ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Desktopový nástroj pro správu skupin lidí
Student:	Jakub Štercl
Vedoucí:	Ing. Jan Baier
Studijní program:	Informatika
Studijní obor:	Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2017/18

Pokyny pro vypracování

Proveďte analýzu požadavků a posléze navrhněte a implementujte desktopovou aplikaci řešící problém rozdělování lidí do skupin. Implementaci proveďte v libovolném jazyce, nutnou podmínkou je ale funkčnost aplikace na OS Linux. Celé řešení řádně otestujte a sepište uživatelskou příručku.

Aplikace musí umožňovat následující:
- uložit/editovat pojmenované skupiny lidí,
- rozdělit skupinu lidí do týmů,
- zobrazit pro každého člověka seznam lidí, se kterými již byl v týmu a kolikrát,
- vizualizovat rozdělení pro potřeby tisku,
- navrhnout takové rozdělení do týmů, které zohledňuje zadaná integritní omezení,
- při nemožnosti splnění všech omezení navrhnout možnost, která porušuje nejmenší počet omezení.

Možná omezení:
- minimální/maximální počet lidí v týmu,
- minimální/maximální počet lidí, kteří již byli ve stejném týmu,
- vynucení/zákaz konkrétních dvojic.

Seznam odborné literatury

Dodá vedoucí práce.

L.S.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 25. ledna 2017

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra softwarového inženýrství

Bakalářská práce

Desktopový nástroj pro správu skupin lidí

Jakub Štercl

Vedoucí práce: Ing. Jan Baier

15. května 2017

Poděkování

Chtěl bych poděkovat vedoucímu práce Ing. Janu Baierovi za jeho cenné rady, poznatky a připomínky.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 15. května 2017 .....

Odkaz na tuto práci

Štercl, Jakub. *Desktopový nástroj pro správu skupin lidí*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2017.

Abstrakt

Tato práce se zabývá vývojem desktopové aplikace pro správu skupin lidí a jejich rozdělování do týmů, která vychází z požadavků organizátorů soustředění FIKS. Součástí práce je celý cyklus vývoje, od analýzy požadavků, přes návrh, implementaci až po testování. Výsledkem práce je desktopová aplikace v jazyce Python využívající framework Qt, která dokáže podle zadaných požadavků navrhnout rozdělení lidí do týmů.

Klíčová slova Desktopová aplikace, Python, Qt, správa skupin, rozdělování do týmů

Abstract

This work covers development of desktop application for management of groups of people and splitting them into teams based on requirements of FIKS training camp organizers. The work describes the whole development cycle, starting from requirements analysis, through application design and implementation, to testing. The result of this work is desktop application, written in Python programming language using Qt framework, which can propose how to split a group of people into teams, based on given requirements.

Keywords Desktop application, Python, Qt, group management, splitting into teams

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Moderní technologie a výpočetní technika dokáže nejen usnadňovat život, ale rozhodovat i rychleji, lépe nebo třeba spravedlivěji než člověk. Velmi dobře se uplatní při řešení úkolu, před který jsou mnohdy postaveni učitelé, organizátoři různých akcí nebo projektoví vedoucí a to rozdělování lidí (žáků, účastníků, apod.) do týmů. Nejde-li pouze o početní rozdělení, ale i splnění určitých požadavků na sestavení týmu, může jít o složitější problém. Jelikož neexistuje uspokojivý nástroj, který by pomáhal tento úkol řešit, rozhodl jsem se jej v rámci své bakalářské práce vytvořit.

Kapitola 1

Cíl práce

Cílem práce je na základě analýzy požadavků navrhnout a posléze naimplementovat aplikaci, která bude organizátorům soustředění korespondenčního semináře FIKS (Fiťácký informatický korespondenční seminář)[1] sloužit při tvorbě týmů pro jednotlivé hry. Hlavní motivací, proč organizátoři takovouto aplikaci chtějí je, aby se účastníci mezi sebou poznali, tj. každou hru hráli, pokud možno, s novými spoluhráči. Velmi dobře může být využitelná třeba i při organizaci firemních teambuildingových akcí, které mají podobnou motivaci.

1.1 Existující řešení

Pro rozdělování skupin lidí do týmů samozřejmě existují různé nástroje, většinou se jedná buď o webové nebo mobilní aplikace. Jednou z takových webových aplikací je například Team Maker[2], podobná mobilní aplikace je Teamistr[3]. Tyto aplikace ale neumožňují vytvoření a uložení seznamů lidí (např. paralelky), tudíž neukládají ani historii vytvořených týmů, neumožňují zobrazení, s kým už daná osoba ve skupině byla a neumožňují ani zvolit, zda preferujeme rozdělení, kde lidé v jednotlivých týmech spolu již spolupracovali, nebo naopak. Proto nejsou zdaleka dostačující a jejich jediný účel je rozdělovat skupiny do týmů „spravedlivě“.

1.2 Analýza požadavků

Prvním krokem jakéhokoli úspěšného softwarového projektu je pochopit, co od výsledného systému zákazník (zadavatel) chce, co od něj požaduje. K tomu slouží tzv. analýza požadavků (requirements analysis).

Nejdůležitější částí analýzy požadavků je samotný sběr požadavků, který probíhá spoluprací se zástupci zadavatele. V ideálním případě se do diskuse zapojují zástupci všech skupin budoucích uživatelů. Například není vhodné, aby u systému, který budou využívat žáci i učitelé, proběhla diskuse o požadavcích pouze se zástupci učitelů. Mohlo by se totiž stát, že ačkoli výsledný systém splňuje všechny vytyčené požadavky, nebude systém pro žáky použitelný.

1.2.1 Funkční požadavky

Požadavky sebrané v analýze (příp. sběru) požadavků se nejčastěji dělí na funkční a nefunkční požadavky. Samozřejmě existuje mnoho jiných způsobů jejich kategorizace, ale pro tento projekt je toto rozdělení naprosto dostačující. Funkčním požadavkem je popis požadované funkce systému – popis toho, co by měl systém umět.[4]

Na základě diskuse se zadavatelem jsem dospěl k následujícím funkčním požadavkům:

- **Správa skupin** – Aplikace bude umožňovat vytvořit, pojmenovat a upravovat skupiny lidí. Úpravou skupiny se myslí možnost přidávat členy, případně členy odebírat a přejmenovávat. Členové jsou identifikováni jménem a skupinou.

- **Rozdělení skupiny do týmů** – Aplikace bude umět navrhnout rozdělení skupiny do týmů podle zadaných parametrů. Těmito parametry může být minimální/maximální počet lidí v týmu, počet týmů a vynucení/zakázání určitých dvojic (nebo i větších skupin) lidí ve stejném týmu. Dále bude aplikace umožňovat preferovat rozdělení na týmy, ve kterých lidé, již spolu byli, či naopak. V případě, že požadované rozdělení neexistuje, aplikace navrhne takové rozdělení, které porušuje co nejméně požadavků.

- **Tisk rozdělení** – Aplikace bude umět vytvořené rozdělení vizualizovat pro potřeby tisku.

- **Export rozdělení** – Aplikace bude umět vytvořené rozdělení exportovat do textového souboru. Soubor se bude skládat z jednotlivých týmů, každý na samostatné řádce. Každá řádka bude začínat jménem týmu, následovaném jmény členů týmu, oddělenými čárkou.

- **Zobrazení historie rozdělení** – Aplikace bude umět pro každého člena skupiny umět zobrazit, s kterými členy skupiny již v týmu byl

a kolikrát. Zároveň bude aplikace umět pro každou skupinu zobrazit uložená rozdělení.

1.2.2 Nefunkční požadavky

Jako nefunkční požadavky označujeme všechny požadavky, které přímo neříkají, co má systém umět. Nefunkční požadavek je v podstatě omezující nebo upřesňující podmínka uvalená na daný systém.[4]

Pro tento projekt jsem identifikoval tyto nefunkční požadavky:

- **OS Linux** – Aplikace musí fungovat na operačním systému Linux.
- **Velikost skupiny** – Aplikace bude využívána pro skupiny o velikosti přibližně 24 členů.

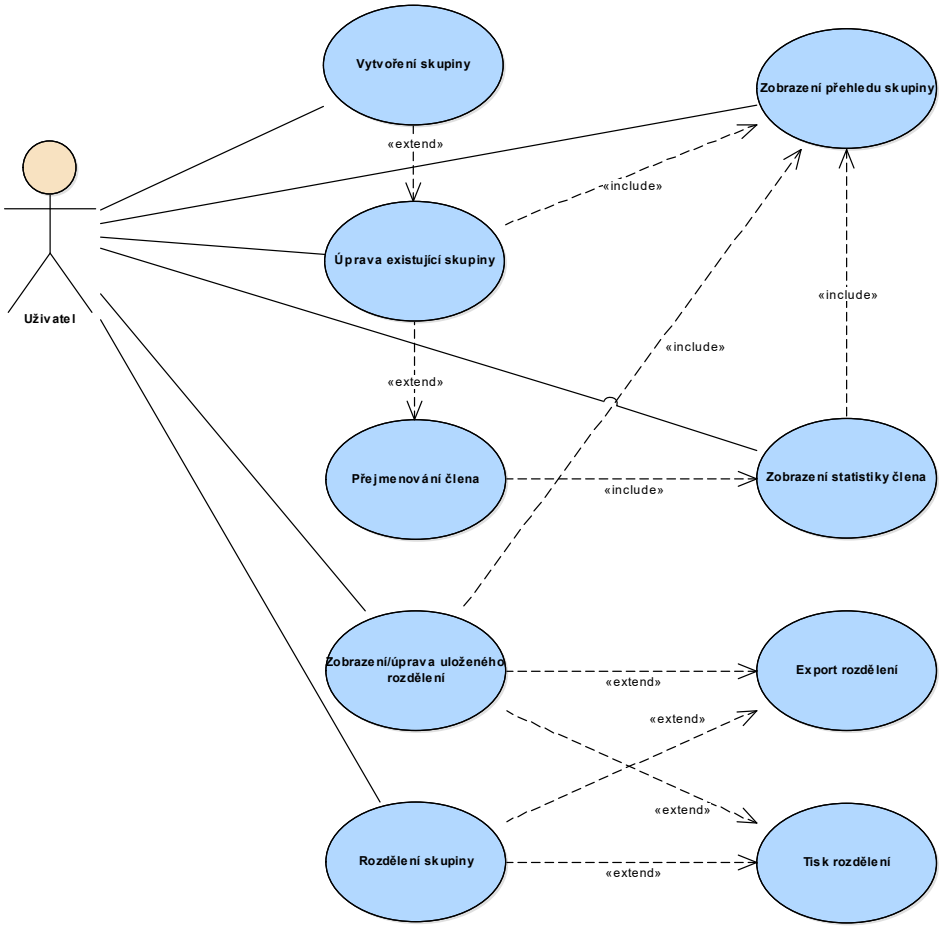
O validitě posledního nefunkčního požadavku (*Velikost skupiny*) by se dalo polemizovat, jelikož se vlastně nejedná o požadavek, nýbrž o informaci o tom, jak bude aplikace používána. Přesto jsem se rozhodl začlenit tuto cennou informaci přímo do požadavků, jelikož takováto informace bude hrát roli při návrhu uživatelského rozhraní, testování a podobně.

1.2.3 Případy užití

Případy užití přesně zachycují funkčnost, která bude budoucím systémem pokryta a vymezují tak jednoznačně rozsah prací. Každý případ užití popisuje jeden ze způsobů použití systému, popisuje tedy jednu jeho požadovanou funkčnost.[5]

Případ užití je vlastně seznam kroků, který definuje jednu konkrétní interakci uživatele se systémem. Ačkoli struktura případu užití (popis, jak by vlastně takový případ užití měl vypadat) není standardizována, tak UML nabízí tzv. diagram užití (use case diagram). Jedná se vlastně o přiřazení jednotlivých činností k aktérům a případné znázornění souvislostí jednotlivých případů užití (relace *«extend»* a *«include»*). Samotný scénář případu užití (posloupnost jednotlivých kroků potřebných pro vykonání případu užití) už není součástí diagramu.

Pro tento projekt jsem dospěl k následujícím případům užití znázorněným v diagramu 1.1. Dále jsou uvedeny scénáře případů užití.



Obrázek 1.1: Diagram případů užití

- **Vytvoření skupiny**

  1. Uživatel stiskne tlačítko *Vytvořit skupinu*.
  2. Aplikace zobrazí přehled nově vytvořené skupiny s automaticky přiděleným jménem.

- **Zobrazení přehledu skupiny**

  1. Uživatel vybere ze seznamu skupinu.
  2. Aplikace zobrazí obrazovku s přehledem členů, s možností přepnutí na přehled uložených rozdělení.

- **Úprava existující skupiny**

  1. Případ užití *Zobrazení přehledu skupiny.*
  2. Uživatel přidá člena, vybere ze seznamu člena, kterého chce ze skupiny odebrat, nebo skupinu přejmenuje, příp. smaže.
  3. Aplikace zkontroluje, zda je takováto změna možná a změnu uloží nebo zobrazí chybovou hlášku.

- **Zobrazení statistiky člena**

  1. Případ užití *Zobrazení přehledu skupiny.*
  2. Uživatel vybere ze seznamu člena.
  3. Aplikace zobrazí obrazovku s přehledem člena.

- **Přejmenování člena**

  1. Případ užití *Zobrazení statistiky člena.*
  2. Uživatel stiskne tlačítko pro úpravu jména člena.
  3. Uživatel vyplní nové jméno.
  4. Aplikace zkontroluje unikátnost jména ve skupině a jméno uloží nebo zobrazí chybovou hlášku.

- **Rozdělení skupiny**

  1. Uživatel vybere ze seznamu skupinu, kterou si přeje rozdělit.
  2. Uživatel nastaví parametry rozdělení (omezení velikostí týmů, vynucení/zakázání dvojic, apod.)
  3. Aplikace vytvoří rozdělení a zobrazí jej uživateli.
  4. Uživatel může rozdělení upravit, případně uložit.

- **Zobrazení/úprava uloženého rozdělení**.

  1. Případ užití *Zobrazení přehledu skupiny*
  2. Uživatel vybere ze seznamu rozdělení.
  3. Aplikace zobrazí uživateli rozdělení.
  4. Uživatel rozdělení upraví a uloží, smaže nebo ponechá.

- **Export rozdělení**
  1. Případ užití *Zobrazení/úprava uloženého rozdělení*, nebo případ užití *Rozdělení skupiny*.
  2. Uživatel zvolí *Export*.
  3. Uživatel zvolí soubor, do kterého chce skupinu exportovat.
  4. Aplikace provede export do zvoleného souboru.
- **Tisk rozdělení**
  1. Případ užití *Zobrazení/úprava uloženého rozdělení*, nebo případ užití *Rozdělení skupiny*.
  2. Uživatel zvolí *Tisk*.
  3. Aplikace převede rozdělení do tisknutelné podoby a provede tisk.

Po vytvoření případů užití se doporučuje namapovat je na funkční požadavky, a to zejména z kontrolních důvodů, abychom odhalili případné nesrovnalosti.

	Funkční požadavky				
Případy užití	Správa skupin	Rozdělení sk. do týmů	Tisk rozdělení	Export rozdělení	Zobrazení historie rozdělení
Vytvoření skupiny	√				
Zobrazení přehledu skupiny	√				
Úprava existující skupiny	√				
Zobrazení statistiky člena					√
Přejmenování člena	√				
Rozdělení skupiny		√			
Zobrazení/úprava uloženého rozdělení					√
Export rozdělení				√	
Tisk rozdělení			√		

Tabulka 1.1: pokrytí funkčních požadavků

Kapitola 2

Návrh řešení

2.1 Volba technologií

Logika a struktura jakékoli aplikace je úzce spjata s výběrem technologií, pomocí kterých je vystavěna. Vzhledem k tomu, že v zadání není určeno jaké technologie má implementace využívat, bude nutné je zvolit.

Pro tvorbu desktopových aplikací se nabízí celá řada možností a jazyků. Vždy je ovšem potřeba myslet na to, že bude nutné vytvořit nejen logickou část aplikace, ale také GUI – uživatelské rozhraní, proto musíme zvolit také technologii pro jeho tvorbu.

2.1.1 Volba programovacího jazyka

Jedním z nejpoužívanějších jazyků pro tvorbu desktopových aplikací je jednoznačně C++[6]. Mezi jeho výhody patří vysoká efektivita vedoucí k rychlejším běhům programů. To je ovšem vykoupeno větší náročností pro programátora, jelikož se musí starat o uvolňování a alokování paměti apod. Proto se podle mého názoru nehodí pro tento projekt.

Další možností by bylo využití jazyku Java a tvorba GUI v JavaFX. Java řeší mnoho nevýhod C++ například integrovaným garbage collectorem, který se stará o uvolňování paměti. To s sebou samozřejmě přináší cenu v podobě rychlosti.[7] Java se ovšem v poslední době přesunuje spíše na servery a vývoj GUI v JavaFX je poměrně náročný. Další nevýhodou Javy je, pro tento projekt, nutnost instalace Java virtual machine (JVM) na uživatelově stroji. JVM sice nabízí naprostou přenositelnost mezi platformami, to ale není pro tento projekt vůbec přínosem, jelikož cílem projektu je vytvořit aplikaci čistě pro operační systém Linux.

Nakonec jsem se rozhodl pro programovací jazyk Python, který umožňuje velmi snadný vývoj aplikací, jelikož nabízí řadu praktických konstruktů, které ulehčují práci (jako např. *for-else*).

2.1.2 Python

Python je na rozdíl od obou dříve zmíněných jazyků tzv. dynamicky typovaný (někdy také jen dynamický) jazyk. Dynamicky typované jazyky se liší od staticky typovaných jazyků tím, že proměnná není vázána na typ, jen na objekt. To prakticky znamená, že v průběhu programu se v jedné proměnné může vystřídat více typů (např. integer a string).[8] Tato vlastnost může, ale nemusí, některé operace a úkony usnadnit a v některých případech také značně zlepšit čitelnost kódu. Na druhou stranu může docházet k chybám za běhu, které by se u staticky typovaných jazyků projevily již při kompilaci.

U Pythonu je důležité, jakou verzi budeme používat, jelikož na konci roku 2008[9] byla vydána verze 3.0, která již není zpětně kompatibilní. To samozřejmě vedlo k částečnému rozdělení Python komunity, protože někteří programátoři musí stále pracovat se starší verzí Pythonu, například protože rozšiřují již existující aplikaci nebo pracují s knihovnou, která operuje se starší verzí Pythonu, a také se vždy najdou tací, jímž se změny nelíbí a proto budou pokračovat s prací se starší verzí. Dle ankety z roku 2014[10] by téměř polovina dotazovaných započala nový projekt v Pythonu verze 2.x, spíše než v některé z 3.x verzí. Zároveň anketa ukazuje, že nejpoužívanější verzí Pythonu byla verze 2.7, následovaná v té době aktuální, verzí 3.4. V anketě také můžeme vidět, že nejčastějším (59%) důvodem pro setrvání v Pythonu 2.x byly závislosti na různých knihovnách a balíčcích, z kterých mnoho není do dnešní doby aktualizováno do Pythonu 3. Předpokládám, že kdyby anketa proběhla v dnešní době, tato čísla a celkové vyznění ankety, by se příliš nezměnilo. Jenom nejpoužívanějším Pythonem 3.x by snad byl nejnovější Python 3.6.

Pro tento projekt jsem zvolil Python 3.6, jelikož se starším Pythonem nemám zkušenost a ani nepotřebuji využívat žádnou knihovnu, která by nebyla k dispozici pro tuto verzi. Jediná nevýhoda verze 3.6, tentokrát oproti starší verzi 3.4 je absence knihovny PySide, která slouží jako „vylepšení“ knihovny PyQt, zprostředkovávající napojení Pythonu na GUI framework Qt. Výhody PySide jsou ale, podle mého názoru, téměř zanedbatelné, proto mi pro tento projekt plně postačila „jen“ knihovna PyQt5.

2.1.3 GUI

Jak jsem již naznačil v předchozí sekci, pro tvorbu GUI jsem se rozhodl využít knihovnu PyQt5 a tudíž i framework Qt.

Qt je multiplatformní framework pro vytváření uživatelského rozhraní. Ačkoli je to knihovna pro C++, existují implementace pro různé jazyky jako například Python, Ruby, C# a další. Qt podporuje lokalizaci, správu vláken, nabízí předpřipravená řešení pro časté případy (dialogy pro otevírání souborů, tisk apod.) a přejímá nativní vzhled operačního systému. Mezi další výhody patří velmi dobrá a přehledná dokumentace[11] a kvalitní vývojové programy jako QtCreator pro návrh, nebo Linguist pro překlad. Aktuální verze Qt je verze 5.8.

2.1.4 Persistence dat

Poslední důležité rozhodnutí byla volba způsobu zachování dat mezi jednotlivými běhy aplikace – persistence dat. Vzhledem k povaze projektu se jedná o zápis na pevný disk uživatelova počítače a nabízí se v podstatě dvě možnosti, serializace nebo databáze.

Serializace označuje proces, při kterém je instance objektu nějakým způsobem převedena do uložitelné podoby. Zároveň může být cílem serializace také minimalizace velikosti dat, z důvodu úspory místa na disku, nebo rychlejšího přenosu po síti.[12] Opačným procesem je deserializace, čili získání těchto dat zpět. Python nabízí pro serializaci řadu modulů, například modul *pickle*.[13]

Jako databáze se dá využít SQLite – knihovna napsaná v jazyce C, která poskytuje jednoduchou databázi umístěnou na pevném disku. Pro Python dostupná například pomocí modulu *sqlite3*.[14] Přístup k datům v databázi je realizován pomocí SQL dotazů.

Pro tento projekt jsem zvolil databázový přístup, jelikož relační databáze ze své podstaty usnadňuje vyhledávání dat (pomocí dotazů), dále také není při každém běhu programu nutné načíst všechna uložená data do paměti ve formě objektů Pythonu, což by při použití serializace nejspíše bylo nutné.

2.2 Doménový model

Doménový model slouží k pochopení vztahů, konceptů a objektů v cílové bussiness doméně (oblasti). Snaží se dekomponovat doménu do objektů a vztahů mezi nimi.[15] Přestože je doménový model nezávislý na platformě

a objekty v tomto modelu nejsou softwarové objekty, je doménový model také základem pro design softwaru.

Doménový model 2.1 pro tento projekt je celkem jednoduchý, obsahuje pouze čtyři logické objekty:

- **Group** – skupina

  Skupina je vlastně agregace (sjednocení) členů. Každá skupina má jméno.

- **Member** – člen

  Člen má jméno a patří právě do jedné skupiny.

- **Distribution** – rozdělení

  Rozdělení symbolizuje výsledek procesu rozdělení skupiny do týmů, je to agregace týmů. Rozdělení se váže právě k jedné skupině. Zároveň má také jméno (kvůli uložení).

- **Team** – tým

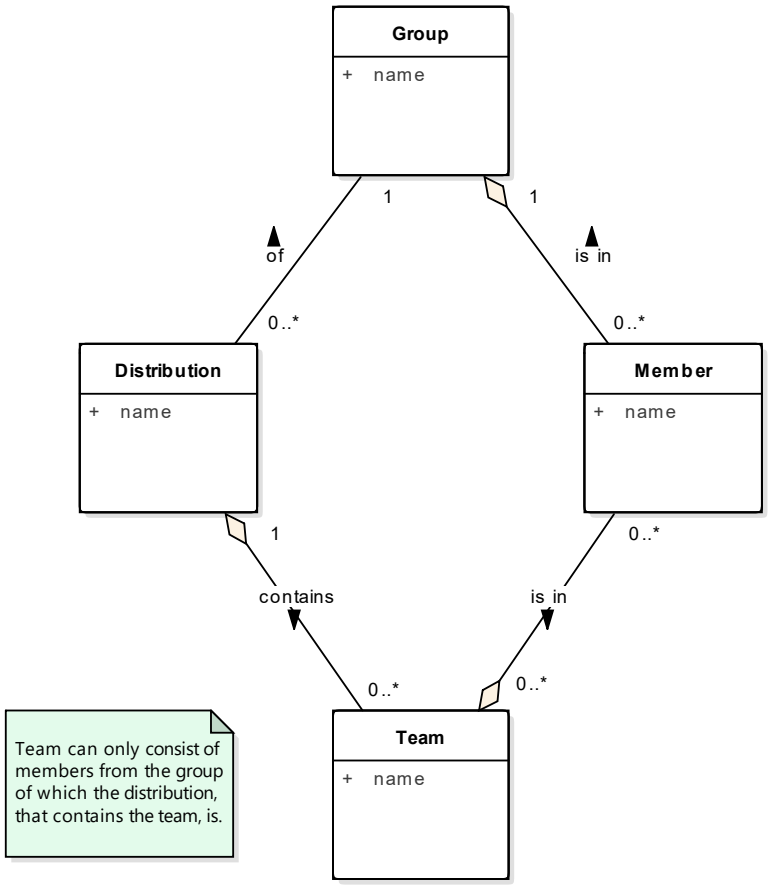
  Tým je agregace členů, jak můžeme vidět na poznámce v diagramu 2.1, tým může obsahovat pouze členy ze skupiny svého rozdělení. To znamená, že pokud rozdělení *dist* patří (rozděluje) skupinu A, pak každý tým z rozdělení *dist* může obsahovat pouze členy skupiny A a žádné jiné. Tým má opět své jméno, aby uživatel mohl rozlišit týmy v rozdělení.

2.3 Proces rozdělení skupiny

Nejdůležitějším a zároveň také nejkomplikovanějším procesem v aplikaci je vytvoření rozdělení, tj. rozdělení skupiny do týmů. Proto jsem se rozhodl při návrhu jej popsat a ujasnit, jak bude vlastně takové rozdělení probíhat. Pro popis tohoto procesu jsem využil diagram aktivit (activity diagram).

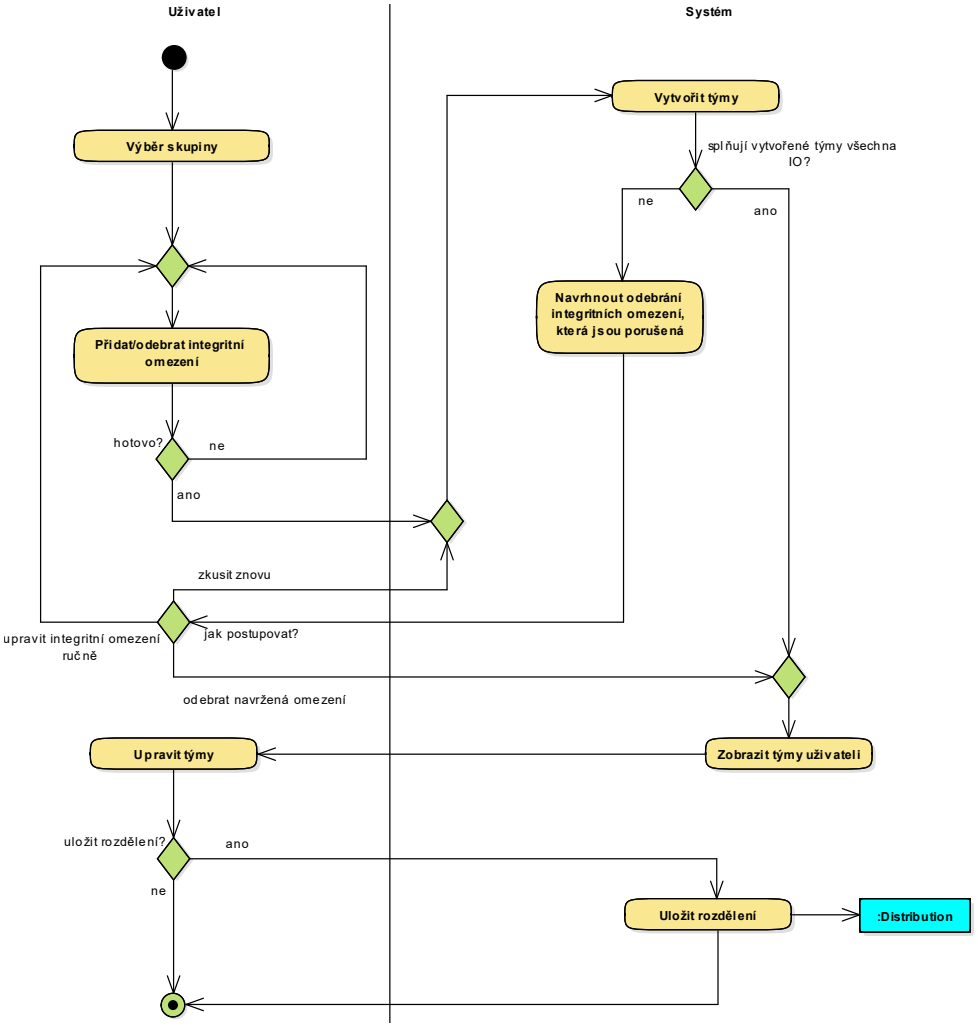
Diagram aktivit slouží právě k popisu procesů. Ať už se jedná o bussiness procesy nebo procesy přímo v systému, obojí můžeme snadno a přehledně popsat právě diagramem aktivit. Dle [4] „nabízí univerzální mechanismus pro modelování chování, který můžeme využít, kdykoli se nám to bude hodit“ a právě při popis procesu rozdělení skupiny se tento diagram náramně *hodí*.

Při pohledu na diagram 2.2 může vyvstat otázka, proč, pokud systém nenajde vhodné rozdělení do týmů, má uživatel možnost hledání opakovat se stejnými parametry a proč by to chtěl udělat. Tuto možnost jsem do



Obrázek 2.1: Doménový model

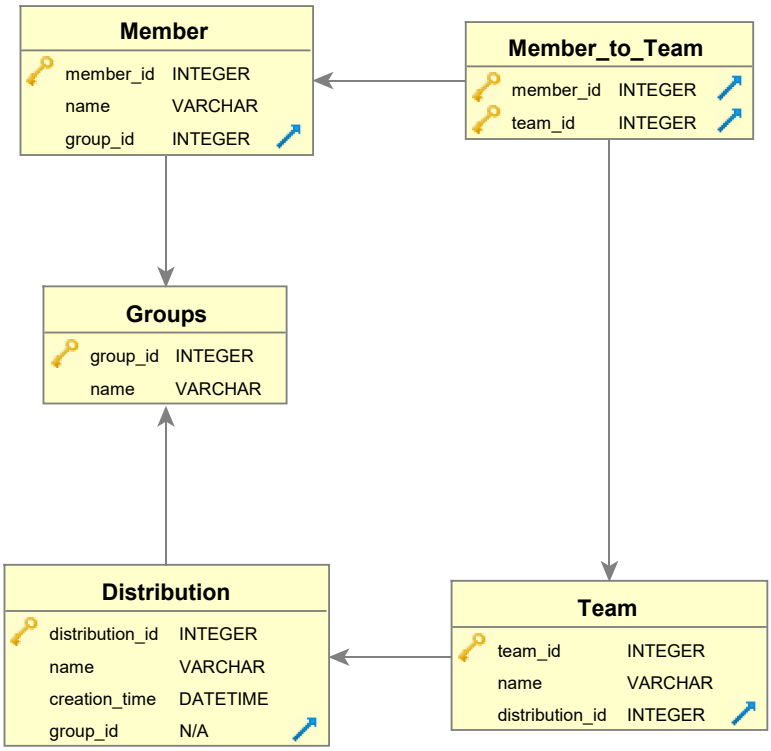
diagramu, a také do aplikace, doplnil až v průběhu implementace a souvisí to se způsobem (algoritmem), který se používá pro nalezení vyhovujícího rozdělení. Více o tomto algoritmu v části 3.5.1.



Obrázek 2.2: Diagram aktivit znázorňující proces vytvoření rozdělení

2.4 Datový model databáze

Na obrázku 2.3 můžeme vidět datový model databáze, model vychází z bussiness doménového modelu 2.1. Tabulka *Member_to_team* slouží k zachycení vztahu *M:N*, dále bych chtěl vypíchnout, že tabulka *Groups* má jméno v množném čísle, abych se vyhnul střetu s klíčovým slovem v SQL *GROUP*[16].



Obrázek 2.3: Datový model databáze

Kapitola 3

Implementace

3.1 Verzovací systém

Verzovací systém je systém, který zaznamenává změny souboru, nebo sady souborů, v průběhu času. Uživatel tak může kdykoli obnovit jeho (jejich) konkrétní verzi (tzv. verzování).[17] Většinou mluvíme o tzv. centralizovaných systémech správy verzí, které řeší také problém spolupráce s dalšími pracovníky.

V tomto případě jsem používal verzovací systém Git[18], konkrétně repozitář hostovaný na fakultním serveru gitlab[19], hlavně kvůli možnosti vrátit zpět změny, které způsobily problémy a možnosti práce na více zařízeních. Repozitář obsahující zdrojový kód, analýzu a také tuto práci je k dispozici na `https://gitlab.fit.cvut.cz/stercjak/BP`.

3.2 Struktura projektu

3.2.1 Architektura

Model-View-Controller (MVC) je jeden z nejpoužívanějších a nejobecnějších architektonických vzorů, existuje také mnoho variant tohoto návrhového vzoru, které se zpravidla liší jen v některém dílčím paradigmatu.[20] Dle [21] je MVC trojice objektů, model, aplikační objekt, view, jeho reprezentace na obrazovce a controller, který definuje způsob, jakým uživatelské rozhraní reaguje na vstup od uživatele. Pokud spojíme view a controller, dostaneme tzv. *model/view architekturu*. Tato architektura stále odděluje způsob, jakým jsou data uchovávána a způsob, jakým jsou data zobrazována uživateli. Toto rozdělení umožňuje zobrazování stejných dat ve více

pohledech, na více obrazovkách, ale spojuje zpracování vstupů od uživatele a definici obrazovky.[22]

Tím, že framework Qt využívá právě model/view architekturu, je určena i architektura celého projektu. Při pohledu do zdrojových kódů, ovšem můžeme vidět, že se v projektu nachází i něco, co se jmenuje *controller*, není to ale controller ve smyslu MVC architektury, nýbrž se opět jedná o spojení controlleru a view. Hlavním úkolem tříd typu *controller* je ovládání jednotlivé obrazovky, zároveň se ale také mnohdy podílí na její tvorbě (definici), například vkládáním tabulky.

Každá třída typu controller využívá vícenásobné dědičnosti, kdy jednou z rodičovských tříd je třída *Ui_Form* z příslušného modulu (souboru) a druhou je třída *QWidget* nebo některá z jejích podtříd. Díky tomu, že controller je potomkem třídy *QWidget* lze každou jednotlivou obrazovku vložit do *QStackedWidget*u v hlavním okně aplikace. Obsluhou tohoto widgetu je realizována změna obrazovek. Podrobnosti o těchto třídách jsou k dispozici v oficiální dokumentaci Qt[11]. Další rodičovskou třídou je třída *Ui_Form*, nabízející metodu *setupUi*, která instanci třídy nadefinuje rozložení obsahu na obrazovce, jako například umístění tlačítek apod. Více o této třídě zmiňuji v části 3.3.

```
├── controllers ..... třídy typu controller
├── model ..... vlastní model (třídy, které se starají o backend logiku)
├── resources
│   └── icons ..... ikony použité v aplikaci
├── translations ..... soubory pro překlad
├── user_guide_sources ..... zdrojové soubory pro uživatelskou příručku
├── utils ..... utility, jako např. třída DistributionMaker
│   ├── qtmodels ..... modely pro různé typy QView (třídy implementující
│   │                  QAbstractModel)
│   └── widgets ..... vlastní widgety (např tabulka s tlačítky)
├── windows ..... definice různých obrazovek (třídy Ui_Form)
│   └── qt ..... projekt QTCreatoru a soubory .ui
├── grou.py ..... hlavní spustitelný skript
└── user_guide.pdf ..... uživatleská příručka
```

Ukázka 3.1: adresářová struktura projektu

3.2.2 Adresářová struktura

Vzhledem k zvoleným technologiím (Python a Qt) není žádná struktura vynucována např. frameworkem, přesto je ale více než vhodné nějakou strukturu dodržovat kvůli přehlednosti.

Na ukázce 3.1 je naznačena struktura, kterou jsem pro tento projekt dodržoval.

3.3 Postup tvorby obrazovek

Při tvorbě jednotlivých obrazovek aplikace jsem postupoval následovně:

1. Obrazovku jsem nejprve navrhnul v drag and drop designéru *Qt Creator*. Výstupem tohoto programu je xml soubor *.ui*.

2. Xml soubor *.ui* jsem pomocí utility *pyuic5*[23] převedl na Python modul.

3. V controlleru příslušné obrazovky jsem naimportoval modul z kroku 2 a dle vzoru tři v [23] jsem využil vícenásobnou dědičnost.

U tohoto přístupu je důležité, aby controller ve svém konstruktoru volal nejprve konstruktor předka (*QWidget*, nebo některý z jeho potomků) a poté metodu *setupUi* s parametrem *self*. Tato metoda zajistí, že se do controlleru vloží všechny požadované prvky se správnými parametry (jako např. tlačítko se správným popisem a umístěním na obrazovce). Následuje ukázka přímo z kódu v aplikaci.

```
class Ui_Form(object):
  def setupUi(self, Form):
    Form.setObjectName("Form")
    Form.resize(622, 526)
    self.verticalLayout_2 = QtWidgets.QVBoxLayout(Form)
    self.verticalLayout_2.setObjectName("verticalLayout_2")
    self.verticalLayout_3 = QtWidgets.QVBoxLayout()
    self.verticalLayout_3.setSpacing(0)
```

Ukázka 3.2: část automaticky generované třídy Ui_Form

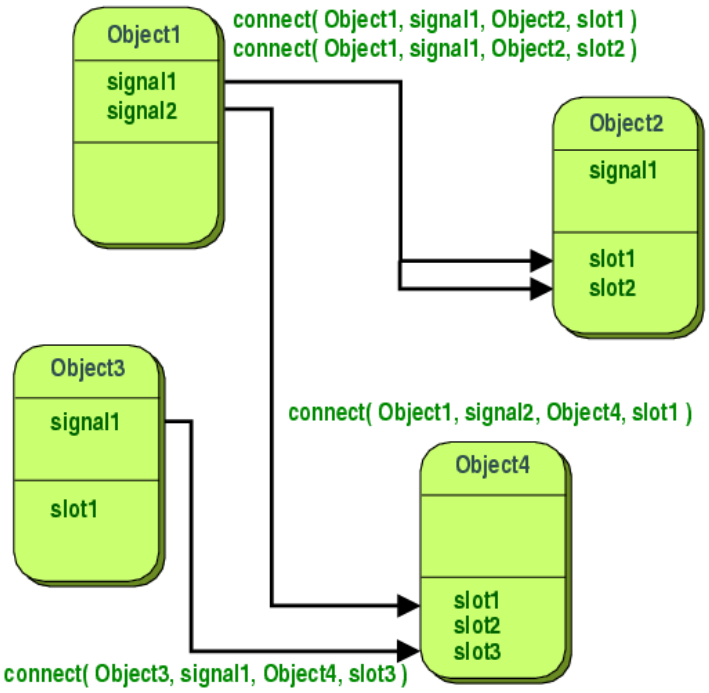
```
class DistributionSetup(BaseController, QWidget,
↪  distributionsetup.Ui_Form):
def __init__(self, main_window, group, parent=None):
 super(DistributionSetup, self).__init__(parent)
 self.setupUi(self)
```

Ukázka 3.3: část konstruktoru controlleru

3.4 Signály a sloty

Signály a sloty jsou v Qt použity pro komunikaci mezi objekty. Je to jedna z hlavních vlastností Qt a zároveň pravděpodobně ta část, kterou se Qt nejvíce odlišuje od jiných nástrojů pro tvorbu GUI.

Objekt vyšle definovaný signál, když se stane určitá událost, například uživatel stiskne tlačítko. Slot je funkce, která se zavolá na základě přijatého signálu. Qt widgety nabízejí mnoho předdefinovaných signálů, ale zároveň také umožňuje definovat vlastní.[24] Toho jsem také využil při tvorbě vlastních widgetů, jako tabulka s tlačítky.



Obrázek 3.4: Signály a sloty v Qt[24]

Vytvoření, odchytávání a vyslání signálu ukážu na třídě *ElemTable* symbolizující tabulku s tlačítky. Tato třída definuje vlastní signál *elementSelected*, který vysílá, pokud uživatel vybere nějakou řádku v tabulce. V tomto signálu zároveň tabulka posílá i přímo instanci objektu, který tato řádka v tabulce symbolizuje, tj. pokud jsou v tabulce skupiny, pak signál bude obsahovat přímo příslušnou instanci objektu *Group*. Signál je definován následovně:

```
class ElemTable(QWidget, table.Ui_Form):
  elementSelected = pyqtSignal(object)
```

Tento signál je vysílán v metodě *onElementSelected*, která je zároveň slotem pro signál *currentChanged* selection modelu *QItemSelectionModel* tabulky. (viz [25]) To znamená, že metoda *onElementSelected* je volána vždy, když se v tabulce změní označený řádek.

```
@pyqtSlot()
def onElementSelected(self):
  elem = self.model.data(self.table.currentIndex(),
  ↪  Qt.UserRole)
  self.elementSelected.emit(elem) # zde je vyslán signál
  ↪  elementSelected
```

Můžeme si všimnout dekorátoru `@pyqtSlot()`, kterým mohou být označeny metody, které slouží jako sloty. Dle oficiální dokumentace PyQt[26] může někdy být nutné označit metodu dekorátorem a připojení signálu do odekorovaného slotu je rychlejší a spotřebovává méně paměti. Pokud signál obsahuje nějaké parametry, které slot bude vyžadovat, pak je nutné do dekorátoru zapsat typy těchto parametrů, např. pro signál *elementSelected* by mohl vypadat dekorátor slotu takto `@pyqtSlot(Group)` (za předpokladu, že víme, že v tabulce budou pouze objekty typu *Group*).

Připojení signálu do slotu probíhá pomocí metody *connect*, která je definovaná přímo na samotném signálu:

```
table = ElemTable(...)
table.elementSelected.connect(method)
```

Od této chvíle se vždy, když se vykoná `self.elementSelected.emit()` v instanci *table*, zavolá se i metoda *method()*.

3.5 Hledání vhodného rozdělení

Problém nalezení vhodného rozdělení, podle zadaných požadavků a omezení, dle funkčního požadavku *Rozdělení skupiny do týmů* se ukázal jako velmi netriviální. Je pravděpodobné, že neexistuje algoritmus, který by dokonale řešil zadaný problém v rozumné složitosti. Jedná se totiž vlastně o problém nalezení rozkladu množiny, který by splňoval zadaná omezení.

Rozklad množiny S je takový disjunktní systém $T = \{A_1, A_2, \ldots, A_n\}$, že

$$\bigcup_{i=1}^{n} A_i = S$$

V tomto případě je množinou S rozdělovaná skupina a T rozdělení s týmy A_1 až A_n.

Jelikož, se mi nepodařilo najít vhodný algoritmus, který by požadované rozdělení (rozklad) nějakým způsobem konstruoval, nabízí se možnost vygenerovat všechna možná rozdělení a testovat, zda splňují všechny požadavky. Takový postup je bohužel velmi nedostatečný, jelikož počet rozkladů množiny o n prvcích P je

$$P \;=\; \sum_{j=1}^{n} S(n,j) \;=\; \sum_{j=1}^{n} \frac{1}{j!} \sum_{i=0}^{j-1} (-1)^i \binom{j}{i} (j-i)^k$$

Čísla $S(n,k)$ de nazývají Stirlingova čísla druhého druhu a vyjadřují počet různých rozkladů n-prvkové množiny do k tříd.[27] Číslo P se nazývá Bellovo číslo.[28]

Velikost skupiny	1	3	4	5	6	7	8	9	10
Počet možných rozdělení	2	5	15	52	203	877	4140	21147	115975

Tabulka 3.1: počet možných rozdělení

Tabulka 3.1 ukazuje, jak rychle roste počet možných rozdělení skupiny do týmů. Pro ještě lepší představu, proč není možné generovat všechny možnosti uvedu, že pro skupinu o 24 členech je počet možných rozdělení $445958869294805289 \approx 4 \times 10^{17}$. Proto jsem se spokojil raději s algoritmem, který sice nemusí vždy uspět, zato ale vždy doběhne v přijatelném čase.

3.5.1 Implementovaný algoritmus

Použitý algoritmus by se dal zjednodušeně popsat asi takto:

1. Vytvoř n množin, v každé bude právě 1 člen.

2. Pro všechna omezení typu „A musí být v týmu s B“ spoj množinu, ve které je A s množinou, ve které je B.

3. Každou množinu označ jako nezařazenou.

4. Dokud existuje nezařazená množina opakuj:

   Vezmi nezařazenou množinu a:

   - Pokud je aktuální počet týmů < minimální počet týmů, vytvoř nový tým a zařaď množinu do něj. (Buď je uživatelem zadán přesný počet požadovaných týmů nebo je minimální počet týmů = *počet členů skupiny* ÷ *minimální velikost týmu*.)
   - Jinak, zařaď aktuální množinu do nejmenšího týmu, do kterého můžeš. (Kontroluje se velikost týmu a omezení typu „A nesmí být v týmu s B“)

     Pokud takový tým neexistuje, vytvoř nový tým a množinu zařaď do něj. (po)

Úspěch tohoto algoritmu závisí na pořadí členů, resp. množin v bodě 4, protože pokud bychom množiny brali v nevhodném pořadí, může se stát, že výsledné rozdělení nebude splňovat požadovaná omezení, přestože takové rozdělení existuje. Například proto, že jako poslední množinu budeme zařazovat množinu s někým, kdo nesmí být v týmu s členem, který je už zařazen do jediného nezaplněného týmu a přitom, kdybychom tohoto posledního člena zařazovali jako předposledního, mohli bychom ho zařadit do jiného týmu. Abych alespoň trochu omezil takováto nevhodná pořadí, bere se nejprve větší množiny, jinak je pořadí náhodné (resp. randomizované). Což vysvětluje, proč je uživateli nabízena možnost vyzkoušet najít rozdělení se stejnými požadavky znovu i když se jej nepodaří najít.

Pokud uživatel vyžaduje, aby se preferovali týmy, ve kterých spolu lidé již spolupracovali, příp. naopak, v bodě 4.a. se hledá nejen jakýkoli nejmenší možný tým, ale právě ten, ve kterém spolu zařazovaní členové a členové v týmu spolupracovali nejčastěji, příp. nejméně často. Takže pokud budeme hledat rozdělení skupiny, ve které spolu v jednom týmu byli jen A a B, přičemž budeme preferovat opakující se týmy, ještě to nutně neznamená, že A bude v týmu s B. To se dá brát jako výhoda, jelikož rozdělování do týmu by asi mělo být trochu náhodné, ale i jako nevýhoda, jelikož v tomto případě by se asi dalo očekávat, že A a B ve stejném týmu budou.

3.5.2 Možnost vylepšení

Přesto, že použitý algoritmus je pro většinu případů dostatečný, rozhodně není dokonalý a dal by se určitě zvolit i komplikovanější přístup, například pomocí modifikace některých grafových algoritmů, který by možná řešil problém lépe nebo spolehlivěji. Proto jsem se snažil navrhnout rozhraní mezi aplikací a třídou hledající vhodné rozdělení tak, aby byla pokud možno nahraditelná a požadavky na ní srozumitelné.

Jedná se o třídu *DistributionMaker* v balíčku *utils.distribution_maker*. Pro pochopení API je ovšem nutné znát alespoň část struktury tříd *Group*, *Member* a *Requirement*.

3.5.3 Member

Třída *Member* nabízí pro hledání rozdělení následující parametry:

- **id__** – unikátní identifikátor člena (`int`).
- **name** – jméno člena.
- **history__count** – slovník (dictionary), ve kterém je pro každého dalšího člena skupiny, kolikrát s ním člen byl ve stejném týmu. Klíčem ve slovníku je instance *Member* a hodnotou je `int`. (`{Member: 1, Member: 2, Member: 0, ...}`)

Třída *Member* má také definovány metody `__eq__()` a `__hash__()`. *Member* nabízí i další metody, které by neměly být při hledání rozdělení volány, jelikož se většinou jedná o metody, které mění vlastnosti člena.

3.5.4 Requirement

Třída *Requirement* nabízí pro hledání rozdělení následující parametry:

- **member** – člen, který musí/nesmí být v týmu se všemi členy v `self.target_members`
- **target__members** – seznam členů, kteří musí/nesmí být v týmu s `self.member`
- **keyword** – klíčové slovo, symbolizující požadavek. (člen *keyword* být v týmu s . . .)

Třída *Requirement* sice nabízí parametr *keyword*, pro rozlišení, o jaký požadavek se jedná je ale vhodnější použít

```
isinstance(x, RequirementSameTeam)
```

pro požadavky typu „*A* musí být v týmu s *B*“ a `isinstance(x, RequirementDifferentTeam)` pro požadavky typu „*A* nesmí být v týmu s *B*“. Tento přístup ovšem vyžaduje import těchto tříd pomocí

```
from model.requirements import RequirementSameTeam,
↪  RequirementDifferentTeam
```

Aby nebylo nezbytně nutné importovat tyto třídy, je možné odlišit typ požadavku i pomocí parametru *keyword*, kdy hodnota `"musí"` odpovídá `RequirementSameTeam` a hodnota `"nesmí"`, `RequirementDifferentTeam`.

3.5.5 API třídy DistributionMaker

Dále uvádím požadované API třídy:

- **konstruktor** – Třída musí poskytovat konstruktor se sedmi parametry. Konstruktoru budou předány následující parametry:
  1. Seznam členů – seznam instancí třídy *Member*, viz část 3.5.3.
  2. Seznam požadavků – seznam instancí podtříd *Requirement*, viz část 3.5.4.
  3. Enum pro signalizování porušeného požadavku – třída typu *Enum*, sloužící pro signalizaci, že nalezené rozdělení porušuje některé z požadavků, viz Ukázka 3.5.
  4. Požadovaný počet týmů – `int` nebo `None`, pokud není požadovaný počet týmů specifikován.
  5. Požadovaná minimální velikost týmu – `int` nebo `None`, pokud není minimální velikost týmu specifikována.
  6. Požadovaná maximální velikost týmu – `int` nebo `None`, pokud není maximální velikost týmu specifikována.
  7. Preference – jedna z hodnot:
     - `DistributionMaker.PREFER_SAME_TEAMS` – pokud požadujeme opakující se týmy
     - `DistributionMaker.PREFER_DIFFERENT_TEAMS` – pokud požadujeme nové týmy
     - `None` – pokud není preference specifikována.

  Třída *DistributionMaker* musí definovat zmíněné hodnoty.

- metoda **distribute** – bezparametrická metoda, jejíž návratovou hodnotou je seznam seznamů (nebo jiných iterovatelných struktur) symbolizující rozdělení do týmů. Například

  ```
  [[Member, Member], [Member], [Member, Member, Member]]
  ```

  představuje šest členů rozdělených do tří týmů o jednom až třech členech.

- metoda **getViolatedRequirements** – bezparametrická metoda, jejíž návratová hodnota je seznam porušených požadavků. Hodnoty v seznamu jsou buď instance tříd typu *Requirements* nebo hodnoty z Enum *Violated*. Violated nabízí následující hodnoty:

  - *TEAM_COUNT* – požadovaný počet týmů.
  - *MIN_TEAM_SIZE* – požadovaná minimální velikost týmu.
  - *MAX_TEAM_SIZE* – požadovaná maximální velikost týmu.

  Například návratová hodnota

  ```
  [RequirementSameTeam, Violated.TEAM_COUNT,
  ↪ RequirementDifferentTeam, RequirementSameTeam]
  ```

  symbolizuje, že nalezené rozdělení porušuje dva požadavky typu „*A* musí být v týmu s *B*“, jeden požadavek typu „*A* nesmí být v týmu s *B*“ a nemá požadovaný počet týmů.

  Je zaručeno, že tato metoda bude volána až po volání metody *distribute*.

Třída *DistributionMaker* musí také splňovat testy definované v *tests/test_distribution_maker.py*, tyto testy jsou zaměřené právě na kontrolu API, viz část 4.2.

3.6 Ikony

Abych vylepšil vzhled aplikace, rozhodl jsem se přidat vlastní ikony k tlačítkům. Qt sice nabízí nějaké standardní ikony, viz [29]. Tyto ikony, se ale hodí pouze k některým častým operacím a dle mého názoru narušují celkový vzhled aplikace. Proto jsem využil ikony nabízené na webu `https://iconmonstr.com`, iconmonstr je velká sbírka kvalitních, jednoduchých ikon, dostupných v různých formátech. Ikony jsou k použití zdarma a to i pro komerční užití.[30]

```
class Violated(Enum):
        """
        Enum for symbolizing which requirement was violated
        """
        TEAM_COUNT = 1
        MIN_TEAM_SIZE = 2
        MAX_TEAM_SIZE = 3
```

Ukázka 3.5: Enum Violated pro signalizaci porušených požadavků

Pro přidání vlastních ikon do qt aplikace se nabízí dvě možnosti. Jednou z možností je ikony pouze stáhnout v některém z podporovaných formátů a při každém běhu aplikace se je pokoušet načíst přímo z určitého adresáře, pravděpodobně natvrdo zadrátovaného v kódu. Nebo, a to je také možnost, kterou jsem využil, lze vytvořit tzv. resource file, soubor, ve kterém jsou ikony v binární reprezentaci. Tudíž se při distribuci aplikace k uživateli nemusí přidávat všechny soubory s ikonami, nýbrž jen jeden, ve kterém jsou všechny.

Při vytváření tohoto souboru jsem postupoval následovně, nejprve jsem stáhl vybrané ikony ve formátu svg. Ikony jsem poté pomocí programu *Qt Creator* přidal do souboru *resources.qrc*, což je XML soubor, ve kterém jsou pro potřeby Qt uloženy informace o souborech (jako např. alias). Z tohoto souboru jsem pak pomocí programu *pyrcc5*, což je Python ekvivalent k Qt utilitě rcc[31], vytvořil modul *resources_rc.py*. Poté, co se modul importuje, je možné vyhledávat soubory v něm uložené pomocí definované cesty v Qt Creatoru a tedy i v již zmíněném XML souboru, s dvojtečkou na začátku, např. `QIcon(':/icons/back.svg')`.

Kapitola 4

Testování

4.1 Testování programátorem

Testováním programátorem se rozumí prověření funkčnosti kódu ihned po jeho vytvoření. V praxi jsou tyto testy mnohdy označovány jako *Assembly tests* a většinou je neprovádí přímo programátor, který kód vytvořil, nýbrž jiný programátor.[32] Toto testování je obzvláště výhodné v tom, že oprava chyby je v této fázi nejméně nákladná, jelikož programátor kód zrovna dokončil, takže jej má „v hlavě“ a je pravděpodobné, že zdroj chyby nalezne mnohem rychleji, než kdyby stejnou chybu měl opravovat na konci projektu, příp. na konci iterace projektu.

Při vývoji aplikace byl tento typ testování samozřejmě využit, ačkoli vzhledem k tomu, že jsem na projektu pracoval sám, testoval jsem kód, který jsem sám napsal. Přesto jsem těmto testům věnoval velkou pozornost, právě kvůli snadné opravě chyby. Při testování jsem se snažil zaměřit na potenciálně problematické situace, které jsem, díky znalosti kódu, vytipoval. V rámci tohoto testování jsem se také snažil kontrolovat kód z hlediska čitelnosti a srozumitelnosti.

4.2 Jednotkové testy

Jednotkové testy jsou testy, testující nejmenší samostatně testovatelnou jednotku. U objektově orientovaného programování se zpravidla jedná o testování jednotlivých tříd, příp. metod. Tyto testy se zapisují pomocí programového kódu a při jejich vytváření se využívá testovacích frameworků.[32]

Python nabízí ve své distribuci modul *unittest*[33], který jsem využil pro testování „rozdělovače“, třídy *DistributionMaker*, pro kterou jsem vytvořil

sadu testů. Tyto testy si kladou za cíl otestovat rozhraní třídy a validitu výsledků, tj. zda nalezené rozdělení všechny požadavky buď splňuje nebo uvádí jako nesplněné. Zda je třída schopná nalézt existující rozdělení tak, aby nebyly žádné požadavky porušeny se netestuje, protože implementovaný algoritmus pro nalezení rozdělení by takové testy mohl, ale nemusel splnit a aplikace s tím počítá (nabízí uživateli opakování pokusu o nalezení). Viz část 3.5.1.

Tato sada testů se spouští příkazem

```
python -m unittest tests.test_distribution_maker
```

spuštěným z adresáře *Implementace*. Pokud bychom se pokusili spouštět testy přímo z adresáře *tests*, testy by selhali kvůli nenalezení modulů.

4.3 Systémové testy

Posledním typem testů, které jsem využil k otestování aplikace, jsou tzv. systémové testy. Během těchto testů je aplikace ověřována jako funkční celek, testy ověřují aplikaci z pohledu zákazníka. Podle připravených scénářů simulují různé situace, které mohou při používání aplikace nastat. Tyto testy vlastně slouží jako výstupní kontrola, před předáním aplikace zákazníkovi.[32]

Základní testovací scénáře přímo vycházejí z případů užití popsaných v části 1.2.3, proto zde uvedu pouze scénáře, které testují speciální případy nebo přímo neodpovídají případům užití. Testování probíhalo na operačním systému Ubuntu ve verzi 16.04 LTS.

- Unikátnost jména skupiny

    1. Vytvoř novou skupinu.
    2. Přejmenuj vytvořenou skupinu na *skupina*.
    3. Vytvoř novou skupinu.
    4. Pokus se skupinu přejmenovat na *skupina*. Aplikace by neměla toto přejmenování povolit.
    5. Smaž původní skupinu (nyní pojmenovanou *skupina*).
    6. Přejmenuj druhou vytvořenou skupinu na *skupina*.

- Unikátnost jména člena v rámci skupiny
  1. Vytvoř skupinu - *skupina*.
  2. Přidej do skupiny člena *clen1*.
  3. Vytvoř další skupinu - *skupina1*.
  4. Přidej do skupiny *skupina1* člena *clen1*.
  5. Pokus se do skupiny *skupina1* opět přidat člena *clen1*. Aplikace by toto neměla povolit.
  6. Přidej do skupiny *skupina1* člena *clen2*.
  7. Pokus se přejmenovat člena *clen2* na *clen1*. Aplikace by toto neměla povolit.
- Duplikování rozdělení
  1. Nad jakoukoli skupinou vytvoř rozdělení.
  2. Rozdělení pojmenuj *rozdeleni*.
  3. První tým v rozdělení přejmenuj na *prvni_tym*.
  4. Rozdělení ulož.
  5. Zobraz historii rozdělení skupiny, kterou jsi rozděloval a stiskni pravé tlačítko nad rozdělením *rozdeleni*. Zvol „Duplikovat“.
  6. Zkontroluj, že první tým se jmenuje *prvni_tym*, rozdělení se jmenuje *rozdeleni* a složení týmů odpovídá původnímu rozdělení.
  7. Přesuň některého člena do jiného týmu a rozdělení přejmenuj na *duplikovane_rozdeleni*.
  8. Rozdělení ulož.
  9. V historii rozdělení skupiny zkontroluj, že se vytvořilo nové rozdělení *duplikovane_rozdeleni* se správnými týmy a rozdělení *rozdeleni* se nezměnilo.
- Přidání nového člena do rozdělení
  1. Nad jakoukoli skupinou vytvoř a ulož rozdělení.
  2. Do skupiny přidej člena *pridavany_clen*.
  3. Přejdi do úpravy rozdělení.
  4. Zkontroluj, že člen *pridavany_clen* je v tabulce „Nezařazeni“.
  5. Přesuň člena do *pridavany_clen* některého z týmů rozdělení.

6. Rozdělení ulož.
7. V historii skupiny zkontroluj, že člen *pridavany_clen* se v rozdělení nachází ve zvoleném týmu.

- Přepínač -h
  1. Spusť aplikaci s přepínačem -h.
  2. Zkontroluj, zda se zobrazila nápověda k použití.
- Přepínače -d a -fc
  1. Spusť aplikaci s přepínačem -d a cestou k existujícímu souboru, který není databází. Aplikace by měla zahlásit chybu a skončit.
  2. Spusť aplikaci s přepínačem -d a cestou k neexistujícímu souboru *path*. Aplikace vytvoří databázi v daném souboru.
  3. Vytvoř alespoň jednu skupinu a ukonči aplikaci.
  4. Spusť aplikaci s přepínačem -d a cestou *path*. Zkontroluj, že se v aplikaci nachází skupina z bodu 3.
  5. Ukonči aplikaci.
  6. Spusť aplikaci s přepínačem -d a cestou *path* a přepínačem -fc.
  7. Potvrď přepsání souboru pomocí „Y“.
  8. Zkontroluj, že aplikace neobsahuje žádné skupiny

Závěr

Cílem této práce bylo navrhnout a poté naimplementovat desktopovou aplikaci pro správu skupin, která by umožňovala rozdělení skupiny do týmů dle zadaných parametrů.

Výsledkem práce je funkční aplikace, která nabízí jednoduché rozhraní pro správu skupin. Aplikace také dokáže rozdělit skupiny do týmů, ačkoli může nastat situace, kdy se aplikaci požadované rozdělení nepodaří najít, přestože takové rozdělení existuje. Bylo by tedy vhodné do budoucna vylepšit algoritmus pro hledání vhodného rozdělení. Přesto si ale myslím, že aplikace je i v současném stavu plně dostačující a pro organizátory soustředění FIKS použitelná.

Literatura

[1] *FIKS: Fiťácký informatický korespondenční seminář* [online]. [cit. 2017-05-12]. Dostupné z: `https://fiks.fit.cvut.cz/`

[2] *Team Maker* [online]. [cit. 2017-05-12]. Dostupné z: `http://chir.ag/projects/team-maker/`

[3] *Teamistr* [online]. [cit. 2017-05-12]. Dostupné z: `https://play.google.com/store/apps/details?id=com.brownapps.sortr` pro Android nebo `https://itunes.apple.com/us/app/teamistr/id609410714?mt=8` pro iOS

[4] ARLOW, Jim a Ila NEUSTADT. *UML 2 a unifikovaný proces vývoje aplikací: objektově orientovaná analýza a návrh prakticky*. 2., aktualiz. a dopl. vyd. Brno: Computer Press, 2007. ISBN 978-80-251-1503-9.

[5] KANISOVÁ, Hana a Miroslav MÜLLER. *UML srozumitelně*. Brno: Computer Press, 2004. ISBN 80-251-0231-9.

[6] TIOBE Index. *TIOBE* [online]. [cit. 2017-05-01]. Dostupné z: `https://www.tiobe.com/tiobe-index/`

[7] *The Computer Language Benchmarks Game* [online]. [cit. 2017-05-01]. Dostupné z: `https://benchmarksgame.alioth.debian.org/u64q/compare.php?lang=gpp&lang2=java`

[8] FERG, Steve. Static vs. dynamic typing of programming languages. In: *Python Conquers The Universe* [online]. [cit. 2017-05-01]. Dostupné z: `https://pythonconquerstheuniverse.wordpress.com/2009/10/03/static-vs-dynamic-typing-of-programming-languages/`

[9] Python 3.0 Release. *Python.org* [online]. [cit. 2017-05-01]. Dostupné z: https://www.python.org/download/releases/3.0/

[10] OLSON, Randy. Python usage survey 2014. In: *Randal S. Olson* [online]. [cit. 2017-05-01]. Dostupné z: http://www.randalolson.com/2015/01/30/python-usage-survey-2014/

[11] *Qt Documentation* [online]. [cit. 2017-05-01]. Dostupné z: http://doc.qt.io/

[12] Data Serialization. *The Hitchhiker's Guide to Python* [online]. [cit. 2017-05-01]. Dostupné z: http://python-guide-pt-br.readthedocs.io/en/latest/scenarios/serialization/

[13] Data Persistence. *Python Documentation* [online]. [cit. 2017-05-01]. Dostupné z: https://docs.python.org/3.6/library/persistence.html

[14] Sqlite3. *Python Documentation* [online]. [cit. 2017-05-01]. Dostupné z: https://docs.python.org/3.6/library/sqlite3.html

[15] EICHBERG, Michael. *Domain Model and Domain Modeling* [online]. In: . Technische Universität Darmstadt [cit. 2017-05-03]. Dostupné z: http://stg-tud.github.io/eise/WS11-EiSE-07-Domain_Modeling.pdf

[16] SQL As Understood By SQLite: SQLite Keywords. *Sqlite.org* [online]. [cit. 2017-05-06]. Dostupné z: https://www.sqlite.org/lang_keywords.html

[17] *Pro Git: [everything you need to know about the Git distibuted source control tool]* [online]. 2nd ed. New York, NY: Apress, 2014, 1.1 Úvod - Správa verzí [cit. 2017-05-06]. ISBN 978-1484200773. Dostupné z: https://git-scm.com/book/en/v2

[18] *Git* [online]. [cit. 2017-05-06]. Dostupné z: https://git-scm.com/

[19] *GitLab Community Edition* [online]. [cit. 2017-05-06]. Dostupné z: https://gitlab.fit.cvut.cz/help

[20] HRODĚJČUK, Vojtěch. *Model-View-Controller* [online]. In: . [cit. 2017-05-06]. Dostupné z: http://voho.eu/wiki/model-view-controller/

[21] GAMMA, Erich. *Design patterns: elements of reusable object-oriented software*. Reading, Mass.: Addison-Wesley, c1995. ISBN 978-0201633610.

[22] Model/View Programming. In: *Qt Documentation* [online]. [cit. 2017-05-06]. Dostupné z: http://doc.qt.io/qt-5/model-view-programming.html

[23] Using Qt Designer. *PyQt 5.8.2 Reference Guide* [online]. [cit. 2017-05-07]. Dostupné z: http://pyqt.sourceforge.net/Docs/PyQt5/designer.html

[24] Signals & Slots. *Qt Documentation* [online]. [cit. 2017-05-07]. Dostupné z: http://doc.qt.io/qt-5/signalsandslots.html

[25] QItemSelectionModel Class. *Qt Documentation* [online]. [cit. 2017-05-07]. Dostupné z: http://doc.qt.io/qt-5/qitemselectionmodel.html

[26] New-style Signal and Slot Support. *PyQt 4.12 Reference Guide* [online]. [cit. 2017-05-07]. Dostupné z: http://pyqt.sourceforge.net/Docs/PyQt4/new_style_signals_slots.html

[27] KOLÁŘ, Josef. Kombinatorika - 3. In: *Základy diskrétní matematiky, BI-ZDM* [online]. České vysoké učení technické v Praze: Katedra teoretické informatiky FIT, 2011 [cit. 2017-05-07]. Dostupné z: https://edux.fit.cvut.cz/oppa/BI-ZDM/prednasky/ZDM-P08.pdf

[28] A000110: Bell or exponential numbers: number of ways to partition a set of n labeled elements. *The On-Line Encyclopedia of Integer Sequences* [online]. [cit. 2017-05-12]. Dostupné z: https://oeis.org/A000110

[29] JOE, Kuan. *List of Qt Icons* [online]. In: . [cit. 2017-05-08]. Dostupné z: https://joekuan.wordpress.com/2015/09/23/list-of-qt-icons/

[30] License Agreement. *Iconmonstr* [online]. [cit. 2017-05-08]. Dostupné z: https://iconmonstr.com/license/

[31] Resource Compiler (rcc). *Qt Documentation* [online]. [cit. 2017-05-08]. Dostupné z: http://doc.qt.io/qt-5/rcc.html

[32] Fáze a úrovně provádění testů. *Testování softwaru* [online]. [cit. 2017-05-10]. Dostupné z: http://testovanisoftwaru.cz/metodika-testovani/druhy-typy-a-kategorie-testu/faze-testu/

[33] Unittest — Unit testing framework. *Python documentation* [online]. [cit. 2017-05-11]. Dostupné z: https://docs.python.org/3/library/unittest.html

Příloha **A**

Seznam použitých zkratek

FIKS Fiťácký informatický korespondenční seminář

GUI Graphical user interface

UML Unified modeling language

JVM Java virtual machine

MVC Model-view-controller

XML eXtensible Markup Language

SVG Scalable Vector Graphics

Příloha **B**

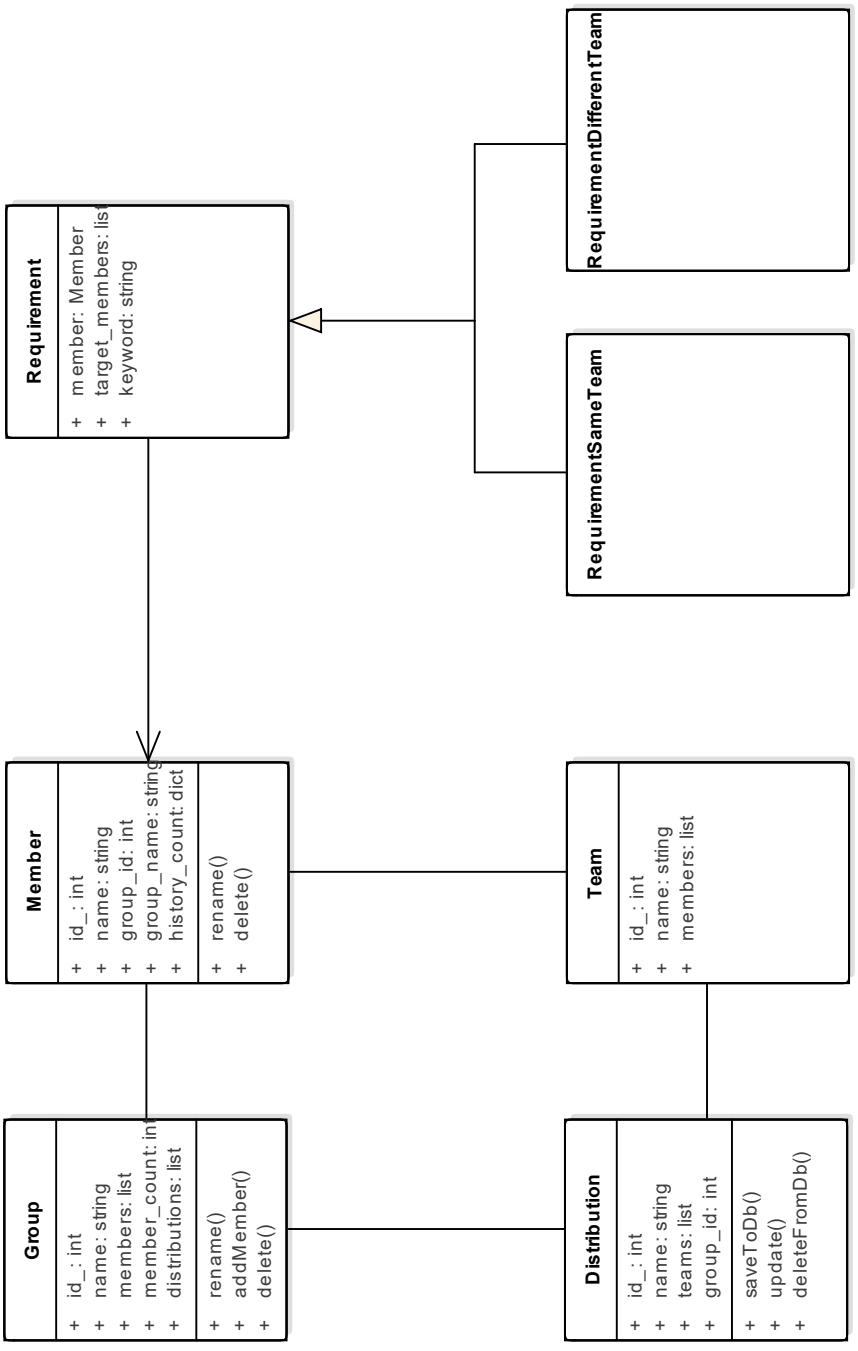
Obsah přiloženého CD

Zdrojové kódy i text práce jsou k dispozici také na `https://gitlab.fit.cvut.cz/stercjak/BP`.

- `BP`
  - `src` ..... zdrojová forma práce ve formátu LaTeX
  - `BP_Stercl_Jakub_2017.pdf` ..... text práce ve formátu pdf
- `Implementace` ..... zdrojové kódy aplikace
  - `grou.py` ..... spustitelný skript
  - `user_guide.pdf` ..... uživatlská příručka aplikace

Příloha C

Diagram tříd



Obrázek C.1: Diagram tříd