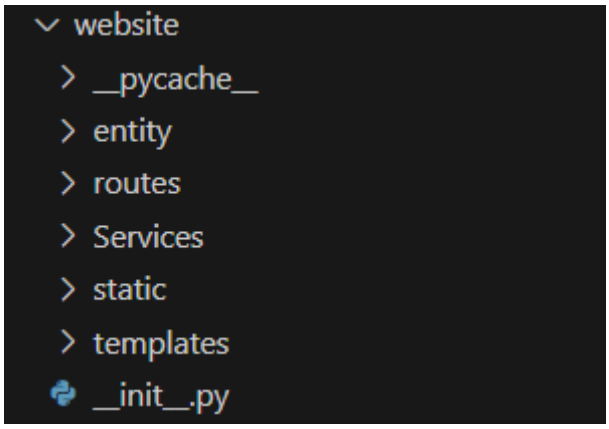
Důležitou součástí projektu je návrh struktury. Jedná se o místa uložení jednotlivých souborů. Struktura by měla být volena takovým způsobem, aby byl výsledný projekt co nejpřehlednější a dalo by se v něm rychle vyhledávat. Struktura projektu tvořeného microframeworkem Flask není přesně dána, takže si jí každý volí podle sebe. Ovšem je dobré dodržovat nějaké postupy, aby se v projektu orientovalo více vývojářů. Adresář, kde jsou uloženy všechny soubory a podadresáře, se nazývá kořenový adresář. V případě popisované aplikace se takový adresář jmenuje *web1*. V něm se nacházejí další podadresáře. Nejdůležitějším z nich je *website*, kde jsou uloženy všechny další podadresáře, které obsahují soubory sloužící pro funkční část aplikace.



Obrázek 11 – Struktura aplikace

Na obrázku (Obr. 11) je vidět základní struktura aplikace. Podadresář *_pycache_* je automaticky generovaný adresář, kde je uložen soubor s příponou *.pyc*. Jedná se o soubor, který uchovává dříve zmíněný bajtkód. Ten se ukládá v paměti cache a právě ve zmíněném souboru, který se nachází v tomhle adresáři. Při opětovném spuštění programu bude z tohoto důvodu vše rychlejší [10]. Důležitou součástí kořenového adresáře je soubor *main.py*, který slouží jako hlavní soubor při načtení aplikace. Dalšími podadresáři, jenž se nacházejí

v kořenovém adresáři, jsou *instance*, který obsahuje databáze a *migrations,* který obsahuje migrace k databázím.



Obrázek 12 – Struktura *website*

Na obrázku (Obr. 12) je struktura adresáře *website*, který obsahuje všechny soubory sloužící k běhu aplikace. I zde se nachází *__pychace__*, jenž má tu stejnou funkci jako v kořenovém adresáři. Ve složce *entity* se nacházejí všechny třídy, jenž slouží jako entity v databázi a jsou totožné, jako v návrhu modelu tříd. V podadresáři *routes* se nacházejí soubory, které zajišťují přiřazování URL adres a slouží k pohybu po webové aplikaci. Neméně důležitým podadresářem je *Service* sloužící k uchování souborů, které obsahují funkčnost, jako je například generování PDF souborů nebo konání jednotlivých výpočtů. Dalším je podadresář *templates*, kde jsou uloženy soubory s příponou *.html* sloužící pro stránky, které může každý uživatel vidět. Posledním podadresářem nacházející se zde je podadresář *static.* V něm se nacházejí soubory s příponami *.css* a *.js*. Soubor s příponou *.css* obsahuje jazyk CSS, který slouží pro upravení jednotlivých prvků nacházejících se na stránce, tedy v HTML souboru. JS soubor obsahuje jazyk JavaScript, který slouží pro dynamické chování stránky. Struktura je volena takovým způsobem, aby byla co nejpřehlednější, a aby byl každý soubor znovupoužitelný.

5.3 Konfigurace prototypu aplikace

5.3.1 Inicializace

Inicializace aplikace nastává v moment zapnutí aplikace, a to voláním metody *create_app()*. Tato metoda se nachází v souboru *__init__.py*, tedy inicializační soubor aplikace nacházející se v adresáři *website.* Úkolem tohoto souboru je zajistit správnou funkčnost všech důležitých prvků aplikace. Nejdůležitější součástí souboru je inicializace samotné aplikace.

```
1  from flask import Flask
2
3  def create_app():
4      app = Flask(__name__)
5      app.config['SECRET_KEY'] = 'bakalarka_frydl'
```

Obrázek 13 – Inicializace aplikace

Na obrázku (Obr. 13) je vidět základní část metody, která je volána po spuštění aplikace. Na prvním řádku dochází k importu třídy Flask z modulu flask a na řádku čtyři se do proměnné *app* vytváří instance této třídy. Proměnná *__name__* pomáhá Flasku určit, kde se aplikace nachází. Řádek pět je důležitý z hlediska bezpečnosti aplikace. Označení *'SECRET_KEY'* znamená tajný klíč, který se používá k bezpečnému podepsání souboru cookie relace a také pro další různé rozšíření týkající se bezpečnosti [17]. Jedním z dalších prvků inicializovaných v souboru jsou tzv. blueprinty. Jedná se o moduly, které rozdělují aplikaci. To znamená, že se jednotlivé prvky aplikace se rozdělí podle toho, co vykonávají a vzniká tak strukturovanější a přehlednější projekt. Příkladem může být nový soubor, který se bude jmenovat *views.py*.

```
1  from flask import Blueprint, render_template
2
3  views = Blueprint('views', __name__)
4
5  @views.route('/', methods=['GET'])
6  def home():
7      return render_template('home.html')
```

Obrázek 14 – Založení souboru views.py

Na obrázku (Obr. 14) lze vidět kód nacházející se ve zmíněném souboru. Na prvním řádku je z modulu flask vložena třída *Blueprint* a metoda *render_template*. Na třetím řádku je názorně ukázáno vytvoření nové instance s názvem *views.* Tato instance slouží k organizaci skupiny souvisejících pohledů a dalších funkcí v rámci aplikace. Parametr *'views'* je identifikátor blueprintu, který se používá k registraci blueprintu. Proměnná *__name__* určuje, kde se blueprint nachází a tím pomáhá Flasku určit, kde hledat související soubory.

```
1  from .views import views
2
3  app.register_blueprint(views, url_prefix='/')
```

Obrázek 15 – Registrace blueprintu

Na obrázku (Obr. 15) je vidět jakým způsobem se registruje daný blueprint. Tento kód se nachází v inicializačním souboru, a to v metodě *create_app()*. Na prvním řádku se nachází relativní import, tedy import, který je bez nutnosti specifikovat absolutní cestu modulu. Na řádku číslo tři se vykoná samotná registrace, a to pomocí metody *register_blueprint()*, kde vstupující parametr *views* říká jaký blueprint se má registrovat a druhý parametr říká jaký bude mít každý prefix URL adresy, který bude spojen právě s blueprintem *views.* Po úspěšném vykonání tohoto kódu by se měla zobrazit úvodní strana webové aplikace. Tímto způsobem se budou registrovat všechny další blueprinty, které budou figurovat v aplikaci. Dalšími prvky nacházející se v inicializačním souboru je inicializace databáze, inicializace migrací k databázi a inicializace přihlášení uživatele do systému. Všechny tyto prvky jsou popsány samostatně.

5.3.2 Routing

Routing ve Flasku představuje mechanismus přiřazování URL adres pro navigaci v aplikaci. K definování rout se využívá dekorátor, tedy speciální funkce, která mění nebo rozšiřuje další funkci. Syntaxe definice route vypadá následovně: *@app.route('/home', methods = ['GET', 'POST'])*, přičemž znak zavináče představuje dekorátor, proměnná *app* představuje zaregistrovaný blueprint. Následuje metoda *route()*, která přebírá v tomhle případě dva parametry. Prvním z nich představuje podobu URL adresy, která může být libovolně zvolena a druhý parametr představuje možné HTTP metody, které může routa přijímat. Zde můžou být zvoleny různé HTTP metody, ovšem metoda GET a POST jsou dvě nejčastěji volené metody. Pokud parametr bude prázdný, automaticky se předpokládá, že se jedná o GET metodu. Tento typ zápisu se aplikuje na funkce. Každá taková funkce představuje jeden pohled (view). [18]

```
1  @account.route('/account')
2  @login_required
3  def account_page():
4      return render_template("account/account.html", user=current_user)
```

Obrázek 16 – Metoda *account_page*

Na obrázku (Obr. 16) je příklad routingu na jednoduché metodě *account_page()*, která slouží pro zobrazení pohledu účtu. Jedinou funkcí této metody je zobrazit stránku obsahující informace o uživateli. Na prvním řádku je popsaný routing, který neobsahuje parametr *methods*, tedy je zde automaticky doplněn o HTTP metodu GET. Druhý řádek zaručuje přístup na

stránku pouze přihlášenému uživateli. Dále se na obrázku vyskytuje metoda, jež vrací metodu *render_template()*, která zobrazí potřebný pohled a informace o aktuálním uživateli. Vstupem do této metody musí být i informace o uživateli, který chce vstoupit na stránku.

5.3.3 Autentizace a autorizace

Přihlášení do systému je v navrhovaném prototypu velmi důležité. Přihlášený uživatel dostane totiž možnost volného pohybu v systému. S tím dostane možnost vytvářet projekty a ukládat si je na později nebo je mazat. V microframeworku Flask se k přihlášení do systému používá modul *flask_login*, který v sobě obsahuje třídu *LoginManager*. Díky této třídě je možné se přihlásit. Nejdříve je třídu potřeba importovat do inicializačního souboru, kde se nachází již zmíněná metoda *create_app()*. Dále se musí vytvořit instance třídy, jako je vidět na obrázku (Obr. 17). Poté se nastaví pohled, kde je přihlašovací formulář. Vždy když bude chtít nepřihlášený uživatel využít funkčnost, kde musí být přihlášen, tak se automaticky zobrazí právě tento pohled. Metoda *login_app(app)* inicializuje login manager s Flask aplikací. Parametr *app* je instance aplikace Flask, ke kterému se váže login manager. Řádek pět je definování tzv. načítače *user_loader*. Jedná se o funkci, která slouží k opětovnému volání uživatele uloženého v relaci. Šestý a sedmý řádek definuje funkci *load_user*, která při zavolání vrací právě aktivního uživatele, který je uložený v databázi. Vstupem zde je id očekávaného uživatele. Pokud uživatel s tímto id neexistuje, vrátí se *None*. [19]

```
1  login_manager = LoginManager(app)
2  login_manager.login_view = 'auth.login'
3  login_manager.init_app(app)
4
5  @login_manager.user_loader
6  def load_user(id):
7      return User.query.get(int(id))
```

Obrázek 17 – Třída LoginManager

Tímto je přihlašování do systému funkční. Dále je potřeba umožnit uživateli se registrovat do systému. Registrace i přihlášení uživatele probíhá ve službě *AuthenticationService.py*, kde jsou vytvořeny metody *login()* a *sign_up()*.

5.3.4 Templates

Templates představují šablony textových souborů, které se využívají na zobrazení všech prvků na stránce. Jedná se tedy o soubory s příponou *.html*. Vzhledem k tomu, že jsou tyto

soubory vytvořeny v microframeworku Flask, mohou obsahovat i samotný kód napsaný v programovacím jazyku Python. Při vytváření webové stránky s python kódem se využívá engine Jinja2. Ten při spuštění dané stránky dokáže zpracovat a propojit jak klasický html kód, tak zmíněný python kód. Funguje to tak, že engine nejdříve zpracuje šablonu, které mohou být předána data a poté vytvoří klasický html kód a následně jej odešle klientovi. Oproti klasickým html souborům se za použití zmíněného engine vytvoří soubor, který se využívá jako společná část pro všechny další soubory. V případě prototypu se jedná o soubor *base.html*, jenž má právě tuto vlastnost.

```
<head>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1" />
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" rel="stylesheet">
  <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/font-awesome/4.7.0/css/font-awesome.min.css"
    crossorigin="anonymous" />

  <title>{% block title %}Home{% endblock %}</title>
</head>
```

Obrázek 18 - <head> část base dokumentu

Pouze zde se setkáme s kódem z obrázku (Obr. 18). Element <head> slouží ke konfiguraci a definici základních informací o webové stránce. Nachází se zde metadata, což jsou v případě na obrázku (Obr.18) informace o kódování znaků a zobrazení stránky v takovém měřítku, aby všichni uživatelé měli možnost se na stránku podívat při různě velikých monitorech nebo na mobilních telefonech. Dále se zde nachází důležité odkazy sloužící pro responzivní zpracování stránky. Posledním elementem je titulek stránky, kde je použita tzv. definice bloků. Jedná se o další funkci v engine Jinja2, která se využije ve společném dokumentu a v dalších dokumentech se rozšíří o informace určené pro danou stránku [20]. Syntaxe je ukázána na obrázku (Obr. 18) při použití elementu <title>, kde se na začátku stanoví začátek bloku a jeho název, poté se napíše název dané stránky, a nakonec se ukončí daný blok.

```
{% extends "base.html" %}
{% block title %}Account{% endblock %}
{% block content%}
{% endblock %}
```

Obrázek 19 – Rozšíření bloků

V jiném souboru představujícím jiný pohled se pouze napíše kód, který je na obrázku (Obr. 19) a mezi začátkem a koncem bloku se napíše vhodný kód pro danou stránku. Kromě definování bloků se tímto způsobem vytváří například podmínka if nebo smyčka for, ale i zde

musí být řádné ukončení. Podobným způsobem se řeší i vypsání proměnných v html souborech, a to napsáním předané proměnné do dvou složených závorek.

Dalším prvkem v souboru *base.html* je vytvoření navigační lišty, která obsahuje odkazy na nejdůležitější stránky v aplikaci. Dále je zde vytvoření zobrazení informačních zpráv, které jsou volány funkcí *flash()*. Ve společném dokumentu se definuje, jakým způsobem mají být zobrazeny a co mají ukazovat. Posledním elementem je zde definice bloku *content*, jenž vymezuje prostor pro další soubory, které rozšiřují společný dokument.

5.3.5 Databáze

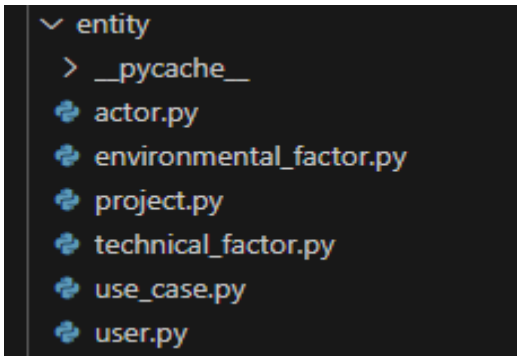
Všechny výpočty a další informace, které v prototypu aplikace vystupují, se musí nějakým způsobem ukládat. Nejlepší způsobem pro zvolený typ aplikace je uložení do databáze. V projektu je použití databáze spojeno s knihovnou sqlalchemy. Jedná se o knihovnu programovacího jazyka Python, která slouží jako nástroj pro mapování objektů do relačních databází a jako jazyk pro dotazování databáze. Po instalaci příkazem *pip install SQLAlchemy* [21] se musí databáze inicializovat, jako je na obrázku (Obr. 20).

```
1  db = SQLAlchemy()
2  DB_NAME = "database.db"
3
4  def create_app():
5      app.config['SQLALCHEMY_DATABASE_URI'] = f'sqlite:///{DB_NAME}'
6      db.init_app(app)
```

Obrázek 20 – Inicializace databáze

Nejdříve se importuje třída *SQLAlchemy*, ze které se pak vytvoří instance, a to představuje proměnná *db*. Poté se uloží jméno databáze. V dalším kroku dochází k samotné konfiguraci databáze, kde se specifikuje, že se použije engine SQLite a umístění databáze bude specifikováno proměnnou *DB_NAME*, a to v již dříve popsané metodě *create_app()*. Jakmile je databáze nakonfigurovaná, vznikne URI (Uniform Resource Identifier) pro databázi, což sqlalchemy využije ke spojení s danou databází. Posledním krokem je inicializace instance třídy *db* se samotnou aplikací. Po vykonání všech popsaných kroků by měla aplikace komunikovat s databází a pracovat s daty. Aby bylo možné ukládat data, musí se vytvořit tabulky, a to podle návrhu ER diagramu, který byl navrhnutý dříve. Pro každou navrhnutou entitu musí být vytvořena samostatná třída, jež musí obsahovat všechny navrhnuté atributy a propojení mezi jednotlivými tabulkami. Pro všechny tyto modely slouží samostatný adresář

entity. Jeho struktura je na obrázku (Obr. 21), kde je ukázáno, že obsahuje všechny navrhnuté entity, jež jsou v samostatném souboru pojmenovaném po nich. [22]



Obrázek 21 – Struktura entity

Pro vytvoření tabulky v databázi se musí specifikovat, jaké atributy bude daná tabulka mít. Na obrázku (Obr. 22) je příklad vytvoření modelu Project, který obsahuje informace o daném projektu.

```
1  from website import db
2  from sqlalchemy.sql import func
3
4  class Project(db.Model):
5      id = db.Column(db.Integer, primary_key = True)
6      title = db.Column(db.String(150), nullable=False)
7      date = date = db.Column(db.DateTime(timezone=True), default=func.now())
8      sector = db.Column(db.String(100),nullable=False)
9      version = db.Column(db.String(150),nullable=False)
10     iteration = db.Column(db.Integer,nullable=False)
11     ucp = db.Column(db.Integer,nullable=False)
12     man_hour = db.Column(db.Integer,nullable=False)
13     state = db.Column(db.String(50), nullable=False)
14     user_id = db.Column(db.Integer, db.ForeignKey('user.id'))
15     actors = db.relationship('Actor')
16     use_cases = db.relationship('UseCase')
17     tech_factors = db.relationship('TechnicalFactor')
18     env_factors = db.relationship('EnvironmentalFactor')
```

Obrázek 22 – Atributy modelu Calculation

Pro nastavení všech sloupců v modelu je potřeba importu *db*, což je již popsaná instance třídy SQAlchemy. Syntaxe jednotlivých sloupců je vždy taková, že první je název sloupce, po kterém následuje znak "=" a metoda *Column()*, která přebírá minimálně jeden parametr, a to datový typ sloupce. Dalšími parametry mohou být nastavení primárního klíče, automatické doplnění času vytvořeného záznamu nebo cizí klíč spojující tabulku s jinou tabulkou.

Pokud je použit v modelu cizí klíč, je potřeba to v daném modelu taky zapsat, a to pomocí metody *relationship*. Celý zápis vztahu je vidět na obrázku (Obr. 22) na dolních řádcích nebo na obrázku (Obr. 23), který se nachází v modelu *User*, tedy v modelu uchovávajícího informace o uživateli.

```
2  projects = db.relationship('Project')
```

Obrázek 23 – Zápis vztahu v modelu

Tím je vytvořený model pro databázi. Posledním krokem k vytvoření tabulky je použití metody *create_all()*, která má za úlohu vytvořit všechny tabulky, které detekuje a které zatím nejsou vytvořeny. Na obrázku (Obr. 24) je ukázaný kód, který vytvoří všechny tabulky. Před vykonáním zmíněné metody je potřeba naimportovat všechny modely, aby aplikace věděla, jaké modely jsou v aplikaci. Dále se vytvoří kontext aplikace, který umožní vytvoření tabulek v aplikaci bez potřeby být v kontextu webového požadavku. Tento kód se nachází v souboru *__init.py__* v metodě *create_app()*, která se volá vždy po zapnutí aplikace.

```
1  from .entity.user import User
2  from .entity.calculation import Calculation
3  from .entity.input import Input
4  from .entity.output import Output
5
6  with app.app_context():
7      db.create_all()
```

Obrázek 24 – Vytvoření tabulek

5.4 Funkčnost aplikace

Aplikace je navrhnutá tak, aby byla schopna vypočítat metodu use case points. Vzhledem k tomu, že se jedná o prototyp aplikace, je možno vypočítat pouze tuto metodu. Daný uživatel má možnost vložit vstupy dvěma způsoby, a to ručně, kdy musí postupně vložit všechny informace o jednotlivých aktérech a případech užití spolu s faktory, nebo nahrát XML soubor, ze kterého se informace automaticky nahrají do databáze a jsou k dispozici pro další změny či úplné vymazání. Nahraný soubor může obsahovat již vykonaný odhad. V takovém případě dostane uživatel možnost založení nové iterace, jako po vykonání odhadu.

5.4.1 Analýza vložených informací

Při nahrání souboru nebo vložení informací ručně se do databáze uloží obecné informace o projektu. Pro výpočet metody use case points je potřeba mít číselné hodnoty, a proto je nutnost mít v aplikaci vytvořené metody, které dokáží převést obecné informace na číselné hodnoty. V prototypu aplikace jsou vytvořeny metody pro analýzu aktérů a případů užití. Technické faktory a faktory užití nemusí být zpracovány, protože se vkládá číselná hodnota jednotlivých faktorů. Pro aktéry je vytvořená metoda *get_inputs_from_data()*, která je na obrázku (Obr. 25).

```
def get_inputs_from_data(project_id):
    actors = Actor.query.filter_by(project_id=project_id).all()
    types = {'simple': 0, 'average': 0, 'complex': 0}

    for actor in actors:
        if actor.atype == 'User' or actor.atype == 'Hardware': types['simple'] += 1
        elif actor.atype == 'Internal-system' or actor.atype == 'Software': types['average'] += 1
        else: types['complex'] += 1

    print(f'actor types: {types}')
    return types
```

Obrázek 25 – Metoda pro analýzu aktérů

Zde se načtou všichni aktéři, jenž jsou přiřazeny k danému projektu a pomocí jednoduchých podmínek se rozhodne, jakého typu je jednotlivý aktér. V podmínce je zmíněno šest různých možností aktérů. Tyto možnosti jsou v aplikaci přednastavené a nikdy nemůže nastat případ, že by aktér byl jiný. Metoda vrátí slovník s přesným počtem aktérů jednotlivých typů.

```
def get_input_from_data(project_id):
        use_cases = UseCase.query.filter_by(project_id=project_id).all()
        types = {'simple': 0, 'average': 0, 'complex': 0}

        for use_case in use_cases:
            if int(use_case.num_step) < 4: types['simple'] += 1
            elif int(use_case.num_step) > 7: types['complex'] += 1
            else: types['average'] += 1

        return types
```

Obrázek 26 – Metoda pro analýzu případů užití

Podobným způsobem je vyřešena i analýza případů užití, jež je na obrázku (Obr. 26). Zde se načtou všechny případy užití. Podmínkou je počet kroků scénáře. Metoda i zde vrací slovník s počtem jednotlivých typů. Technické faktory i faktory prostředí se nijak nezpracovávají.

Zde je jediným krokem vložení číselné hodnoty nula místo volných faktorů, které nebyly uživatelem zadány.

5.4.2 Výpočet metody use case points

Samotný výpočet metody probíhá ve třídě *Project*, kde je vytvořena metoda *use_case_points()*. Do této metody jsou vloženy všechny vstupy, které jsou získané z jednotlivých metod pro analýzu vložených informací. Pro větší přehlednost je každý výpočet vykonán ve třídě *CalculationService*, která slouží pouze k vykonání jednotlivých výpočtů. Celá metoda je na obrázku (Obr. 27).

```
def use_case_points(atypes, uctypes, tech_arr, env_arr):
        uucw = CalculationService.unadjusted_use_case_weight(uctypes)
        uaw = CalculationService.unadjusted_actors_weight(atypes)
        ucp = uucw + uaw

        tcf = CalculationService.technical_complexity(tech_arr)
        ef = CalculationService.environmental_complexity(env_arr)

        aucp = ucp * tcf * ef
        aucp = round(aucp, 2)

        return aucp
```

Obrázek 27 – Metoda na výpočet metody use case points

Na obrázku lze vidět, že se jednotlivé prvky počítají ve zmíněné službě. Jakmile jsou vypočítané všechny dílčí výpočty, provede se finální výpočet hodnoty UCP. Metoda vrátí právě zmíněnou hodnotu UCP, která je pak uložena do databáze a celkový stav databáze změněn na dokončený.

```
def unadjusted_use_case_weight(use_cases):
    uucw = use_cases['simple'] * 5 + use_cases['average'] * 10 + use_cases['complex'] * 15

    return uucw

def unadjusted_actors_weight(actors):
    uaw = actors['simple'] * 1 + actors['average'] * 2 + actors['complex'] * 3

    return uaw
```

Obrázek 28 – Metody pro výpočet vah aktérů a případů užití

Na obrázku (Obr. 28) jsou dvě metody, a to pro výpočet jednotlivých částí metody use case points. V obou případech jsou přebrány analyzované vstupy a dále zpracovány. Vrchní metoda je metoda pro případy užití a spodní pro aktéry.

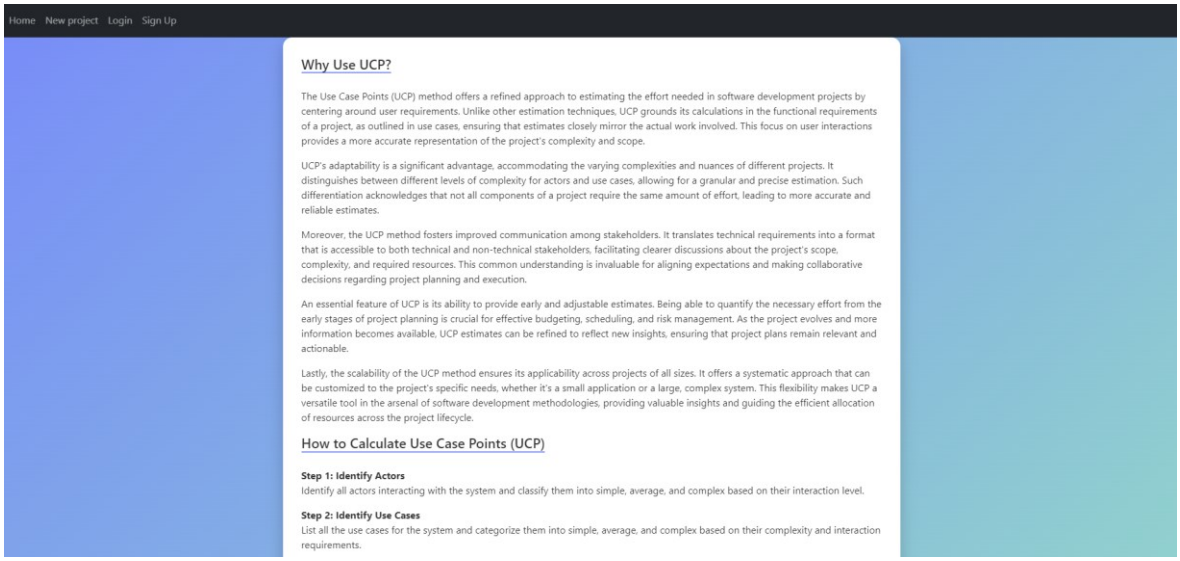
```
def technical_complexity(tf):
    weight = [2,2,1,1,1,0.5,0.5,2,1,1,1,1,1]
    TFactor = 0

    for i in range(len(tf)):
        if tf[i] != 0:
            TFactor += float(tf[i]) * float(weight[i])

    tcf = 0.6 + (0.01 * TFactor)

    return round(tcf,2)

def environmental_complexity(ef):
    weight = [1.5, 0.5, 1, 0.5, 1, 2, -1, 2]
    EFactor = 0

    for i in range(len(ef)):
        if ef[i] != 0:
            EFactor += float(ef[i]) * float(weight[i])

    ef = 1.4 + (-0.03 * EFactor)

    return round(ef,2)
```

Obrázek 29 – Metody pro zpracování faktorů

Na obrázku (Obr. 29) jsou metody pro zpracování faktorů. Princip je takový, že pokud aktuální prvek vloženého pole není nula, tak se vynásobí daný prvek s prvkem v poli obsahující váhu daného faktoru. Váhy jsou neměnné a pevně zapsány v metodě, a to ve všech případech počítání.

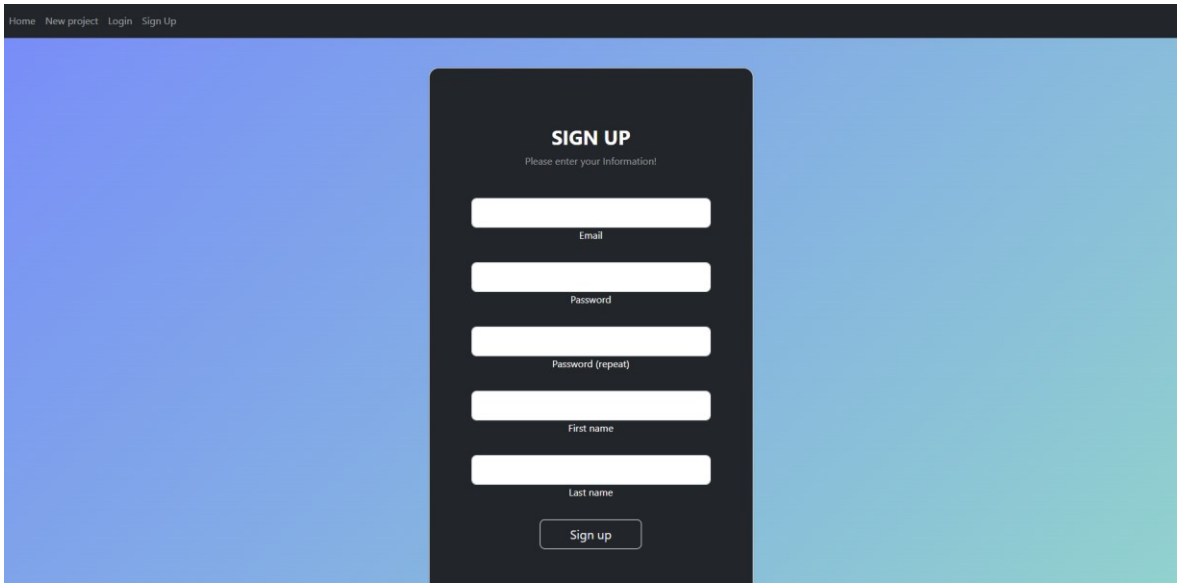
5.5 Popis pohledů prototypu aplikace

V navrhnutém prototypu je vytvořeno několik stránek, které nepotřebují, aby byl daný uživatel přihlášený. Jednou z nich je hlavní stránka, která se zobrazí v případě, kdy daný uživatel poprvé vstoupí do aplikace. Jejím obsahem je stručný popis metody use case points a následný jednoduchý návod, jakým způsobem se počítá tato metoda. V dolní části se nacházejí dvě tlačítka, která odkazují na vložení vstupů ručně a vložení vstupů pomocí souboru. Vrchní polovina stránky je vidět na obrázku (Obr. 30).



Obrázek 30 – Hlavní stránka aplikace

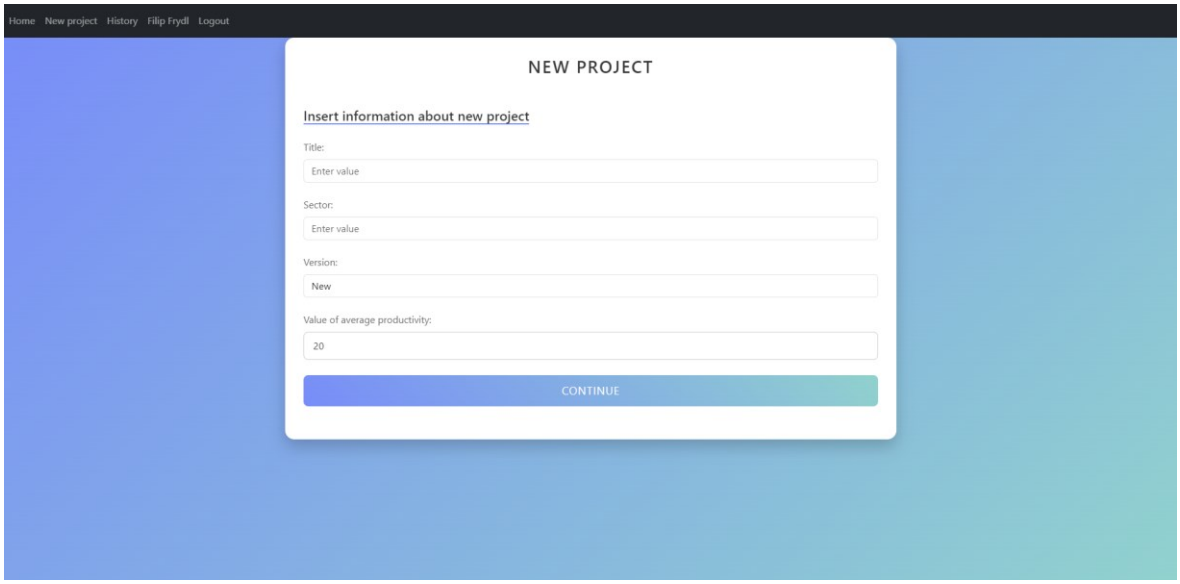
Dalšími volně přístupnými stránkami jsou stránka pro přihlášení a pro registraci do systému. Rozdílem mezi těmito pohledy je počet vstupů, kde se při registraci zadávají všechny povinné informace, přičemž při přihlášení se zadává pouze emailová adresa a heslo. Na obrázku (Obr. 31) je vidět formulář pro registraci. Stejně jako registrační a přihlašovací formulář vypadají i formuláře pro změnu hesla a změnu osobních informací. I zde je rozdíl pouze v počtu vstupů a funkčnosti.



Obrázek 31 – Registrační formulář

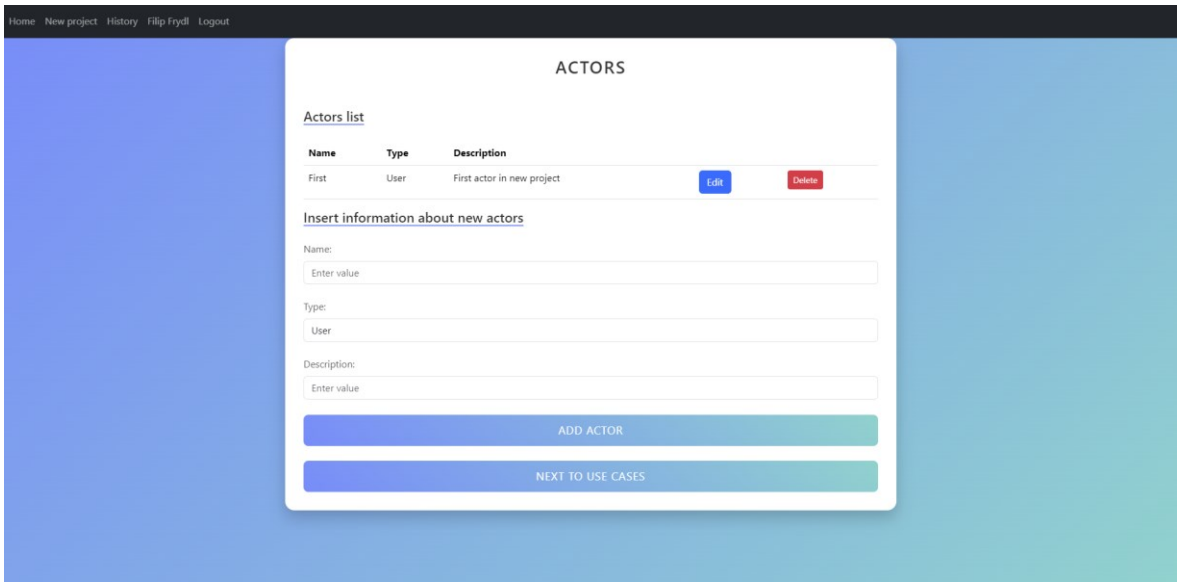
Po přihlášení do systému dostane uživatel možnost založení nového projektu. Prvním krokem je vyplnění základních informací o projektu, jako je název, odvětví projektu a výběr,

zda se jedná o nový projekt nebo jeho rozšíření. Pro tento krok v projektu je vytvořený pohled na obrázku (Obr. 32).



Obrázek 32 – Pohled pro založení nového projektu

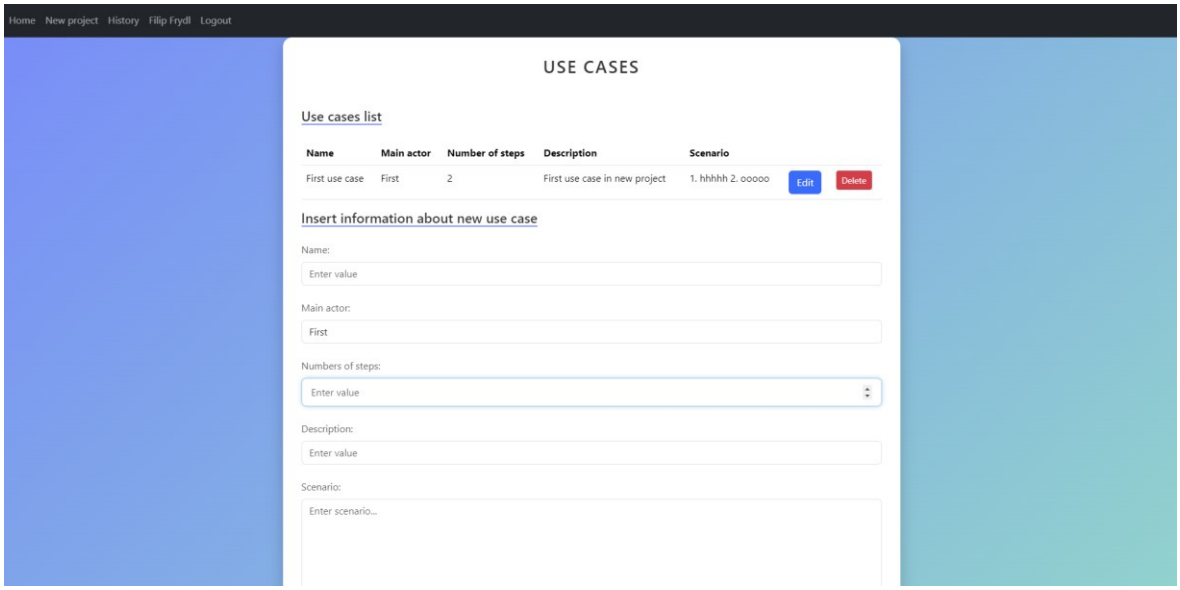
Dalším krokem projektu je vložení informací o aktérech vystupujících v projektu. Zde se vloží všechny důležité informace o aktérovi, jako je vidět na obrázku (Obr. 33). Zvolení typu aktéra je omezeno na určitý počet typů, ze kterých si musí uživatel vybrat.



Obrázek 33 – Pohled pro vložení aktéra do projektu

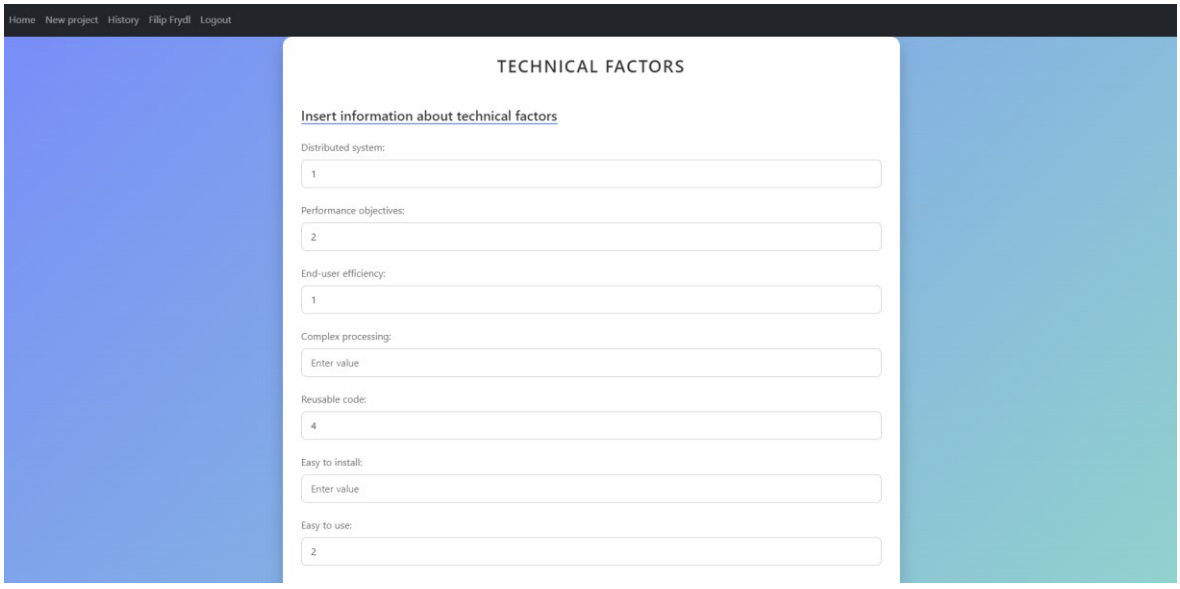
Po přidání alespoň jednoho aktéra dochází k dalšímu kroku, a to k vložení jednoho či více případů užití. I zde se nabízí vložit základní informace o daném případu užití včetně celého

scénáře. Jak je vidět na obrázku (Obr. 34), jedním ze vstupů je výběr hlavního aktéra. Vybraným aktérem může být pouze aktér, jenž je přidán do projektu v předešlém kroku.



Obrázek 34 – Pohled pro přidání případů užití

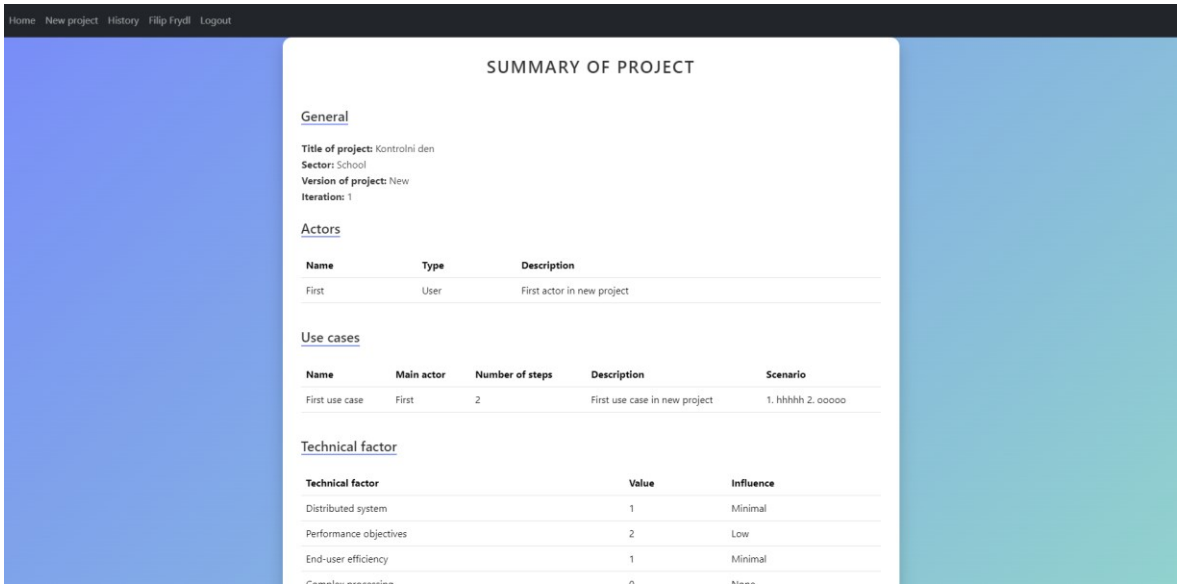
V dalších dvou krocích se zadávají číselné hodnoty pro dané faktory. Na obrázku (Obr. 35) je příklad vložení technických faktorů. Při nechání volného pole pro vložení hodnoty se danému faktoru připíše automaticky nula. Stejně funguje a vypadá vkládání faktorů prostředí.



Obrázek 35 – Pohled pro vložení číselných hodnot faktorů

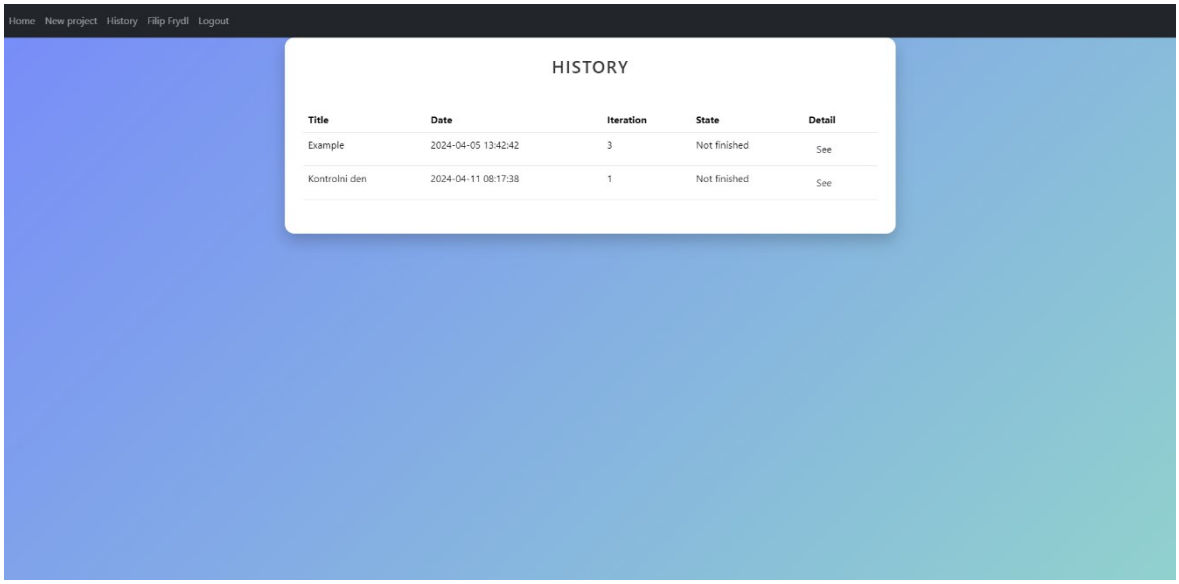
Tímto krokem končí vkládání informací o projektu. Před vykonáním odhadu dostane uživatel možnost si zkontrolovat, zda jsou všechny vložené informace v pořádku nebo je potřeba je změnit. To vše jde udělat v přehledu, jenž je na obrázku (Obr. 36). V případě, kdy je odhad

vykonán, tak již není možnost měnit jednotlivé prvky, lze pouze nahlédnout. Existuje i možnost založení nové iterace projektu, to znamená, že uživatel může měnit prvky projektu, aniž by změnil původní odhad. V tomhle případě se udělá nový projekt se stejnými daty, ale s rozdílnou iterací. Uživatel je tak schopný porovnat projekty s různými iteracemi.



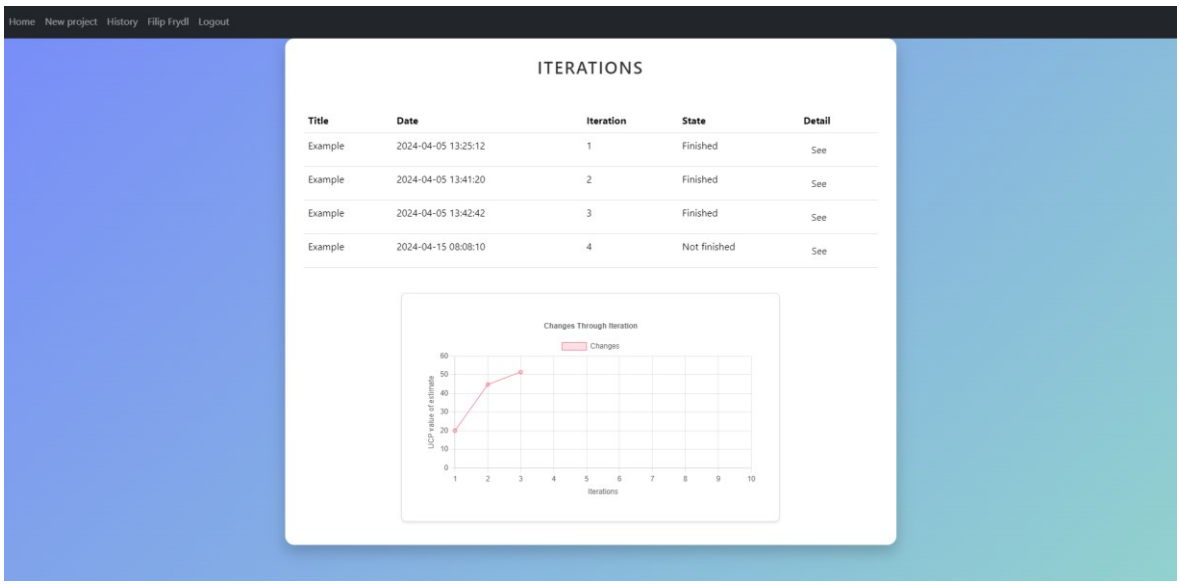
Obrázek 36 – Pohled pro přehled o projektu

Při vykonání odhadu se objeví stejný přehled s přidanými informacemi o výsledné hodnotě UCP a zmizí možnost úpravy dat a provedení odhadu. Při založení nového projektu se vše průběžně ukládá do databáze. To znamená, že po založení se dá zpětně vstoupit do projektu i z historie, která lze vidět na obrázku (Obr. 37). Tam se zobrazí všechny projekty, jež jsou založeny. Jedinou podmínkou je, že se nesmí dva projekty jmenovat stejně.



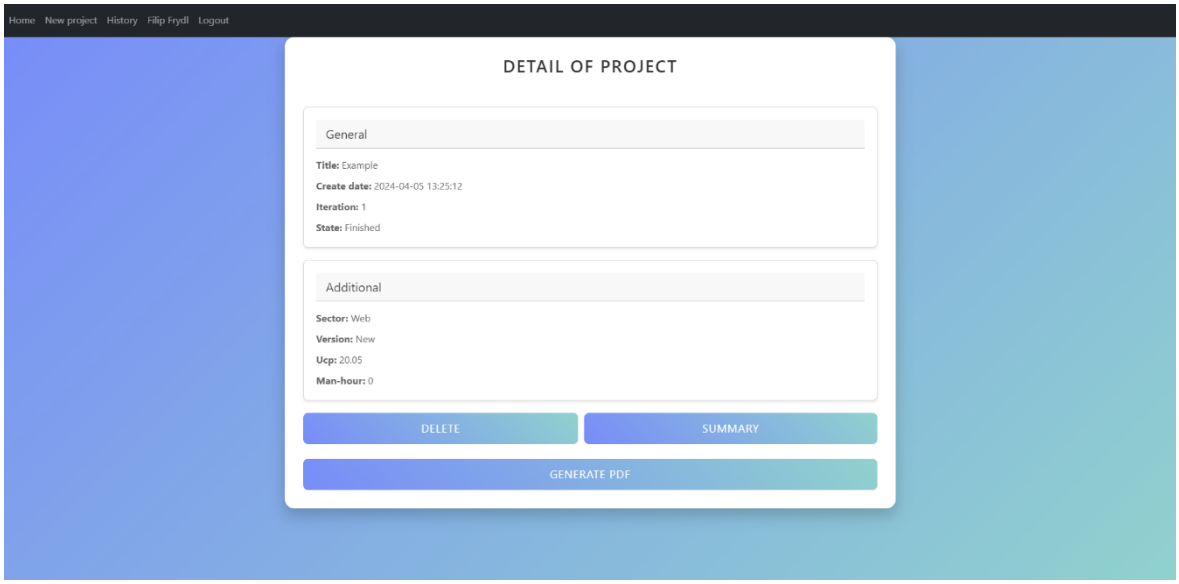
Obrázek 37 – Pohled pro historii projektů

Protože existuje možnost založení více iterací, uživatel má možnost zpětného přístupu na každou z nich, jako je na obrázku (Obr. 38). Založení nové iterace je možné pouze v případě, kdy je předchozí iterace stejného projektu dokončena. K tomu se na stránce nachází graf, který reflektuje rozdíly mezi hodnotami odhadu. V grafu jsou pouze dokončené odhady.



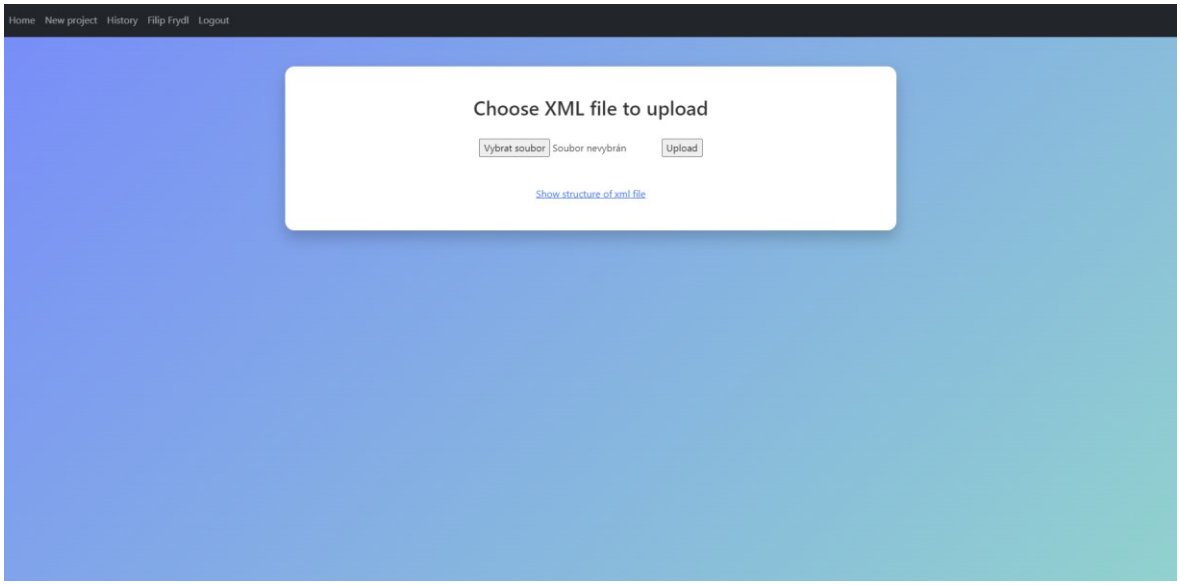
Obrázek 38 – Pohled pro přehled iterací projektu

Ještě před přehledem daného projektu je detailnější popis projektu, jako je na obrázku (Obr. 39). Z tohoto pohledu je možné se dostat na přehled celého projektu obsahující všechny vložené prvky. Zde je také možnost smazat danou iteraci projektu a nachází se tu i možnost generování souboru PDF. Obsah tohoto souboru je podobný, jako přehled celého projektu.



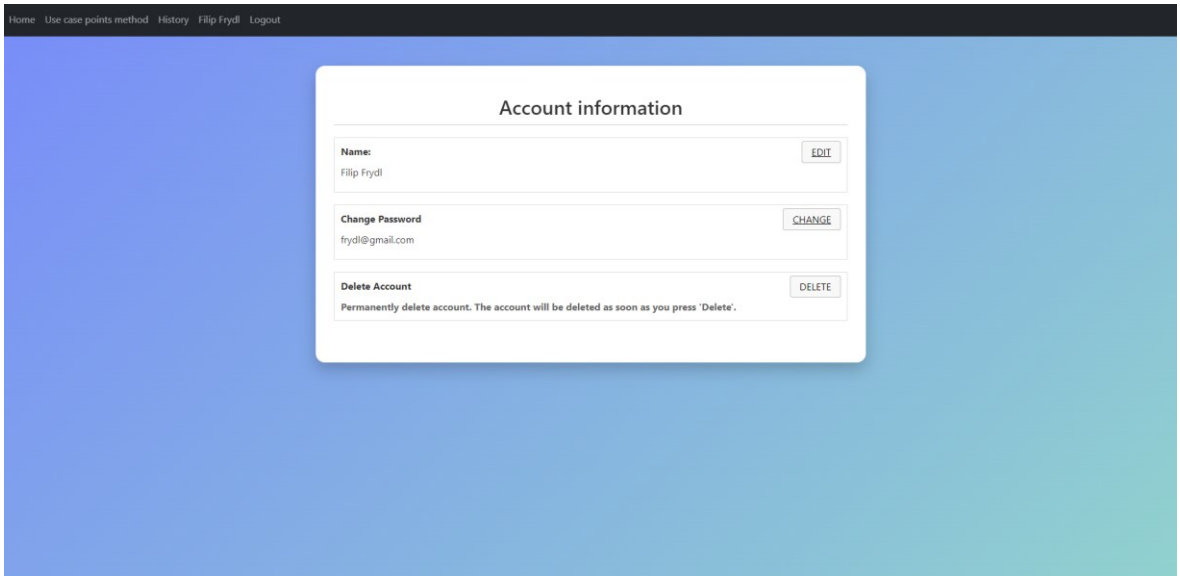
Obrázek 39 – Pohled pro detail projektu

Obrázek (Obr. 40) ukazuje pohled pro nahrání souboru. Aplikace je schopna zpracovat celý projekt, jenž je vložený pomocí XML souboru. Pro lepší představu, jak by měl vypadat soubor, je zde přidaná možnost zobrazit strukturu takového souboru.



Obrázek 40 – Pohled pro vložení souboru

Posledním pohledem, který stojí za zmínění, je pohled zobrazující informace o účtu přihlášeného uživatele. Ten je na obrázku (Obr. 41).



Obrázek 41 – Pohled pro přehled účtu uživatele

Zde se nacházejí tři sekce, a to sekce pro změnu osobních informací, pro změnu hesla a pro smazání účtu. Při použití prvních dvou sekcí se zobrazí již zmíněný pohled, jako u registračního formuláře a při zmáčknutí tlačítka *Delete* se zobrazí okno, zda uživatel opravdu chce vymazat účet. V případě, kdy uživatel klikne na tlačítko *Ano* vymaže se celý účet včetně všech přiřazených projektů a jednotlivých dat.

5.6 Příklad výpočtu metody

Poslední částí popisu implementace bude vzorové založení nového projektu včetně výpočtu odhadu a výpočtu hodnoty v člověkohodinách. Pro tento účel bude vytvořený náhodný projekt, který je inspirován ukázkovým projektem, kde jsou podobná číselná data [23]. Prvním krokem je vložení základních informací o projektu. Vstupy vložené do připravených polí jsou v tabulce (Tab. 12).

Tabulka 12 – Základní informace

Title	Example Project
Sector	School
Version	New
Productivity	20

Zde se zadávají základní informace, jako je název, odvětví projektu, zda je projekt nový nebo se jedná o rozšíření a průměrná produktivita na vykonání jednoho případu užití. Jak lze

v tabulce (Tab. 12) vidět, jedná se o nový projekt, který má název „Example Project“. Dalším krokem je zadání všech vystupujících aktérů do systému.

Tabulka 13 – Hodnoty aktérů v projektu

Name	Type	Description
One	User	Simple actor one
Two	User	Simple actor two
Three	User	Simple actor three
Four	User	Simple actor four
Five	Hardware	Simple actor five
Six	Hardware	Simple actor six
Seven	Hardware	Simple actor seven
Eight	Hardware	Simple actor eight
Nine	Internal system	Average actor one
Ten	Internal system	Average actor two
Eleven	Internal system	Average actor three
Twelve	Internal system	Average actor four
Thirteen	Internal system	Average actor five
Fourteen	Internal system	Average actor six
Fifteen	Software	Average actor seven
Sixteen	Software	Average actor eight
Seventeen	Software	Average actor nine
Eighteen	Software	Average actor ten
Nineteen	Software	Average actor eleven
Twenty	Software	Average actor twelve
Twenty-one	Stakeholders	Complex actor one
Twenty-two	Stakeholders	Complex actor two
Twenty-three	External system	Complex actor three
Twenty-four	External system	Complex actor four

Hodnoty, které se vloží do polí, jsou v tabulce (Tab. 13). Je zde jednoduše ukázáno kolik aktérů je v projektu s jejich typem a jednoduchým popisem, jenž naznačuje, jaké kategorie jsou pro výpočet odhadu. Rozdělení do těchto kategorií je podle typů aktéra. Dále se pomocí takového rozdělení spočítá hodnota uaw, jež reprezentuje sumu aktérů vynásobenou s váhami, tedy číselnou hodnotou zmíněných kategorií. Dalším krokem je vložení informací o případech užití. Všechny data jsou v tabulce (Tab. 14).

Tabulka 14 – Hodnoty případů užití v projektu

Name	Main actor	Steps	Description	Scenario
Uc one	One	1	Simple uc one	#

Uc two	Two	1	Simple uc two	#
Uc three	Three	1	Simple uc three	#
Uc four	Four	2	Simple uc four	##
Uc five	Five	2	Simple uc five	##
Uc six	Six	2	Simple uc six	##
Uc seven	Seven	3	Simple uc seven	###
Uc eight	Eight	3	Simple uc eight	###
Uc nine	Nine	4	Average uc one	####
Uc ten	Ten	4	Average uc two	####
Uc eleven	Eleven	4	Average uc three	####
Uc twelve	Twelve	5	Average uc four	#####
Uc thirteen	Thirteen	5	Average uc five	#####
Uc fourteen	Fourteen	5	Average uc six	#####
Uc fifteen	Fifteen	6	Average uc seven	######
Uc sixteen	Sixteen	6	Average uc eight	######
Uc seventeen	Seventeen	6	Average uc nine	######
Uc eighteen	Eighteen	7	Average uc ten	#######
Uc nineteen	Nineteen	7	Average uc eleven	#######
Uc twenty	Twenty	7	Average uc twelve	#######
Uc twenty-one	Twenty-one	8	Complex uc one	########
Uc twenty-two	Twenty-two	9	Complex uc two	#########
Uc twenty-three	Twenty-three	10	Complex uc three	##########
Uc twenty-four	Twenty-four	11	Complex uc four	###########

V této tabulce se nacházejí data pro případy užití, kde je znovu názorně ukázáno, kolik jich je a jaké budou kategorie pro výpočet. Zde se rozdělením vypočítá hodnota uucw, která je spočítána podobně jako u aktérů. Ve sloupci „Scenario“ reprezentují znaky „#“ jednotlivé kroky ve scénáři, a to kvůli přehlednosti. Ke každému případu užití je potřeba přiřadit jednoho aktéra, který je brán jako hlavní. Dále je potřeba vložit číselné hodnoty jednotlivých faktorů, a to technických faktorů i faktorů prostředí. Hodnoty technických faktorů jsou v tabulce (Tab. 15), kde jsou povolena čísla od 0 do 5.

Tabulka 15 – Hodnoty technických faktorů

Factor	Value
Distributed system	5
Performance objectives	4
End-user efficiency	2
Complex processing	4
Reusable code	2

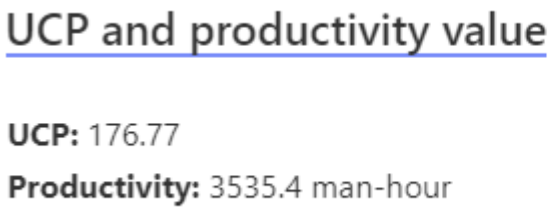
Easy to install	5
Easy to use	3
Portable	3
Easy to change	3
Concurrent use	2
Security	2
Access for third parties	5
Training needs	3

Pro každý faktor je přiřazená váha, která je důležitá pro výpočet technické složitosti. Pro výpočet složitosti prostředí je potřeba faktorů prostředí, jenž se nacházejí v tabulce 16.

Tabulka 16 – Hodnoty faktorů prostředí

Factor	Value
Familiar with the development process	4
Application experience	2
Object-oriented experience	5
Lead analyst capability	2
Motivation	1
Stable requirements	5
Part time staff	0
Difficult programming language	1

Všechna zmíněná data budou vložena do aplikace, kde proběhne odhad. Ke vkládání dat dochází na pohledech, které jsou na obrázcích (Obr. 32 – Obr.35).



Obrázek 42 – Výsledná hodnota odhadu

Na obrázku (Obr. 42) se nachází výsledný odhad vzorového příkladu při vložení ukázaných hodnot. První hodnota je celková hodnota UCP, ze které se později spočítá hodnota pracnosti v člověkohodinách. V aplikaci je výpočet uložen v databázi a je připraven pro založení nové iterace, kdy je uživatel schopný upravit, přidat nebo smazat jednotlivé prvky.

6 DALŠÍ ROZVOJ APLIKACE

Poslední kapitola bakalářské práce je zaměřena na zamyšlení nad budoucím rozvojem implementace nástroje pro odhad rozsahu software. Jak již bylo zmíněno, stále se jedná o prototyp, tedy o aplikace, do které by měla přibýt další funkčnost a další možnosti pro odhad software.

6.1 Přidání dalších metod

Hlavním rozšířením prototypu aplikace by mělo být přidání výpočtu odhadu pro další metody. Takovou metodou by mohla být metoda functional points, jež se hojně využívá pro základní odhad velikosti softwarového projektu. Její použití je vhodné hned po ukončení analýzy požadavků, tedy v době, kdy nejsou navrhnuty detailnější části, jako například model případů užití, který je potřeba pro metodu use case points. Tato metoda je schopna určit rozsah práce, což je užitečná informace pro začátek projektu. Další vhodnou metodou by mohla být metoda COCOMO, která patří mezi nejčastěji volené metody pro odhad software. Tato metoda se používá po získání detailnějších informací o projektu nebo i v počátečních fázích implementace. To znamená, že její použití je možné po vypracované metodě use case points. Všechny tyto metody jsou vhodné k použití na stejném projektu, a to v různých částech. Proto by možnost navázání byla příhodná. Uživatel by mohl začít první metodou a dostat se až do situace, kdy by byly použity všechny tři metody na stejném projektu a viděl by porovnání jakým způsobem pokračuje projekt. Samozřejmě existuje více metod, které by se daly přidat k vytvořenému prototypu, ale tyto dvě zmíněné by neměly chybět.

6.2 Rozšíření implementace

Přidáním nových metod vznikne potřeba rozšířit i zbytek aplikace. Založení nového projektu by mohlo vypadat stejně jako doposud, s tím rozdílem, že by si uživatel mohl vybrat pro jakou metodu nový projekt zakládá. To znamená, že by se úvodní stránka založení projektu rozšířila o několik dalších informací a byla by pro všechny projekty stejná. Dále by měla být přidaná možnost založení nové iterace s využitím starých dat z předešlé iterace. Metoda, která by vykonala novou iteraci, by si vzala jen taková data, která potřebuje a zbytek by musel uživatel dodat. Výhodou v takovém případě by byla větší přehlednost. Rozšířením by taky mělo projít generování souboru PDF. Systém by měl automaticky detekovat o jakou metodu se jedná a vygenerovat soubor připravený pro danou metodu. Hlavní změnou oproti prototypu by mělo být přidání jednotlivých metod. Všechny nově přidané metody by měly

vykazovat správnou funkčnost a správným způsobem vypočítat odhad. Změnou by mohla projít i historie projektů, kde by měla být možnost filtrování projektů podle určité informace nebo přidání vstupního pole pro vyhledávání projektů, například podle názvu.

6.3 Možnosti vložení souborů

Další částí, která by měla být rozšířena, je vkládání souborů. Momentálně aplikace dokáže zpracovat soubor XML, který má pevně danou strukturu. V budoucí verzi aplikace by bylo vhodné, aby aplikace dokázala zpracovat více možností. Jednou z možností by bylo vkládání souborů, jenž jsou vytvořeny v aplikaci sparx enterprise architect, která slouží k návrhu software. Tato aplikace dokáže exportovat jednotlivé diagramy. Aplikace by tedy měla být schopná zpracovat takový formát. Momentálně aplikace vyžaduje přesný formát souboru a není možné přidávat data do rozdělaného projektu, a proto by měla být implementována možnost přidávat do projektů jednotlivé prvky postupně. Díky této změně bude aplikace schopna zpracovat několik různých souborů za sebou do jednoho projektu, a to například několik zmíněných diagramů, ze kterých si vezme to důležité. Existují i další možnosti souborů, jako je například zpracování PDF souboru a další.

6.4 Sdílení projektů

Každý vytvořený projekt je přiřazený uživateli, který projekt vytvořil. To znamená, že všechna data včetně hodnoty odhadu jsou přístupná právě jednomu uživateli. Budoucí rozvoj by se mimo jiné mohl zaměřit i na sdílení projektů. Mohlo by to vypadat tak, že by uživatel nasdílel projekt jinému uživateli, který by ovšem musel mít založený účet. Tímto by došlo ke zjednodušení přenosu projektů mezi více uživateli. Uživatel, jenž dostal přístup k projektu, by mohl pouze sledovat data a nijak do něj nezasahovat. Popřípadě by mohl založit novou iteraci se stejnými daty, kterou by ovšem viděl pouze on sám. V případě, kdy by bylo potřeba, aby daný projekt spravovalo více lidí, by mohla být implementovaná možnost založení týmů. V takovém týmu by byl jeden hlavní uživatel spolu s dalšími uživateli a podle určitých rolí by mohli pracovat s projektem. Například leader týmu by mohl jako jediný zakládat projekty, vedoucí by mohl spravovat data a uživatel s nejmenší rolí by se mohl pouze volně pohybovat projektem. To by už záleželo na uživatelích v týmu. V historii všech projektů by mohla přibýt nová záložka „sdílené“ nebo „týmy“, kde by byly právě tyto projekty. Správa týmů nebo přidání ostatních uživatelů kvůli sdílení by mohlo probíhat na stránce spravující účet nebo by vznikla nová stránka právě pro tyhle potřeby.

ZÁVĚR

Tato bakalářská práce je zaměřena na vývoj nástroje pro odhad rozsahu software. Cílem bakalářské práce bylo navrhnout aplikace s následnou implementací prototypu pro vybranou metodu. Použitá metoda byla metoda use case points.

V teoretické části jsou představeny různé metody, které se běžně používají na odhad. Mezi popisovanými metodami byla i vybraná metoda, kde byl popsán přesný postup při řešení odhadu. To poskytlo lepší přehled o tom, jakým způsobem metoda funguje a kdy je vhodné ji použít.

Praktická část se dá rozdělit na tři samostatné části, a to část pro analýzu s návrhem aplikace, implementaci a budoucí rozvoj. V první části byly navrhnuté požadavky pro aplikaci, které poskytly první představu o tom, co by měl systém umět. Následně byly vytvořeny různé diagramy lépe popisující systém. Na vytvoření těchto diagramů byl použit grafický jazyk UML, jenž slouží právě pro tyto účely.

Druhá část praktické části je věnovaná implementaci navrhnuté aplikaci. Na vývoj aplikace byl použit programovací jazyk Python. Důležitou součástí implementace je správná konfigurace aplikace, která je popsána a kde jsou napsány základy určitých částí aplikace. Po konfiguraci je popsána hlavní funkčnost, kde je ukázán princip výpočtu odhadu i se zpracováním vstupů. Dále jsou popsány jednotlivé pohledy v aplikaci spolu s obrázkem. Poté je popsán ukázkový výpočet odhadu.

Poslední věcí v praktické části je zamyšlení nad budoucím rozvojem aplikace. Možný budoucí rozvoj se snaží o navržení budoucích prací, a to hlavně v samotné implementaci.

Výsledkem práce je vytvořený návrh a implementace prototypu aplikace, která zvládne vše, co je potřeba, aby byl odhad vykonán. Uživatel využívající aplikaci má možnost zpětného náhledu do odhadu včetně vygenerování souboru PDF.

SEZNAM POUŽITÉ LITERATURY

[1] POKORNÝ, Martin, 2009. Metody pro odhady náročnosti pracnosti při plánování projektů. Praha. Diplomová práce. Vysoká škola ekonomická v Praze, Fakulta informatiky a statistiky, Katedra informačních technologií. Vedoucí ráce Dušan Chlapek. Dostupné také z: https://vskp.vse.cz/19603_metody_pro_odhady_narocnosti_pracnosti_pri_planovani_projektu

[2] TRENDOWITZ, Adam a Ross JEFFERY. Software Project Effort Estimation. Cham: Springer International Publishing, 2014. ISBN 9783319036281.

[3] ŠILHAVÝ, Radek. *Softwarové inženýrství, analýza a modelování.* Přednášky pro předmět Analýza a modelování softwarových systémů. Online, prezentace. [cit. 2024-02-23]

[4] STRUSKA, Zdeněk a Robert PERGL. SROVNÁNÍ METOD COCOMO A ANALÝZY FUNCTION POINTS [online]. Praha [cit. 2024-02-25]. Dostupné z: http://www.agris.cz/Content/files/main_files/75/152840/142Struska.pdf. Článek. PEF ČZU Praha.

[5] SILHAVY, Radek; SILHAVY, Petr; PROKOPOVA, Zdenka. Analysis and selection of a regression model for the use case points method using a stepwise approach. Journal of Systems and Software, 2017, 125: 1-14.

[6] IFPUG Function Point Counting Rules. (2008). In *The IT Measurement Compendium: Estimating and Benchmarking Success with Functional Size Measurement* (pp. 453-482). Berlin, Heidelberg: Springer Berlin Heidelberg.

[7] BUNDSCHUH, M. AND C. DEKKERS The IT measurement compendium: estimating and benchmarking success with functional size measurement. Edtion ed. Berlin: Springer, 2008. xxxv, 643 p. p. ISBN 9783540681878

[8] ARLOW, Jim and NEUSTADT, Ila, 2001. UML and the Unified Process: Practical object-oriented analysis and design. Boston, MA: Addison Wesley. ISBN 9780201770605.

[9] UNHELKAR, Bhuvan, Software engineering with UML. Boca Raton: CRC Press: Auerbach Publications, c2006. ISBN 9781351235167.

[10] What is __pycache__ in Python? MYRIANTHOUS, Giorgos. Towards Data Science [online]. 2016, 2.2.2016 [cit. 2024-03-08]. Dostupné z: https://towardsdatascience.com/pycache-python-991424aabad8

[11] What is Python? Executive Summary. Www.python.org [online]. ©2001 [cit. 2024-03-10]. Dostupné z: https://www.python.org/doc/essays/blurb/

[12] Welcome to Flask. Online. Flask documentation (3.0.x). ©2010. Dostupné z: https://flask.palletsprojects.com/en/3.0.x/. [cit. 2024-03-11].

[13] ARLOW, J., & NEUSTADT, I. (2005). UML 2 and the unified process: practical object-oriented analysis and design: Pearson Education.

[14] SOMMERVILLE, I. (2015). *Software Engineering*: Pearson.

[15] PILONE, Dan, UML 2.0 pocket reference. Beijing; Farnham: O'Reilly, c2006. ISBN 0596102089.

[16] Installation. Online. Installation - Flask documentation (3.0.x). ©2010. Dostupné z: https://flask.palletsprojects.com/en/2.3.x/installation/. [cit. 2024-03-13].

[17] Configuration Handling. Online. Flask documentation (3.0.x). ©2010. Dostupné z: https://flask.palletsprojects.com/en/3.0.x/config/. [cit. 2024-03-14].

[18] Quickstart. Online. Flask documentation (3.0.x). ©2010. Dostupné z: https://flask.palletsprojects.com/en/3.0.x/quickstart/#routing. [cit. 2024-03-14].

[19] How it Works. Online. Flask-Login 0.7.0 documentation. Dostupné z: https://flask-login.readthedocs.io/en/latest/#how-it-works. [cit. 2024-03-21].

[20] Template Designer Documentation. Online. Jinja documentation (3.0.x). ©2007. Dostupné z: https://jinja.palletsprojects.com/en/3.0.x/templates/. [cit. 2024-03-29].

[21] Overview. Online. SQLAlchemy 2.0 documentation. 2024. Dostupné z: https://docs.sqlalchemy.org/en/20/intro.html. [cit. 2024-03-29].

[22] Configuration. Online. Flask-SQLAlchemy documentation (2.x). ©2010. Dostupné z: https://flask-sqlalchemy.palletsprojects.com/en/2.x/config/. [cit. 2024-03-29].

[23] Project Estimation with Use Case Points. Online. CodeProject. 2005. Dostupné z: https://www.codeproject.com/Articles/9913/Project-Estimation-with-Use-Case-Points. [cit. 2024-04-18].

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

UML	Unified modeling language
HTML	Hypetext markup language
CSS	Cascading style sheets
JS	JavaScript
URL	Uniform resource locator
ERD	Entity-relationship diagram
URI	Uniform resource identifier
XML	Extensible markup language

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM PŘÍLOH

PŘÍLOHA P I: WEBOVÁ APLIKACE

Jedná se o CD, kde se nachází vytvořená webová aplikace včetně zdrojového kódu.