EKONOMICKÁ UNIVERZITA V BRATISLAVE

FAKULTA HOSPODÁRSKEJ INFORMATIKY

Evidenčné číslo: 103004/I/2024/36122176493267460

Návrh softvéru na podporu personálneho oddelenia konkrétneho podniku

Diplomová práca

2024 **Bc. Daniela Matušková**

EKONOMICKÁ UNIVERZITA V BRATISLAVE

FAKULTA HOSPODÁRSKEJ INFORMATIKY

Návrh softvéru na podporu personálneho oddelenia konkrétneho podniku

Diplomová práca

Študijný program: Informačný manažment
Študijný odbor: Ekonómia a manažment
Školiace pracovisko: Katedra aplikovanej informatiky
Vedúci záverečnej práce: Ing. Pavol Jurík, PhD.

Bratislava 2024 **Bc. Daniela Matušková**

Ekonomická univerzita v Bratislave
Fakulta hospodárskej informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta:	Bc. Daniela Matušková
Študijný program:	informačný manažment (Jednoodborové štúdium, inžiniersky II. st., denná forma)
Študijný odbor:	8. - ekonómia a manažment
Typ záverečnej práce:	Inžinierska záverečná práca
Jazyk záverečnej práce:	slovenský
Sekundárny jazyk:	anglický

Názov: Návrh softvéru na podporu personálneho oddelenia konkrétneho podniku

Anotácia: Cieľom práce je vytvoriť návrh softvéru na podporu procesov prebiehajúcich na personálnom oddelení konkrétneho podniku so zameraním na zozbieranie požiadaviek, modelovanie potrebných UML diagramov a tvorbu databázy. Bude vytvorený konceptuálny model, vykoná sa normalizácia, vytvorí sa logický a fyzický model databázy pre personálne oddelenie vybraného podniku. Pri návrhu softvérového riešenia sa využijú diagramy patriace do modelovacieho jazyka UML (Unified Modeling Language), napríklad use case diagram, sekvenčný diagram, stavový diagram, class diagram a pod.

Vedúci:	Ing. Pavol Jurík, PhD.
Katedra:	KAI FHI - Katedra aplikovanej informatiky
Vedúci katedry:	Ing. Mgr. Peter Schmidt, PhD.

Dátum zadania: 16.02.2023

Dátum schválenia: 14.03.2023

prof. Ing. Ivan Brezina, CSc.
osoba zodpovedná za realizáciu študijného programu

Čestné vyhlásenie

Čestne prehlasujem, že diplomovú prácu som zhotovila samostatne pod vedením vedúceho práce, pričom som sa opierala o dostupnú literatúru, uvedenú v zozname použitej literatúry.

Bratislava 25.04.2024 **.....**

Podpis študenta

Poďakovanie

Chcela by som poďakovať vedúcemu práce, Ing. Pavlovi Juríkovi, PhD., za jeho odbornú pomoc, ochotu poskytnúť cenné rady, skvelý prístup a usmerňovanie počas písania tejto práce.

ABSTRAKT

MATUŠKOVÁ, Daniela: Návrh softvéru na podporu personálneho oddelenia konkrétneho podniku. – Ekonomická univerzita v Bratislave. Fakulta hospodárskej informatiky; Katedra aplikovanej informatiky. – Ing. Pavol Jurík, PhD. – Bratislava: FHI EU, 2024, 81 s.

Téma záverečnej práce je: Návrh softvéru na podporu personálneho oddelenia konkrétneho podniku. Cieľom záverečnej práce bolo vytvoriť návrh softvéru na základe analýzy fungovania a potrieb personálneho oddelenia a s tým súvisiace modely a diagramy. Práca je rozdelená do troch kapitol a obsahuje 39 obrázkov. Prvá kapitola sa venuje charakteristike softvéru, vývoja softvéru, návrhu databázy a UML diagramom. V druhej kapitole je stanovený cieľ práce, metodika práce a metódy skúmania, spoločne s opisom podniku, požiadavkami na softvér a oboznámením sa s aktuálne používaným systémom od spoločnosti SAP. Posledná kapitola je zameraná na analýzu organizačnej štruktúry a funkcií oddelenia, návrh databázy a modelovanie UML diagramov.

Kľúčové slová:

softvér, softvérové inžinierstvo, analýza softvéru, návrh softvéru, návrh databázy, modelovanie diagramov, UML

ABSTRACT

MATUŠKOVÁ, Daniela: Design of software to support the personnel department of a particular company. – University of Economics in Bratislava. Faculty of Economic Informatics; Department of Applied Informatics. – Ing. Pavol Jurík, PhD. – Bratislava: FHI EU, 2024, 81 p.

The topic of this thesis is: Design of software to support the personnel department of a particular company. The aim of the master‘s thesis is to create a software design based on an analysis of the operating and requirements of the personnel department, along with related models and diagrams This thesis is divided into three chapters and it contains 39 figures. The first chapter targets the characteristics of software, software development, database design, and UML diagrams. The second chapter concentrates on setting the main aim of the thesis, work methodology and research methods, along with a description of the company, software requirements, and familiarization with the currently used system product from SAP. The last chapter is devoted to the analysis of the organizational structure and functions of the department, database design, and UML diagram modeling.

Key words:

software, software engineering, software analysis, software design, database design, diagram modeling, UML

Obsah

ÚVOD

Návrh softvéru je rozhodne jednou z najdôležitejších častí celého životného cyklu vývoja softvéru, a súčasne chyby pri analýze a návrhu sú takmer vždy najnáročnejšie na neskoršiu úpravu a rovnako aj na ich odstránenie sú potrebné veľmi veľké finančné, ale aj časové náklady. Z tohto dôvodu je kľúčové pred samotným programovaním a implementáciou akéhokoľvek softvéru klásť dôraz najmä na analýzu prostredia systému, porozumenie procesov, zozbieranie požiadaviek a kvalitný návrh softvéru pre konkrétny podnik. V tejto diplomovej práci sa budeme venovať analýze a návrhu softvéru na podporu personálneho oddelenia podniku Miba Steeltec, s.r.o., ktorý nám dovolí nahliadnuť do jeho fungovania, priblíži nám svoju organizačnú štruktúru, funkcie, ktoré sa na tomto oddelení vykonávajú, ich požiadavky na informačný systém, ako aj skúsenosti a názor na momentálne používaný softvérový produkt od spoločnosti SAP.

V úvode je dôležité vyzdvihnúť význam správneho a efektívneho informačného systému pre manažovanie ľudských zdrojov v organizácii. Personálne oddelenie sa síce nepodieľa priamo na zisku, úspešnosti alebo celkovom dobrom mene organizácie, zohráva však kľúčovú úlohu v správe zamestnancov, ich dát a procesov, ktoré sú kritické pre fungovanie danej spoločnosti. Cieľom tejto práce je navrhnúť softvérové riešenie, ktoré umožní automatizovať a optimalizovať tieto procesy, čo povedie k zlepšeniu efektivity, presnosti a transparentnosti v správe ľudských zdrojov.

Pri analýze a návrhu softvéru sa zameriame na organizačnú štruktúru a funkcie oddelenia human capital, identifikáciu potrieb, zhromažďovanie požiadaviek od používateľov – zamestnancov a definovanie funkčných a nefunkčných požiadaviek na softvér. Ďalej navrhneme vhodné informačné štruktúry, modely, diagramy a rozhrania, ktoré budú podporovať efektívne spravovanie zamestnancov, evidenciu údajov o zamestnancoch, procesy náboru, hodnotenia výkonnosti, školenia a ďalšie personálne aktivity. Budeme sa zaoberať hlavne tvorbou rôznych diagramov, ako sú napríklad konceptuálny, logický a fyzický model a z UML prípady použitia, diagram tried, sekvenčné diagramy, stavové diagramy a ďalšie, ktoré budú slúžiť na vizualizáciu a štruktúrovanie návrhu softvéru.

Kapitola číslo jeden sa bude zameriavať na charakteristiku softvéru, postup a metódy používané pri jeho návrhu, popis relačných databáz, normalizácie, konceptuálneho, logického a fyzického modelu, ako aj vybraných UML diagramových techník. V druhej kapitole zadefinujeme cieľ práce, použitú metodiku a metódy skúmania, uvedieme opis

podniku Miba Steeltec, s.r.o., popíšeme aktuálne používaný softvér, jeho výhody a nevýhody a zhrnieme všetky funkčné aj nefunkčné požiadavky oddelenia human capital na softvér. V kapitole tretej sa budeme venovať samotnému modelovaniu diagramov organizačnej štruktúry, tvorbe databázy, normalizácii a grafickému znázorneniu činností a fungovania oddelenia pomocou UML diagramov.

Vzhľadom na dôležitosť personálneho oddelenia pre fungovanie celej organizácie je dôležité, aby navrhovaný softvér bol dobre premyslený, flexibilný a schopný prispôsobiť sa konkrétnym potrebám a procesom danej organizácie. Na základe analýzy a návrhu sa očakáva, že výsledný softvér poskytne efektívne a spoľahlivé nástroje pre riadenie ľudských zdrojov, čo prinesie zvýšenie produktivity a konkurencieschopnosti celej organizácie.

1 Súčasný stav riešenej problematiky doma a v zahraničí

V súčasnosti sa softvér a softvérové aplikácie používajú tak často, že si to častokrát ani neuvedomujeme. S ich rôznymi variáciami sa stretávame denne, napríklad v práci, v škole, pri voľnočasových aktivitách a už ich berieme ako bežnú súčasť nášho bytia. V mnohom nám uľahčujú život, pomáhajú nám so vzdelávaním, financiami, cestovaním, varením, zdravím, zoznamovaním, spoznávaním samého seba, športovaním a v podstate úplne so všetkým, pretože v dnešnej dobe existuje softvér už takmer na všetko, a to v akejkoľvek oblasti života človeka a jeho záujmov. Samotným vývojom softvérov od počiatočných fáz špecifikácie systému, plánovania, cez návrh, až po údržbu systému po nasadení do prevádzky, sa zaoberá odbor softvérové inžinierstvo.

1.1 Softvér

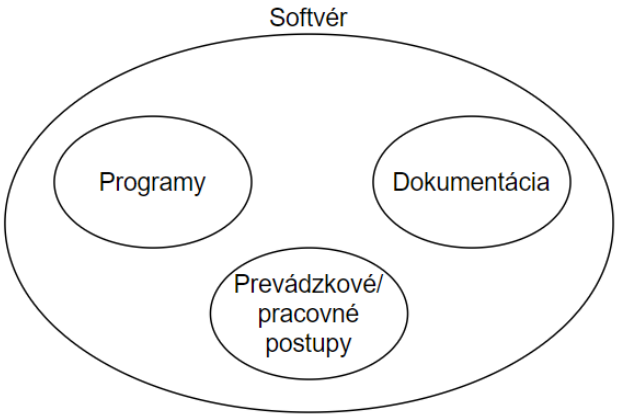
Vzhľadom k tomu, že témou tejto práce je návrh softvéru, na začiatok je potrebné uviesť, čo je to softvér a čím je špecifický. Keďže iba jedna komplexná definícia tohto pojmu nie je, každý autor, ktorý sa zaoberá touto problematikou, ho opisuje trochu inak. Tu uvádzame niekoľko definícií softvéru od uznávaných odborníkov a expertov v oblasti softvérového inžinierstva:

- Osterweil definuje softvér ako súbor programov, inštrukcií a dát, ktorý riadi fungovanie elektronických zariadení, počítačov a informačných systémov. Tento termín zahŕňa nielen kód, ktorý je vykonávaný na počítačoch, ale aj špecifikácie, návrhy a iné aspekty, ktoré sú nevyhnutné pre vývoj, prevádzku a údržbu softvérových aplikácií [32].
- Softvér označuje programové vybavenie informačných a komunikačných technológií. V širšom význame zahŕňa počítačové programy a dáta, a teda všetko, čo nie je hardvér. V praxi sa pojem softvér zväčša používa iba na označenie počítačových programov [25].
- ISO [18] popisuje softvér ako súbor počítačových programov, postupov, súvisiacej dokumentácie a údajov, ktoré sa týkajú prevádzky počítačového systému. Tento termín môže zahŕňať buď celý program alebo len jeho časť, postupy, pravidlá a príslušnú dokumentáciu informačného spracovateľského systému. Ide o program alebo súbor programov, ktoré sa používajú na riadenie prevádzky počítača.
- IBM [17] tvrdí, že softvér je sada inštrukcií alebo programov, ktoré hovoria počítaču, čo má robiť. Je nezávislý od hardvéru a umožňuje tak programovateľnosť počítačov. Rozlišujeme tri základné typy softvérov:

- **Systémový softvér** poskytuje základné funkcie ako operačné systémy, správu diskov, nástroje alebo riadenie hardvéru.
- **Programový softvér** poskytuje programátorom nástroje ako textové editory, kompilátory, linkery, debuggery a iné nástroje na vytváranie kódu.
- **Aplikačný softvér** (aplikácie) pomáha používateľom vykonávať úlohy. Príkladom sú kancelárske produktové balíky (Office), softvér na správu údajov, prehrávače médií alebo bezpečnostné programy (Eset, Avast, Kaspersky). Aplikácie sa vzťahujú aj na webové a mobilné aplikácie na online nákupy ako Amazon, Alibaba, Wolt, Alza alebo na komunikáciu, marketing, zábavu a zdieľanie fotiek pomocou aplikácií ako Facebook, Instagram alebo TikTok.
- Možným štvrtým typom je **vložený (embedded) softvér**. Softvér vložených systémov sa používa na riadenie strojov a zariadení, ktoré obvykle nie sú považované za počítače - telekomunikačné siete, autá, priemyselné roboty a ďalšie. Tieto zariadenia a ich softvér môžu byť pripojené ako súčasť internetu vecí [17].

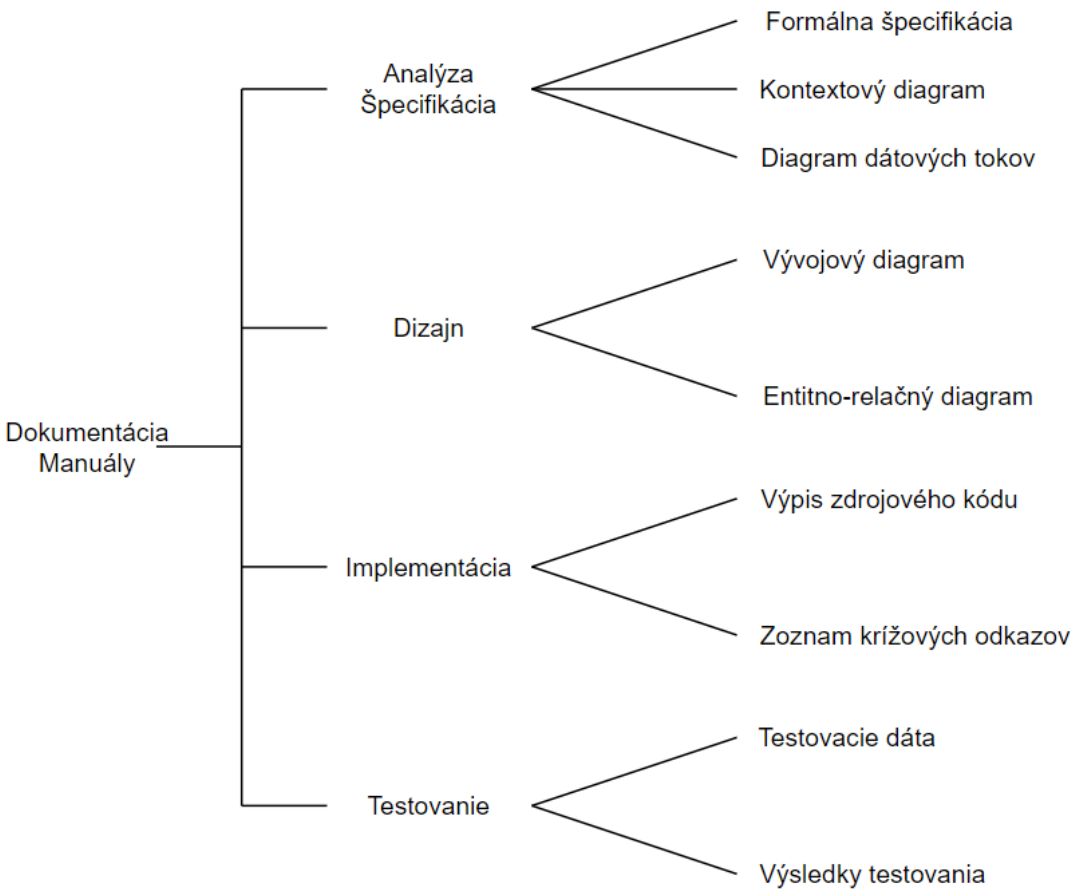
- Aggarwal [1] vo svojej definícii obdobne ako jeho kolegovia uvádza, že softvér nie je len program. Skladá sa z programov, dokumentácie týkajúcej sa akéhokoľvek aspektu programu a postupov používaných na nastavenie a prevádzku softvérového systému.

Jednotlivé komponenty softvérových systémov podľa Aggarwala sú znázornené na obrázku 1.



***Obrázok 1: Komponenty softvéru** (vlastné spracovanie podľa [1])*

Každý program je chápaný ako podmnožina softvéru a softvérom sa stáva len v prípade, že je vypracovaná potrebná dokumentácia a komplexné manuály s postupmi prevádzky. Program tvorí kombináciu zdrojového a objektového kódu. Dokumentácia pozostáva z rôznych typov manuálov, ako je zobrazené na obrázku 2.



***Obrázok 2: Dokumentácia softvéru** (vlastné spracovanie podľa [1])*

1.2 Softvérové inžinierstvo

Softvérové inžinierstvo predstavuje disciplínu, ktorej cieľom je systematický a efektívny vývoj, prevádzka a údržba softvérových systémov. Zahŕňa tvorbu a správu softvérových aplikácií, pričom kladie dôraz na procesy, kvalitu a spoľahlivosť. S narastajúcou komplexnosťou moderných informačných technológií a ich neustálym vplyvom na každodenný život sa softvérové inžinierstvo stáva kľúčovým pilierom pre vytváranie inovatívnych a spoľahlivých softvérových riešení, ktoré formujú našu momentálnu digitálnu realitu.

Na prvej konferencii o softvérovom inžinierstve v roku 1968 definoval Fritz Bauer softvérové inžinierstvo ako: "Stanovenie a využívanie zdravých inžinierskych princípov s cieľom dosiahnuť ekonomicky vyvinutý softvér, ktorý je spoľahlivý a efektívne funguje na skutočných strojoch" [12].

Stephen Schach definoval softvérové inžinierstvo podobne, ako: "Disciplínu, ktorej cieľom je produkcia kvalitného softvéru, softvéru, ktorý je dodaný včas, v rámci rozpočtu a spĺňa svoje požiadavky" [38].

Definície Fritza a Schacha sú síce staršie ale stále sú populárne, aktuálne a akceptovateľné pre väčšinu ľudí. Avšak vzhľadom na nárast nákladov na údržbu softvéru sa cieľ momentálne posúva k produkcii kvalitného softvéru, ktorý je udržateľný, dodaný včas, v rámci rozpočtu a tiež spĺňa svoje požiadavky.

Sommerville [39] uvádza, že softvérové inžinierstvo je inžinierska disciplína, ktorá sa zaoberá všetkými aspektmi tvorby softvéru. Kvalitný softvér by mal poskytnúť požadovanú funkcionalitu a výkon užívateľovi a zároveň by mal byť udržateľný, spoľahlivý a použiteľný. Základné aktivity v softvérovom inžinierstve zahŕňajú špecifikáciu softvéru, vývoj, validáciu a evolúciu. Rozdiel medzi softvérovým inžinierstvom a počítačovou vedou spočíva v ich zameraní. Počítačová veda sa sústreďuje na teóriu a základy, zatiaľ čo softvérové inžinierstvo rieši skôr praktické aspekty vývoja a dodávky užitočného softvéru. Systémové inžinierstvo sa potom zaoberá všetkými aspektmi vývoja systémov zahŕňajúcimi hardvér, softvér a procesné inžinierstvo.

Kľúčovými výzvami v oblasti softvérového inžinierstva sú najmä zvládanie rastúcej diverzity, požiadavky na skrátenie dodacích lehôt a vývoj dôveryhodného softvéru. Náklady na samotné softvérové inžinierstvo tvoria približne 60 % vývojových nákladov a 40 % testovacích nákladov. Pri vlastnom softvéri často náklady na evolúciu prevyšujú náklady na vývoj.

Nie je možné jednoznačne povedať, ktoré techniky a metódy v softvérovom inžinierstve sú najlepšie, pretože rôzne projekty si vyžadujú rôzne prístupy. Napríklad, hry by mali byť vytvárané pomocou série prototypov, zatiaľ čo kritické ovládacie systémy vyžadujú kompletnú a analyzovateľnú špecifikáciu. Príchod internetu znamenal dostupnosť softvérových služieb a možnosť vývoja vysoko distribuovaných systémov založených na službách. Taktiež viedol k dôležitým pokrokom v jazykoch programovania a opätovnému využívaniu softvéru [39].

1.2.1 Softvérový tím

Proces vývoja softvéru majú na starosti predovšetkým programátori, softvéroví inžinieri a softvéroví vývojári. Uvedené úlohy sa však vzájomne prekrývajú, a tak sa dynamika medzi nimi môže veľmi líšiť v závislosti od rôznych vývojových oddelení a komunít.

Programátori (programmers) alebo kóderi píšu zdrojový kód pre špecifické úlohy, ako sú napríklad zlučovanie databáz, spracovanie online objednávok, smerovanie komunikácií, vykonávanie vyhľadávaní alebo zobrazovanie textu a grafiky. Programátori typicky interpretujú pokyny od softvérových vývojárov (developers) a inžinierov. Najpoužívanejšie programovacie jazyky na vývoj softvéru sú podľa portálu Statista [40] napríklad JavaScript, HTML/CSS, Python, SQL, TypeScript C++, Java, Go, C# alebo PHP.

Softvéroví inžinieri (software engineers) aplikujú inžinierske princípy na vytváranie softvéru a systémov na riešenie problémov. Používajú pri tom modelovací jazyk a iné nástroje na vytváranie riešení, ktoré zväčša môžu byť uplatnené na problémy všeobecným spôsobom, na rozdiel od riešenia, ktoré sa hodí len pre konkrétny prípad alebo klienta a nikdy sa už ďalej nepoužije. Riešenia softvérového inžinierstva sa riadia vedeckou metódou a musia fungovať v reálnom svete. Ich zodpovednosť sa s rokmi zväčšila vzhľadom na to, že produkty sa stali čoraz inteligentnejšími s pridaním mikroprocesorov, senzorov a softvéru. Nielenže sa viac produktov spolieha na softvér pre diferenciáciu na trhu, ale aj vývoj daného softvéru musí byť koordinovaný s mechanickým a elektrickým vývojom a napredovaním produktov.

Vývojári softvéru (software developers) majú menej formálnu úlohu než inžinieri, ktorá môže úzko súvisieť s konkrétnymi oblasťami projektu, vrátane písania kódu. Zároveň riadia celý životný cyklus vývoja softvéru, vrátane práce s jednotlivými funkcionálnymi tímami na transformáciu požiadaviek na funkcie, správy vývojových tímov a procesov a vykonávania testovania a údržby softvéru.

Práca na vývoji softvéru nie je obmedzená len na programátorský alebo vývojársky tím. Profesionáli ako vedci, tvorcovia zariadení a výrobcovia hardvéru tiež vytvárajú softvérový kód, aj keď nie sú primárne vývojármi softvéru. Tvorba softvéru nie je obmedzená len na tradičné odvetvia informačných technológií, ako sú softvérové alebo polovodičové spoločnosti, keďže v dnešnej dobe tieto spoločnosti už predstavujú menej ako polovicu firiem vykonávajúcich vývoj softvéru [17].

1.3 Postup vývoja softvéru

Vývoj softvéru je pomerne náročný proces, ktorý môže pokojne trvať pár mesiacov alebo aj niekoľko rokov a zvyčajne zahŕňa nasledujúce kroky:

- **Výber metodológie**: Vybranie metodológie na stanovenie rámca, v ktorom sa budú aplikovať kroky vývoja softvéru. Popisuje celkový pracovný proces alebo cestovný plán pre konkrétny projekt. Metodológie môžu zahŕňať napríklad agilný vývoj (agile development), DevOps (spojenie vývoja a prevádzky), rýchly vývoj aplikácií (rapid application development - RAD), škálovaný agilný rámec (scaled agile framework - SAFe), vodopádovú metodológiu (waterfall) a iné.
- **Získavanie požiadaviek**: Porozumenie, presné zadefinovanie a zdokumentovanie toho, čo je požadované od používateľov a iných zainteresovaných strán.
- **Výber alebo vytvorenie architektúry**: Architektúra je chápaná ako podkladová štruktúra, v rámci ktorej bude neskôr výsledný softvér fungovať.
- **Tvorba dizajnu**: Vytvorenie dizajnu okolo riešení problémov, ktoré sú vyslovené požiadavkami. Často súvisí aj s procesnými modelmi alebo storyboardmi a celkovou víziou.
- **Vytvorenie modelu**: Použitie modelovacieho nástroja, ktorý používa modelovací jazyk ako SysML alebo UML na vykonávanie skorého overovania, prototypovania a simulácie dizajnu.
- **Konštrukcia kódu**: Samotné písanie kódu v príslušnom programovacom jazyku. Zahŕňa taktiež aj recenzie od kolegov a tímu na včasné odstránenie problémov a rýchlejší proces tvorby kvalitného softvéru.
- **Testovanie**: Testovanie s vopred naplánovanými scenármi ako súčasť dizajnu a kódovania a vykonávanie výkonnostného testovania na simuláciu zaťaženia na aplikácii.
- **Správa konfigurácie a chýb**: Zahŕňa porozumenie všetkým častiam softvéru (požiadavky, dizajn, kód, test) a vytváranie odlišných verzií softvéru. Stanovenie priorít kvality zabezpečenia a kritérií vydania na riešenie a sledovanie chýb.
- **Nasadenie softvéru**: Nasadenie softvéru na konečné použitie a riešenie a následné vyriešenie problémov používateľov.
- **Migrácia údajov**: Migrácia údajov na nový alebo aktualizovaný softvér z existujúcich aplikácií alebo zdrojov údajov v prípade, ak je to potrebné.

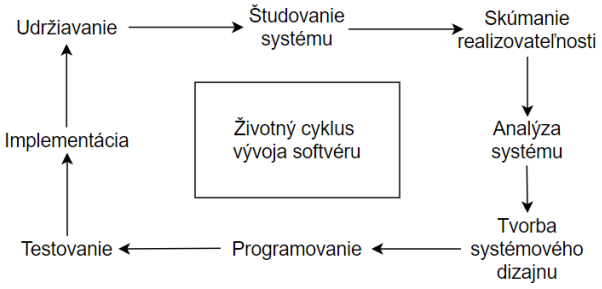
- **Správa a meranie projektu**: Správa a meranie projektu na udržanie kvality a dodávky počas životného cyklu aplikácie a na hodnotenie vývojového procesu pomocou modelov ako Capability Maturity Model (CMM) [17].

Fázy analýzy a špecifikácie požiadaviek, dizajnu a vývoja, testovania, nasadenia, údržby a podpory softvéru možno zaradiť do fáz životného cyklu, ale dôležitosť životného cyklu spočíva v tom, že sa recykluje s cieľom umožniť neustále zlepšovanie. Napríklad problémy používateľov, ktoré sa objavia vo fáze údržby a podpory, sa môžu stať požiadavkami na začiatku ďalšieho cyklu [17].

1.4 Životný cyklus softvéru

Software Development Life Cycle (SDLC) je metóda, pomocou ktorej môže byť softvér systematicky vyvíjaný, čo zvyšuje pravdepodobnosť dokončenia softvérového projektu v rámci stanoveného termínu a rozpočtu, rovnako ako aj pravdepodobnosť udržania kvality softvérového produktu podľa daných štandardov. Rámec životného cyklu vývoja systému popisuje postupnosť aktivít, ktoré majú systémoví návrhári a vývojári dodržiavať pri vytváraní softvéru. Často je považovaný za podmnožinu celkového životného cyklu vývoja systému. Všetky softvérové projekty prechádzajú fázami zhromažďovania požiadaviek, analýzy podnikania, návrhu systému, implementácie a testovania kvality zabezpečenia. Každý model SDLC má svoje výhody a nevýhody, a rovnako každý model SDLC môže poskytnúť lepšie funkcionality v jednej situácii než v druhej, keďže každý model a systém je individuálny. Vzhľadom k uvedenému je pomerne náročné rozhodnúť sa, ktorý model by mal byť vybratý pre poskytnutie konkrétnej sady funkcií za určitých okolností pre riešený problém, takže pri rozhodovaní je potrebné zvážiť kompromis a vybrať ten, ktorý je v danej situácii najvhodnejší [26, 29].

Jednotlivé modely SDLC sú si navzájom podobné a takmer všetky majú niektoré fázy životného cyklu vývoja softvéru spoločné. Tieto fázy sú znázornené na nasledovnom diagrame. (Obrázok 3)



***Obrázok 3: Fázy životného cyklu vývoja softvéru** (vlastné spracovanie podľa [29])*

Okrem aktivít vykonávaných počas samotného vývoja softvéru sa niektoré kroky životného cyklu vykonávajú až po ukončení hlavného vývoja. Vždy sa vykonáva implementačná fáza, ktorá sa týka už konkrétnej inštalácie systému do počítačových systémov klienta a následného testovania. Údržba softvéru je činnosť, ktorá začína až po dokončení vývoja softvéru. Softvér treba udržiavať nielen preto, že sa niektoré jeho komponenty "opotrebovávajú" a treba ich pravidelne kontrolovať a prípadne aj vymeniť, ale aj kvôli tomu, že v systéme často zostávajú určité reziduálne chyby, ktoré sa musia neskôr odstrániť. Údržba je preto nevyhnutná a kľúčová pre softvérové systémy a ich správne fungovanie [29].

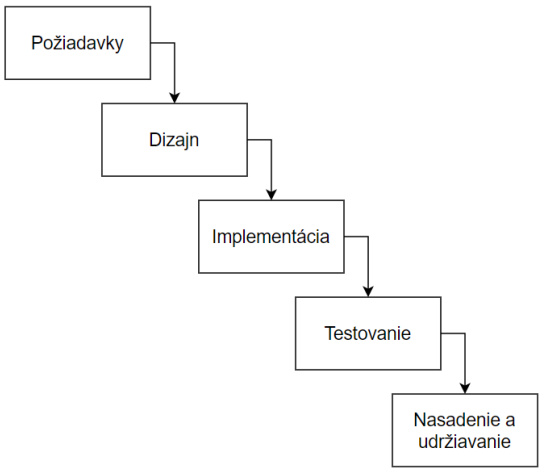
1.4.1 Modely SDLC

Vodopádový model (Waterfall model) bol navrhnutý Winstonom W. Royceom v roku 1970. Ide o lineárny sekvenčný model životného cyklu vývoja softvéru. Obsahuje fázy ako analýza požiadaviek, návrh, programovanie, testovanie a implementácia. Je nastavený tak, že fáza, ktorá je raz ukončená, sa už ďalej neopakuje, a zároveň proces vývoja nemôže prejsť na ďalšiu fázu, pokiaľ predchádzajúca fáza nie je úplne dokončená. Preto nie je veľmi užitočný, keď sú požiadavky projektu dynamického charakteru a je potrebné niektoré časti opakovať alebo sa k niečomu vracať (Obrázok 4) [29].

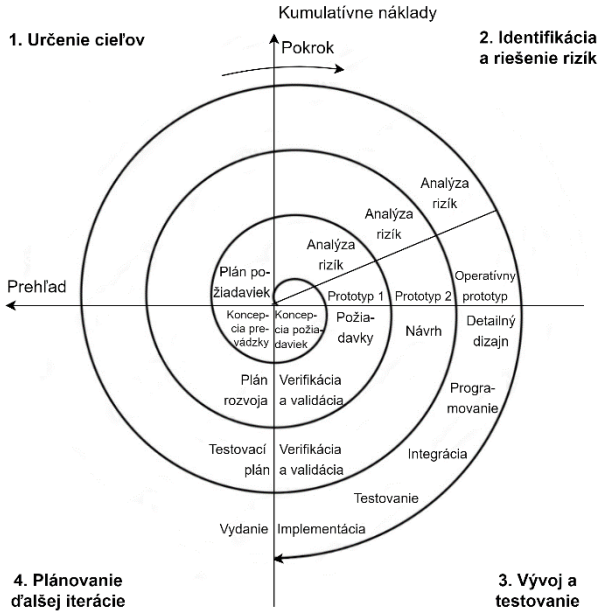
V reakcii na slabé stránky a neúspechy vodopádového modelu životného cyklu vývoja softvéru (SDLC) bolo vyvinutých niekoľko nových modelov, ktoré pridávajú určitú formu opakovania do procesu vývoja softvéru. V **špirálovom modeli** (spiral model), ako je znázornené na obrázku 5, začína vývojový tím s malým súborom požiadaviek a prechádza každou fázou vývoja (okrem implementácie a údržby) pre tento súbor požiadaviek. Na základe skúseností získaných počas počiatočnej prvej iterácie po spustení modelu vývojový tím postupne pridáva funkcionalitu pre ďalšie požiadavky v stále sa zväčšujúcich "špirálach", až kým skúmaný systém nie je pripravený na fázu implementácie a následnej údržby. Jeho nevýhodou je to, že fáza testovania je naplánovaná až po skončení implementácie [29, 30].

Ak sú požiadavky dobre pochopené a rozsah projektu je obmedzený (väčšinou z dôvodu blížiacich sa termínov odovzdania projektu), používa sa model rýchleho vývoja aplikácií (rapid application development - RAD). Tento model umožňuje vývojovému tímu vytvoriť plne funkčný systém v krátkych časových obdobiach (napr. 60 až 90 dní) (Obrázok 6) [29].

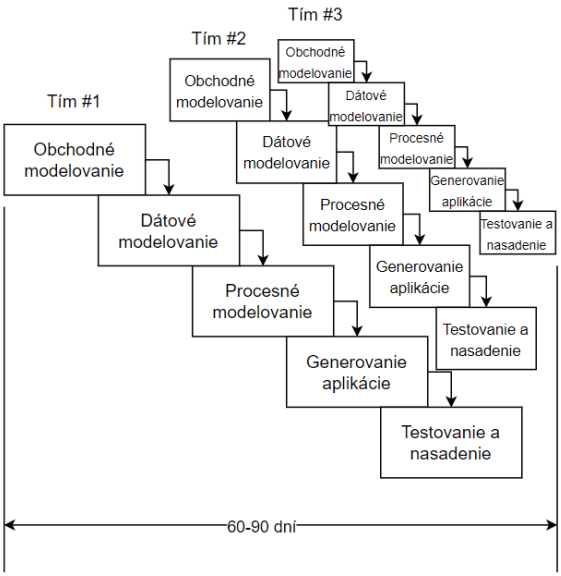
V-shaped model je variant vodopádového modelu, ktorý kladie dôraz najmä na overovanie a validáciu produktu, pričom testovanie produktu je plánované paralelne s príslušnou fázou vývoja. Grafické znázornenie modelu uvádzame na obrázku 7 [29].



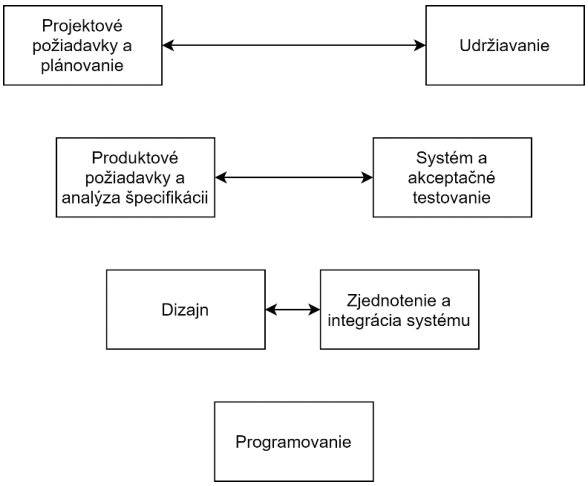
Obrázok 4: Vodopádový model SDLC
(vlastné spracovanie podľa [29])



Obrázok 5: Špirálový model SDLC
(vlastné spracovanie podľa [29])



Obrázok 6: RAD model SDLC
(vlastné spracovanie podľa [29])



Obrázok 7: V-shaped model SDLC
(vlastné spracovanie podľa [29])

Jednotlivé modely životného cyklu softvéru bolo pre potreby pochopenia postupov návrhu softvéru potrebné charakterizovať a priblížiť, ale nebudeme sa im venovať viac do hĺbky, nakoľko praktická časť práce je zameraná len na analýzu a návrh systému a ostatné časti, ako je programovanie, implementácia, testovanie a udržiavanie už nie sú predmetom tejto diplomovej práce. Zväčšené verzie obrázkov 5 a 6 uvádzame v prílohe č.1 a č.2.

1.5 Problémy a časté chyby pri tvorbe softvéru

V softvérovom inžinierstve sa často stretávame s významnými problémami a chybami, ktoré vyplývajú zo zložitosti, neviditeľnosti a meniteľnosti softvéru. Tieto faktory sú ovplyvňované aj nedostatočným porozumením zo strany zákazníkov, ktorí často nepresne vyjadrujú svoje požiadavky. Každý programátor má síce za cieľ minimalizovať počet chýb počas vývoja, ale neočakávané zmeny, najmä zmeny v požiadavkách, ich nútia k neustálym úpravám systému.

Pojem chyby by nemal byť chápaný len v kontexte behu programu, nakoľko zahŕňa aj nedostatky vo fázach špecifikácií požiadaviek, návrhu a implementácie softvéru. Napriek snahám o zníženie výskytu chýb, nevyhnutné zmeny a nejasné požiadavky zákazníkov vedú k častým zmenám v systéme. Spôsobené to býva často nedostatočnou špecifikáciou, pričom oprava takýchto chýb po uvedení systému do prevádzky sa často ukazuje ako veľmi nákladná a tiež aj časovo náročná. Je dôležité podotknúť, že rýchla identifikácia a odstránenie chýb počas fázy špecifikácie a návrhu môže výrazne znižovať náklady v porovnaní s neskoršími opravami.

Faktom je, že chyby sa často vyskytujú už v špecifikáciách a rovnako aj v nepochopení požiadaviek, čo potom vedie k zvýšeným nákladom na odstránenie daných chýb v neskorších fázach životného cyklu softvéru. Odstránenie chýb je spojené s požadovanou vysokou spoľahlivosťou softvéru, ktorá je kľúčovým faktorom, ktorý ovplyvňuje náklady na vývoj a údržbu. Investície do spoľahlivého softvéru sa ale môžu veľmi vyplatiť, pretože menej spoľahlivé systémy často potom vyžadujú výrazne vyššie náklady na ich údržbu. Výskum v oblasti identifikácie chýb a zlepšenia softvérových procesov je preto nevyhnutný, aby sme mohli lepšie porozumieť a efektívnejšie riešiť výzvy spojené s tvorbou softvéru. Aj keď výskyt chýb nie je nevyhnutný, ich systematické riešenie a odstránenie môže prispieť k vytváraniu kvalitnejších a spoľahlivejších softvérových produktov [3].

Barry Boehm a Victor R. Basili [4] uvádzajú v ich dielach desať najzávažnejších chýb, ktoré sa zvyknú objaviť počas vývoja softvéru, pričom zdôrazňujú potrebu odstraňovania týchto chýb. Tu uvádzame niektoré z nich:

- **Chybné a nepochopené požiadavky**: Ako už bolo uvedené, problémy často vznikajú z nejednoznačných alebo nesprávne interpretovaných požiadaviek, čo vedie k chybám v neskorších fázach vývoja.

- **Chybná a nepochopená funkcionálna špecifikácia**: Nespresnené a chybné funkcionálne špecifikácie môžu viesť k chybám v návrhu a implementácii, zvyšujúc tak náklady na opravy.
- **Zlý návrh komponentov**: Chyby v návrhu, ako napríklad nesprávne predpoklady o hodnote, štruktúre alebo o konkrétnych údajoch, nesprávne riadenie, plánovanie alebo zlý výpočet, môžu výrazne ovplyvniť celkovú funkčnosť vyvíjaného systému.
- **Chyby v návrhu alebo implementácii v jednom module**: Nesprávny návrh alebo implementácia v jednotlivých moduloch systému môžu spôsobiť chyby a komplikovať ich odstránenie, čím ovplyvňujú celý systém.
- **Nepochopenie vonkajšieho prostredia**: Nesprávne pochopenie externého prostredia môže viesť k chybám v interakcii systému s okolím.
- **Chyba v implementačnom jazyku alebo prekladači**: Chyby spojené s jazykom implementácie alebo jeho prekladačom tiež môžu viesť k nesprávnemu fungovaniu systému.
- **Chyby, ktoré vznikli odstránením predošlých chýb**: Opravy predchádzajúcich chýb môžu spôsobiť nové chyby alebo nežiaduce dôsledky, s ktorými vývojári nepočítali a musia ich ďalej riešiť.
- **Chyby vznikajúce pri zmene požiadaviek používateľov**: Zmeny požiadaviek používateľov môžu viesť k neustálym zmenám v systéme, čo vedie k novým chybám, nekonzistentnostiam, navýšeniu rozpočtu a prekročeniu stanovených termínov odovzdania produktu [4].

1.6 Prístupy riadenia a tvorba požiadaviek

Prístupy k riadeniu a vytváraniu požiadaviek v softvérovom inžinierstve sú pomerne rôznorodé. Zatiaľ čo jeden z prístupov, štruktúrované riadenie požiadaviek, sa snaží o systematické získavanie, dokumentovanie a spravovanie požiadaviek v procese vývoja softvéru, ďalší prístup, modelovo orientované riadenie požiadaviek, sa zameriava na vytváranie a spravovanie modelov požiadaviek, ktoré slúžia ako abstraktné reprezentácie a umožňujú ich ľahšiu analýzu a komunikáciu [10].

Agilný prístup k riadeniu požiadaviek sa zameriava na rýchlu adaptáciu na zmeny a flexibilitu v procese vývoja softvéru. Tento prístup kladie veľký dôraz na spoluprácu medzi členmi tímu a zákazníkom, časté dodávky funkčného softvéru a schopnosť rýchlo reagovať na spätnú väzbu. Vyznačuje sa iteratívnym a inkrementálnym štýlom vývoja, kde sa tímy

dynamicky prispôsobujú meniacim sa požiadavkám zákazníka. Tento prístup podporuje individuálnych členov tímu a ich interakcie, zameriava sa na dodávanie funkčného softvéru a spoluprácu so zákazníkom, pričom kladie dôraz na reagovanie na zmeny a flexibilitu v plánovaní. Niektoré z prístupov agilného vývoja zahŕňajú Dynamic Software Development Method (DSDM), Crystal, Feature Driven Development (FDD) a Adaptive Software Development (ASD), s najviac rozšírenými metódami Scrum a XP. XP sa zameriava na ľahkú metodiku pre malé a stredné tímy s nejasnými alebo rýchlo sa meniacimi požiadavkami, ktorá sa snaží riešiť problémy vývoja softvéru. V poslednej dobe je často skloňovaný najmä scrum, čo je agilný metodologický rámec, pri ktorom je práca rozdelená do krátkych iterácií nazývaných "sprinty", ktoré obvykle trvajú 2 až 4 týždne. Na začiatku každého sprintu sa tímu priradia úlohy z produktového backlogu, ktorý obsahuje zoznam všetkých požiadaviek na softvér. Počas sprintu tímy pracujú na týchto úlohách a na konci sprintu prezentujú hotový produktový inkrement. Scrum podporuje flexibilitu a adaptabilitu tímu, čo umožňuje rýchlu reakciu na zmeny v požiadavkách a prioritách [10, 13].

Klasický prístup sa práve naopak často spolieha na predvídateľnosť a štruktúrované plánovanie vývoja softvéru. V tomto prístupe sa najprv vykonáva detailné plánovanie a analýza požiadaviek a až následne sa začne samotný vývoj. Nevýhodou tohto prístupu je fakt, že zmeny v požiadavkách sa obvykle ťažšie prijímajú a implementujú, a proces vývoja je v porovnaní s agilným prístupom menej flexibilný. Tradičný prístup k vývoju softvéru je charakterizovaný riadením tímov, ktoré sú organizované hierarchicky a majú viacero vrstiev autority. Tieto tímy sú riadené manažérmi, ktorí delegujú prácu jednotlivým členom tímu na základe ich funkčných úloh. Praktiky tradičných tímov zahŕňajú dokumentáciu, špecifikácie a plánovanie, pričom existujú nepriame komunikačné kanály cez rôzne úrovne organizačnej hierarchie [10, 13].

Požiadavky sa môžu deliť do rôznych kategórií, ako napríklad funkčné požiadavky, ktoré popisujú konkrétne funkcie a operácie, ktoré má softvér vykonávať, a nefunkčné požiadavky, ktoré sa týkajú vlastností softvéru, ako je výkon, bezpečnosť a použiteľnosť.

Výhody správne riadených požiadaviek zahŕňajú zlepšenú komunikáciu medzi členmi tímu a zákazníkom, lepšiu predvídateľnosť projektu a minimalizáciu zmien v priebehu vývoja, čo vedie k zníženiu finančných a časových nákladov potrebných na dokončenie projektu. Správne riadenie požiadaviek tiež znižuje riziko chýb a zabezpečuje, že výsledný softvér bude lepšie vyhovovať potrebám a očakávaniam zákazníka. Požiadavky

sa však môžu v priebehu vývoja softvéru meniť, a preto je kľúčové mať flexibilný prístup k ich riadeniu a aktualizácii [10].

1.7 Relačné databázy a normalizácia

Použitie normalizácie a relačných databáz je kľúčové pri tvorbe návrhu softvéru. Normalizácia dátového modelu slúži na zabezpečenie optimálneho usporiadania dát v databáze, nakoľko tento proces pomáha minimalizovať redundanciu, zlepšuje integritu dát a zabezpečuje, aby dáta boli štruktúrované tak, aby boli ľahko spravovateľné a efektívne prístupné pre ďalšie spracovanie. Pri tvorbe softvéru je dôležité zvážiť použitie normalizácie a relačných databáz ako základných nástrojov pre efektívne a spoľahlivé spravovanie dát. Tento prístup zabezpečuje, že softvér je odolný, výkonný a ľahko rozšíriteľný vzhľadom na zmeny a nové požiadavky. Tvorcom relačného modelu databáz je Edgar F. Codd, americký matematik a informatik, ktorého práca na relačných databázach viedla k vývoju relačnej algebry a k predstaveniu konceptu normalizácie.

Normalizácia je proces, ktorý sa v relačných databázach používa na organizovanie dát tak, aby boli účinné, konzistentné a minimalizovali redundanciu. Cieľom normalizácie je rozložiť databázu do štruktúr, ktoré zabezpečujú správne uchovávanie a manipuláciu s dátami. Postup normalizácie zahŕňa identifikáciu funkčných závislostí, rozdelenie do normálnych foriem, odstránenie redundancie, zachovanie závislostí a overenie integrity. Normálne formy sú štandardizované požiadavky na štruktúru dát v relačných databázach. Každá normálna forma má svoje špecifické pravidlá a kritériá, ktoré definujú, ako by mali byť dáta organizované. Medzi najznámejšie normálne formy patria prvá (1NF), druhá (2NF) a tretia (3NF) normálna forma. 1NF vyžaduje, aby každý riadok v tabuľke mal jedinečný identifikátor a žiadne zložené hodnoty. 2NF vyžaduje, aby každý nekľúčový atribút v tabuľke bol plne funkčne závislý na celej primárnej kľúčovej časti. 3NF vyžaduje, aby každý nekľúčový atribút v tabuľke bol závislý na primárnom kľúči a nie na iných nekľúčových atribútoch. Použitie normálnych foriem pomáha zlepšiť výkon databázy, minimalizuje redundanciu a zabezpečuje, že dáta sú dobre štruktúrované a spravované [5,6].

1.7.1 Konceptuálny, logický a fyzický model

Konceptuálny, logický a fyzický model sú kľúčovými nástrojmi pri tvorbe návrhu softvéru a návrhu relačných databázových systémov. Konceptuálny model poskytuje komplexný pohľad na celú databázu a opisuje hlavné dátové objekty, pričom sa vyhýba detailom. Tento model je abstraktný a zameraný na požiadavky majiteľa a ciele podniku.

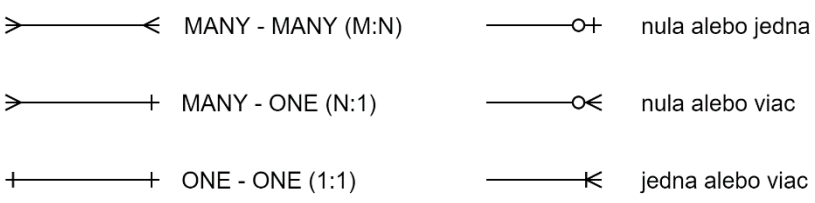
Logický model sa používa na preklad konceptuálneho modelu do interného modelu vybraného DBMS (Database Management System). Poskytuje podrobnosti o entitách a ich vzájomných vzťahoch, čím umožňuje štruktúrovanie dát. Fyzický model opisuje už aj fyzické charakteristiky dát, ako sú ich umiestnenie, cesty, formát a dátové typy. Zohľadňuje hardvérové a softvérové obmedzenia a je závislý od konkrétneho hardvéru a softvéru.

Postup pri použití týchto modelov sa zvyčajne pohybuje od vysokej abstrakčnej úrovne v konceptuálnom modeli, cez väčšie detaily v logickom modeli, až po implementačné detaily vo fyzickom modeli. Logický model sa často aktualizuje tak, aby spĺňal normálne formy a pravidlá čistého relačného dizajnu, takže dáta v logickom a neskôr aj fyzickom modeli bývajú spravidla v tretej normálnej forme [2].

Spomenuté modely sa zvyčajne zakresľujú pomocou entitno-relačných diagramov, ktoré už podľa názvu zachytávajú entity, relácie (vzťahy) a zvyknú sa pridávať aj ich atribúty. ERD nemajú jednotnú notáciu na zakresľovanie a väčšinou sa používa buď Chenov formát alebo relačný formát, ktorý sa snaží eliminovať jeho nedostatky. Pri znázorňovaní relácií medzi entitami sa zväčša uvádza aj počet výskytov jednej entity voči druhej v rámci ich vzájomného vzťahu, t. j. násobnosť (kardinalitu relácie). Najčastejšie sa zvyknú vyskytovať 3 situácie, a to:

- MANY – MANY (jeden zamestnanec pracuje na výrobe jedného výrobku a súčasne jeden výrobok vyrába viacero zamestnancov), používa sa vzťah M:N.
- MANY – ONE (na jednom oddelení pracuje viacero zamestnancov a zamestnanec pracuje na jednom oddelení), vzťah N:1.
- ONE – ONE (jeden zamestnanec má jednu pracovnú zmluvu, jedna zmluva patrí jednému zamestnancovi), vzťah 1:1 [8].

Graficky sa tieto vzťahy zakresľujú pomocou Crow's foot notácie. Na obrázku 8 uvádzame vzťahy M:N, N:1, 1:1 a aj zápis pre špeciálne prípady, kedy je potrebné použiť zápis násobnosti vzťahov nula alebo jedna, nula alebo viac a jedna alebo viac.



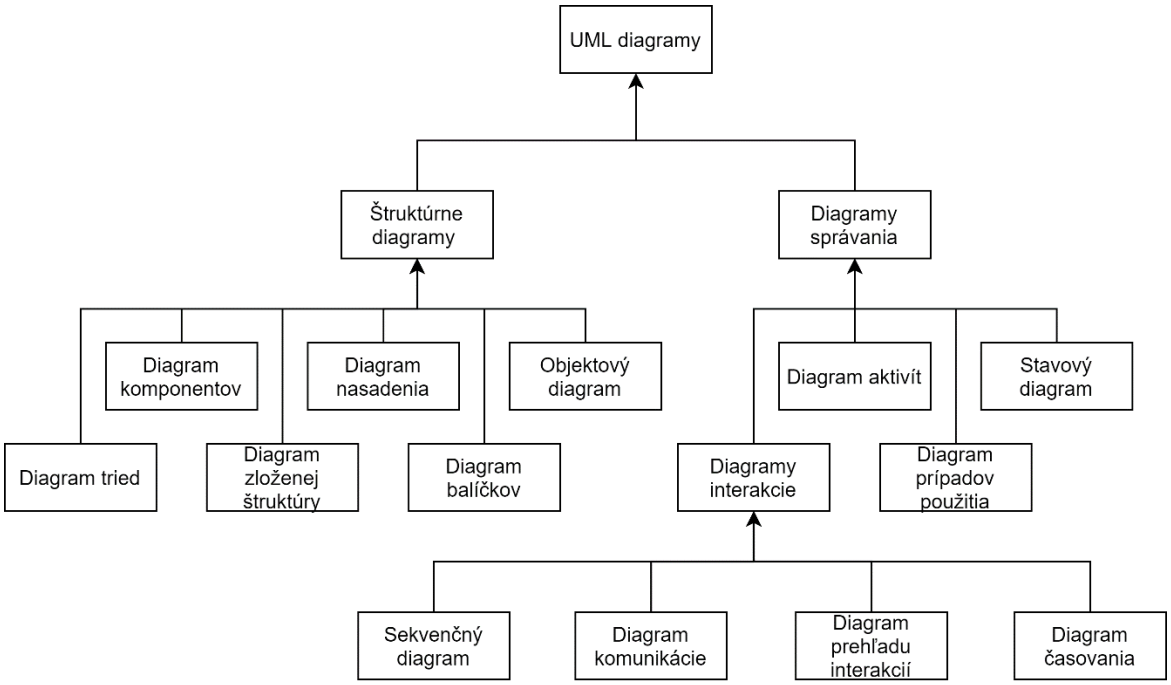
***Obrázok 8: Zápis kardinality** (vlastné spracovanie podľa [8])*

1.8 Štandardizovaný jazyk UML

Unified Modeling Language (UML) je špecifikácia, ktorá definuje štandardný vizuálny jazyk používaný pre analýzu, návrh a implementáciu softvérových systémov. Jeho cieľom je poskytnúť systémovým architektom, softvérovým inžinierom a vývojárom nástroje na modelovanie softvérových systémov a procesov súvisiacich s podnikaním. UML vznikol spojením troch vedúcich objektovo orientovaných metód (Booch, OMT a OOSE) a integroval množstvo postupov z návrhu modelovacích jazykov, objektovo orientovaného programovania a jazykov opisu architektúry. Vo verzii UML 2.0 sú definície abstraktnej syntaxe a sémantiky UML presnejšie, štruktúra jazyka je modulárnejšia a umožňuje výrazne lepšie modelovanie veľkých systémov.

Hlavným cieľom UML je posunúť stav softvérového priemyslu tým, že umožní interoperabilitu nástrojov na vizuálne modelovanie objektov. Na dosiahnutie zmysluplnej výmeny informácií o modeli systému medzi jednotlivými nástrojmi je potrebná zhoda v sémantike a syntaxi. UML spĺňa tieto požiadavky vďaka formálnej definícii spoločného metamodelu, presnému vysvetleniu sémantiky každého konceptu modelovania UML a špecifikácii ľudsky čitateľných prvkov notácie na reprezentáciu jednotlivých konceptov modelovania UML vrátane pravidiel pre ich kombinovanie do rôznych typov diagramov zodpovedajúcich rôznym aspektom modelovaných systémov [31].

Jazyk UML je štandard, ktorý definovala organizácia OMG (Object Management Group). Vznikol v roku 1995 zásluhou troch autorov - Grady Booch, Jim Rumbaugh a Ivar Jacobson. V roku 1997 bol UML prijatý ako štandard organizáciou OMG a od tej doby začali postupne iné metódy modelovania systémov upadať. Aktuálna verzia UML je UML 2.0, ktorá sa skladá z 13 diagramov, ktoré sú rozdelené na štruktúrne diagramy (statické) a diagramy správania (dynamické), pričom niektoré zdroje uvádzajú aj tretiu kategóriu, a to diagramy interakcií, ktoré sú chápané ako podkategória diagramov správania. Rozdelenie UML diagramov je zobrazené na obrázku 9. UML poskytuje súbor nástrojov pre analýzu, návrh a implementáciu softvérových a podnikových systémov, pričom inžinierom a vývojárom umožňuje modelovať a graficky znázorňovať rôzne aspekty modelovaného systému pomocou vizuálnych diagramov [34].



***Obrázok 9: Prehľad UML diagramov** (vlastné spracovanie podľa [34])*

Rumbaugh, Jacobson a Booch [19] vo svojej príručke uvádzajú, že hlavnými cieľmi vývoja UML bolo vytvorenie všeobecného modelovacieho jazyka, ktorý môžu používať všetci, podpora osvedčených návrhových postupov, ako je zapuzdrenie a separácia záujmov, a tiež aj riešenie súčasných problémov softvérového vývoja, ako je rozsiahlosť, distribúcia, konkurencia a tímový vývoj. UML nemá za cieľ byť kompletnou vývojovou metódou, ale podporuje existujúce procesy vývoja a poskytuje modelovacie koncepty pre moderné iteratívne prístupy. Zároveň sa UML snaží byť čo najjednoduchším, ale zároveň dostatočne výstižným na modelovanie plného rozsahu praktických systémov, čo vyžaduje zahrnutie komplexných koncepcií ako je súčasnosť, distribúcia, zapuzdrenie a komponenty. UML sa skladá z rôznych modelov a nie je niečo, čo sa dá zvládnuť za jeden deň, no postupným učením sa stáva čoraz zrozumiteľnejším [19].

UML je dnes považovaný za jeden z najpopulárnejších modelovacích jazykov v oblasti softvérového priemyslu. Jeho popularita vychádza predovšetkým z rozsiahlej podpory nástrojov, ktorá umožňuje využívať jazyk UML na modelovanie softvérových systémov a vykonávanie mnohých ďalších operácií, ako je generovanie kódu, analýza, modelovanie veľkých pohľadov, rozšírenie nástrojov, spolupráca viacerých používateľov atď. Dnes máme k dispozícii desiatky rôznych nástrojov podporujúcich štandard UML, ktoré sú široko využívané pre analýzu, návrh a vývoj softvéru [33].

Ako bolo vyššie už viackrát spomenuté, jazyk UML sa používa na tvorbu rôznych modelov. Vec, ktorá sa modeluje, sa všeobecne môže považovať za systém v nejakej oblasti. Pre už existujúci systém môže model predstavovať analýzu vlastností a správania sa a pre systém, ktorý je ešte len vo fáze plánovania, model predstavuje najmä špecifikáciu toho, ako má byť daný systém skonštruovaný a ako sa má správať [31].

UML model sa skladá z troch hlavných kategórií modelových prvkov, ktoré sa môžu použiť na opis modelovaného systému. Tieto kategórie sú:

- **Klasifikátory**: Klasifikátor opisuje množinu objektov. Objekt je jednotlivec so stavom a vzťahmi k iným objektom. Stav objektu identifikuje hodnoty vlastností klasifikátora objektu.
- **Udalosti**: Udalosť opisuje súbor možných výskytov. Výskyt je niečo, čo sa stane a má nejaké dôsledky vo vzťahu k systému.
- **Správania**: Správanie opisuje súbor možných vykonávaní. Vykonávanie je uskutočnenie sady akcií (v určitom čase), ktoré môže generovať a reagovať na výskyty udalostí, vrátane prístupu a zmeny stavu objektov.

UML modely neobsahujú objekty (object), výskyty (occurrence) alebo vykonávania (execution), pretože tieto časti sú súčasťou domény, ktorú modelujeme, a nie sú súčasťou obsahu samotných modelov. UML však obsahuje modelovacie konštrukcie pre priamu modeláciu jednotlivcov: špecifikácie inštancií, špecifikácie udalostí a špecifikácie vykonávania na modelovanie objektov, udalostí a vykonávaní v rámci konkrétneho kontextu. Tieto sú však opäť len modelové prvky, ktoré robia vyhlásenia o jednotlivcoch, ktoré sa modelujú. Takéto vyhlásenia môžu však byť pre modely neúplné, nepresné a abstraktné vzhľadom na účel daného modelu a môžu sa časom ukázať ako nesprávne. Jednotlivci, ktorí sa modelujú, sú však spravidla vždy úplní, presní a konkrétni v rámci svojej domény.

Vykonávanie správaní v modelovanom systéme môže mať za následok vytváranie a zánik objektov v rámci tohto systému. Systém môže tiež odkazovať na iné objekty, ktoré sú mimo systému. Všeobecne platí, že rozlíšenie, či je objekt interný alebo externý, nie je dôležité pre formálne sémantiky správ, avšak v niektorých prípadoch, predovšetkým v prípade statických vlastností a rozsahov klasifikátorov, sa môže stať, že systém explicitne vymedzuje objekty existujúce v rámci rozsahu vykonávania systému od tých, ktoré sú mimo systému [31].

1.9 Popis vybraných UML diagramov

V tretej kapitole tejto diplomovej práce sa budeme venovať modelovaniu vybraných UML diagramov, vďaka ktorým budeme môcť lepšie pochopiť a graficky znázorniť fungovanie konkrétneho personálneho oddelenia. Z tohto dôvodu v nasledujúcej časti bližšie definujeme nami vybrané UML diagramy, ich notáciu, použitie, výhody, ale samozrejme aj nevýhody.

1.9.1 Diagram tried

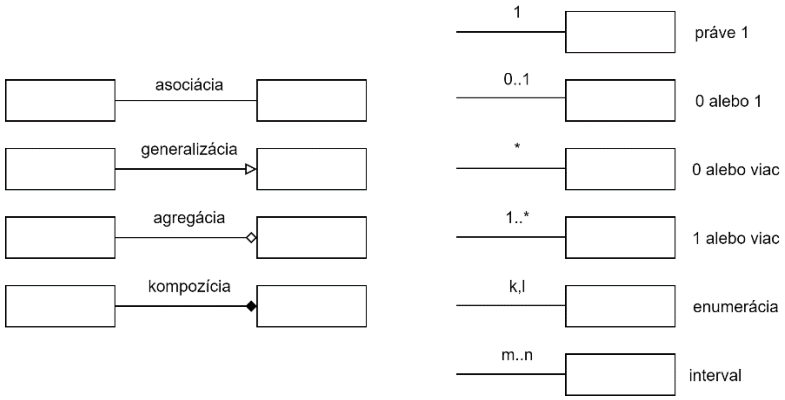
Diagram tried (class diagram) je vytvorený v rámci objektovo-orientovaného modelovania a slúži na vizuálne zobrazenie tried a ich vzťahov v rámci systému. Tento diagram patrí medzi štruktúrne UML diagramy, z čoho vyplýva, že zachytáva štruktúru systému a ukazuje, ako sú jednotlivé triedy vzájomne prepojené. Diagram tried je dôležitým nástrojom v návrhu softvérových systémov, nakoľko umožňuje zobrazenie kompozície a hierarchie tried.

V diagrame tried, ako aj vo väčšine diagramov UML, sa využíva stereotypovanie na definovanie rôznych typov tried. Najčastejšie používanými skupinami tried sú entitné triedy (entity), ktoré zobrazujú konkrétne entity alebo objekty v systéme, ako sú napríklad zákazníci, produkty alebo objednávky. Ďalej sa používajú triedy rozhraní (boundary), ktoré sú zodpovedné za interakciu systému s jeho okolím, ako sú napríklad používateľské rozhrania alebo služby poskytované externým systémom a v neposlednom rade triedy riadenia (control), ktoré obsahujú logiku riadenia a zodpovedajú za spracovanie požiadaviek a manipuláciu s dátami v systéme.

V diagrame tried sa taktiež zachytávajú vzťahy medzi triedami, ktoré sa označujú ako asociácie. Konce asociácií môžu mať pridelené role, označujúce úlohu, ktorú objekty v rámci vzťahu plnia. Na zachytenie kardinality a voliteľnosti vzťahov sa používa notácia M:N, pričom kardinalita definuje počet prvkov/inštancií jednej triedy, ktoré môžu byť spojené s prvkami inej triedy prostredníctvom asociácie a reprezentovaná je číslami alebo symbolmi na spojnicových líniách medzi triedami. Typické kardinality zahŕňajú 1:1, 1:N, M:N, kde 1 označuje jeden prvok a N označuje viacero prvkov.

V class diagramoch a celkovo v jazyku UML sa okrem „klasického“ vzťahu, asociácie, používajú navyše aj špeciálne druhy vzťahov, ako je agregácia, kompozícia a generalizácia. Agregácia sa využíva na zobrazenie vzťahu medzi celkom a jeho časťou, kde jedna trieda je súčasťou inej triedy, zatiaľ čo generalizácia vyjadruje vzťah nadradenosti a

podradenosti. Kompozícia je silnejšia forma agregácie, pričom časť je priamo závislá od celku a zmazaním celku zaniká aj časť. Jednotlivé druhy vzťahov a ich grafické znázornenie je možné vidieť na obrázku 10 [35].



Obrázok 10: Relácie a kardinalita vzťahov v jazyku UML (vlastné spracovanie podľa [8])

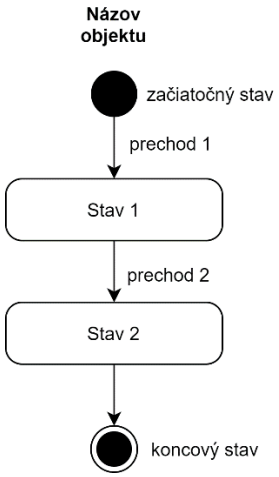
1.9.2 Stavový diagram

Stavové diagramy (state machine diagram) opisujú, ako sa systém mení v čase. Definujú možné stavy objektu, prechody medzi nimi, udalosti, ktoré spúšťajú prechody, podmienky pre prechody a s nimi súvisiace akcie. Používajú sa na popis dynamiky objektov, metód alebo protokolov komunikácie s používateľom. Tieto diagramy sa zvyčajne vytvárajú pre špecifické triedy, aby zobrazili životný cyklus objektu. Diagram zvyčajne obsahuje jeden počiatočný stav a aspoň jeden stav, ktorý reprezentuje ukončenie životného cyklu objektu. Dynamiku reprezentujú prechody medzi stavmi a udalosti, ktoré tieto prechody vyvolávajú [11, 35].

Prechod medzi stavmi objektu sa často spúšťa externým podnetom, najčastejšie v podobe správy alebo externej udalosti. Platný stav objektu je definovaný prípustnými hodnotami jeho atribútov, pričom prechod medzi stavmi vyjadruje zmenu týchto hodnôt. Okrem udalostí môžu objekty prechádzať medzi rôznymi stavmi aj pomocou volania metód alebo vykonávania určitých akcií. Tieto metódy môžu byť definované vo vnútri objektu a môžu sa spustiť v reakcii na príchod určitej udalosti alebo podnetu [23].

Prechody v stavových diagramoch definujú podmienky pre prechod objektu z jedného stavu do druhého. Sú reprezentované orientovanými líniami, ktoré vedú od jedného stavu k druhému a môžu zahŕňať udalosť, podmienku a akciu. Akcia a teda aj prechod do ďalšieho stavu sa vykoná len vtedy, ak je pri vzniku udalosti splnená podmienka. Ak udalosť nie je uvedená, prechod do ďalšieho stavu sa vykoná automaticky [11].

Jednotlivé stavy objektov sa znázorňujú prostredníctvom obdĺžnikov so zaoblenými hranami, ktoré sú prepojené šípkami. Začiatočný stav sa zakresľuje ako plný krúžok a koncový stav je znázornený menším plným krúžkom vo vnútri prázdneho krúžku. Príklad stavového diagramu je zakreslený na obrázku 11.



***Obrázok 11: Stavový diagram** (Vlastné spracovanie podľa [11])*

1.9.3 Diagram prípadov použitia

Use case diagram patrí medzi diagramy správania a používa sa na zachytenie a dokumentáciu funkcionality systému a jeho interakciu s aktérmi a okolím. Medzi hlavné komponenty patria aktéri, hranice systému, prípady použitia a komunikácia medzi nimi.

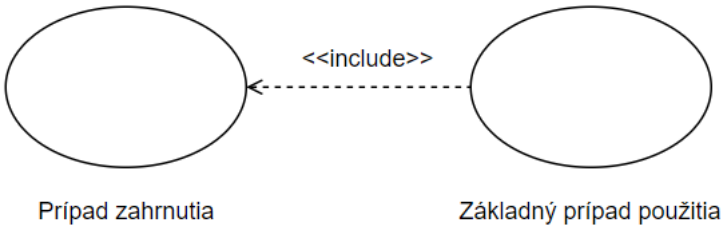
Aktér reprezentuje používateľské role alebo externé systémy, ktoré interagujú so systémom. Hranice systému ohraničujú pôsobnosť systému a jeho interakciu s okolím. Prípady použitia dokumentujú udalosti alebo scenáre, na ktoré musí systém reagovať, a zároveň opisujú funkčnosť, ktorú systém poskytuje, zatiaľ čo komunikácia zobrazuje vzťah medzi aktérmi a prípadmi použitia, kde aktéri komunikujú so systémom prostredníctvom jednotlivých prípadov použitia. Komunikácia medzi aktérmi a prípadmi použitia sa zvykne zakresľovať pomocou orientovaných spojníc [35].

Medzi jednotlivými prípadmi použitia môžu existovať aj dva typy vzťahov, a to include a extend.

Vzťah zahrnutia (include) je vzťahom, v ktorom jeden prípad použitia (základný prípad použitia) zahŕňa funkcionalitu ďalšieho prípadu použitia (prípad zahrnutia). Vzťah zahrnutia podporuje opätovné použitie funkcionality v modeli prípadov použitia. Využíva sa v prípadoch, kedy správanie prípadu zahrnutia je spoločné pre dva alebo viaceré prípady

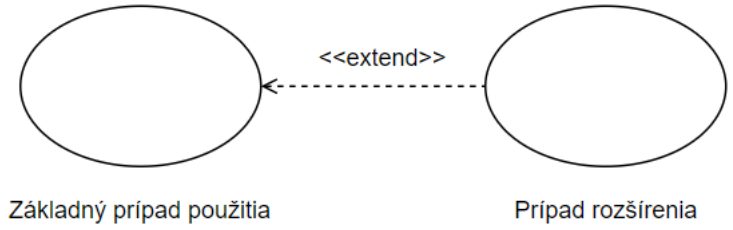
použitia alebo ak výsledok správania, ktorý prípad zahrnutia špecifikuje, je dôležitý pre základný prípad použitia.

V diagrame sa vzťah zahrnutia zobrazuje ako prerušovaná čiara so šípkou smerujúcou zo základného prípadu použitia na prípad zahrnutia, pričom kľúčové slovo include je pripojené ku konektoru (Obrázok 12) [16].



***Obrázok 12: Use case vzťah include** (Vlastné spracovanie podľa [16])*

Vzťah rozšírenia extend určuje, že zahrnutie rozšíreného prípadu použitia závisí od toho, čo sa deje pri vykonávaní základného prípadu použitia a či sú splnené podmienky rozšírenia. Zatiaľ čo základný prípad použitia je definovaný nezávisle a má vlastný význam, rozšírený prípad použitia sám o sebe význam nemá. Pozostáva z jednej alebo viacerých sekvencií správania, ktoré opisujú ďalšie správanie, ktoré môže postupne rozšíriť správanie základného prípadu použitia. Každý segment môže byť vložený do základného prípadu použitia v inom bode, ktorý sa nazýva bod rozšírenia. Príklad prípadu použitia s rozšírením uvádzame na obrázku 13 [15].

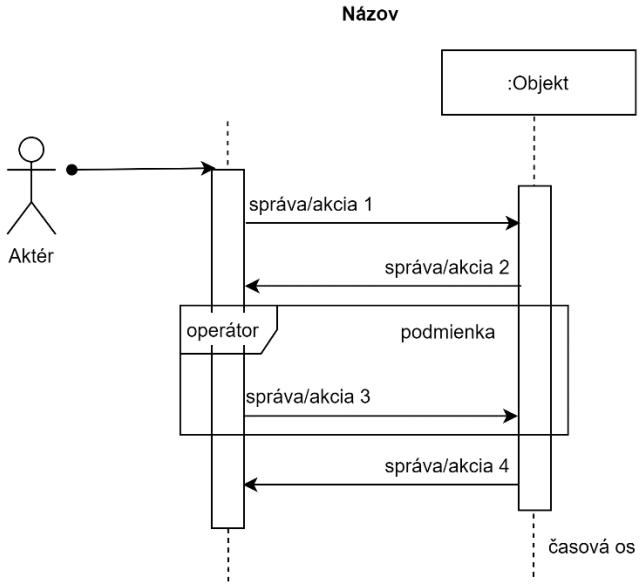


***Obrázok 13: Use case vzťah extend** (Vlastné spracovanie podľa [15])*

1.9.4 Sekvenčný diagram

Sekvenčné diagramy (sequence diagram) patria medzi diagramy interakcie a dokumentujú spoluprácu účastníkov v časovom sledovaní. Sú užitočné najmä pri opise interakcií s používateľmi. Vytvárajú sa na základe konkrétneho scenára prípadu použitia, ktorý zahŕňa základné aj alternatívne toky udalostí. Preto je dôležité mať správny a podrobný

use case, ktorý sa používa na konštrukciu interakcií medzi objektmi zapojenými v danom prípade použitia. Zložky sekvenčného diagramu zahŕňajú aktéra, objekty, správy, časovú os, výskyt vykonávania a rámec. Jednoduchý príklad sekvenčného diagramu je zobrazený na obrázku 14 [24, 35].



***Obrázok 14: Sekvenčný diagram** (Vlastné spracovanie podľa [24, 35])*

V sekvenčných diagramoch je možné využiť rôzne operátory, ktoré napomáhajú k zlepšeniu čitateľnosti, prehľadnosti a presnosti. Operátor opt (možnosť) reprezentuje výberové správanie na základe platnej podmienky, alt (alternatíva) označuje rôzne možnosti, pri ktorých sa vykoná spravidla len jedna z nich, v závislosti od splnenia určitej podmienky. Operátor loop (cyklus) popisuje opakované vykonanie jedného alebo viacerých krokov a na záver ref (odkaz) umožňuje prepojenie s iným sekvenčným diagramom, čím sa zjednodušuje zobrazenie komplexnejších procesov [24].

1.10 Výhody použitia UML diagramov pri návrhu softvéru

Koç a kolektív [22] vo svojej publikácii uvádzajú, že môžeme pozorovať viaceré výhody použitia UML diagramov v softvérovom inžinierstve. V tejto časti uvádzame niektoré z hlavných prínosov, ktoré možno identifikovať:

- **Vizuálna reprezentácia návrhu**: UML diagramy slúžia ako vizuálna reprezentácia návrhu softvéru. S ich pomocou je možné graficky znázorniť štruktúru, vzťahy a interakcie medzi rôznymi časťami systému.
- **Identifikácia požiadaviek a rozsahu**: UML diagramy uľahčujú identifikáciu požiadaviek a určenie rozsahu systému. Pomáhajú špecifikovať, ako budú jednotlivé

časti systému komunikovať a ako budú interagovať s užívateľmi a ostatnými systémami.

- **Zníženie komplexity softvérového vývoja**: Vytvorenie UML diagramov môže prispieť k zníženiu komplexity softvérového vývoja, nakoľko tieto diagramy umožňujú inžinierom lepšie porozumieť štruktúre a správaniu sa systému ešte pred začatím implementácie.
- **Štandardizácia a jednotné porozumenie**: UML sa považuje za štandard v oblasti objektovo orientovaného návrhu. Jeho použitie prispeje k štandardizácii návrhu, čo uľahčuje komunikáciu medzi členmi tímu a zabezpečuje jednotné porozumenie návrhu systému.
- **Zlepšená dokumentácia**: Vytvorenie UML diagramov poskytuje dôkladnú dokumentáciu návrhu. Tieto diagramy môžu slúžiť ako užitočný záznam pre budúcich vývojárov, správcov softvéru a analytikov, ktorí prídu do kontaktu so softvérom.
- **Podpora pre všetky fázy vývoja**: UML diagramy môžu byť využívané v rôznych fázach životného cyklu softvérového vývoja, vrátane analýzy, návrhu, implementácie a testovania. Týmto spôsobom poskytujú podporu pre celý životný cyklus vývoja softvéru.
- **Zlepšenie riadenia projektov**: Použitie UML diagramov môže prispieť k lepšiemu riadeniu projektov. Vytváranie jasných vizuálnych modelov môže uľahčiť plánovanie, identifikáciu rizík a monitorovanie pokroku projektu.
- **Zdokonalenie efektivity a spolupráce tímu**: Pozorujeme zvýšenú efektivitu tímov a spoluprácu medzi členmi tímu, nakoľko vizualizácia návrhu umožňuje ľahšie komunikovať a zdieľať nápady medzi členmi tímu [22].

Spoločnosť Microsoft [27] vníma štandardizovaný jazyk UML ako nástroj, ktorý posilňuje efektivitu a celkový úspech v oblasti vývoja softvéru a medzi ďalšie výhody jeho používania zaraďuje:

- **Zjednodušenie zložitých problémov**: UML poskytuje grafický a štandardizovaný spôsob pre prezentáciu a analýzu komplexných systémov, čo zjednodušuje porozumenie a riešenie zložitých otázok.
- **Otvorené cesty komunikácie**: UML definuje jasné a jednotné vizuálne prvky, ktoré uľahčujú komunikáciu medzi členmi tímu a zúčastnenými stranami v rámci projektu.

- **Automatizácia výroby softvéru a procesov**: UML diagramy môžu byť využité na generovanie kódu a automatizáciu procesov vývoja softvéru, čo vedie k zvýšeniu efektivity a presnosti.
- **Pomoc pri riešení dlhodobých problémov s architektúrou**: UML umožňuje detailné modelovanie architektúry, čo pomáha identifikovať a riešiť problémy s architektúrou už v počiatočných fázach vývoja.
- **Zvýšenie kvality práce**: Používanie jazyka UML zlepšuje kvalitu práce tým, že poskytuje jasné a presné modely, ktoré môžu byť ľahko analyzované a verifikované.
- **Znížené výrobné náklady a čas potrebný na uvedenie výrobku na trh**: UML umožňuje efektívnejší vývoj softvéru, čo vedie k skráteniu času na uvedenie výrobku na trh a znižovaniu nákladov s tým spojených [27].

Tieto výhody naznačujú, že použitie UML diagramov je pre softvérové inžinierstvo významným nástrojom, ktorý pomáha vytvárať lepšie navrhnuté a riadené softvérové systémy [22, 27].

1.11 Chyby pri UML diagramoch

V predchádzajúcej časti sme zhrnuli všetky benefity a výhody UML diagramov, no nemôžeme zabudnúť ani na to, že všetky diagramy sú modelované ľuďmi, ktorí nie sú neomylní a niekedy sa dopustia chýb najmä v dôsledku nedostatočného porozumenia modelu a požiadaviek. Pri tvorbe UML diagramov môže teda dôjsť k rôznym chybám, ktoré môžu ovplyvniť ich presnosť a účinnosť. Niektoré z najčastejších chýb zahŕňajú napríklad nesprávne porozumenie domény modelovaného systému a požiadaviek systému, chyby súvisiace s nekonzistenciou s diagramom tried, ako napríklad chýbajúce alebo nadbytočné vzťahy, nesprávne multiplicity a nejasné pomenovanie entít alebo atribútov. Ďalej sa často objavuje nesprávne pomenovanie časových osí (lifelines), nepochopenie notácie a sémantiky rôznych šípok, objektov a správ alebo nesprávne vytváranie nových objektov [14].

Dranidis [7] vo svojom článku opisuje, že najčastejšími problémami pri UML diagramoch a najmä v diagrame tried sú:

- Chýbajúce alebo nesprávne umiestnené metódy v diagramoch tried.
- Chýbajúce vzťahy medzi triedami.
- Nesprávny typ vzťahu medzi triedami.

Tieto chyby vedú k sémanticky chybným diagramom, ktoré potom nemôžu byť správne prekonvertované na kód [7].

Keďže UML diagramy sú súčasťou praktickej časti tejto práce, vybrali sme 4 diagramy, ktoré sme použili neskôr v kapitole 3 a bližšie popísali chyby, ktoré sa zvyknú opakovať v každom z nich. Medzi najčastejšie patrí:

- Diagram prípadov použitia (use case diagram): Chyby v textovej špecifikácii detailov prípadov použitia, nesprávne pomenovanie prípadov použitia a nesprávne použitie vzťahov include, extend alebo dedenia vzťahov.
- Diagram tried (class diagram): Chýbajúca navigovateľnosť asociácií, nesprávne pomenovanie lifelines a tiež chýbajúci akýkoľvek popis asociácií.
- Stavový diagram (state machine diagram): Použitie neznámych operácií vo volajúcich udalostiach (call event) a použitie neznámych operácií alebo atribútov v akciách.
- Sekvenčný diagram (sequence diagram): Nesprávne pomenovanie lifelines a chybné argumenty správ [14].

2 Cieľ práce, metodika práce a metódy skúmania

V druhej kapitole tejto diplomovej práce sa zaoberáme definovaním hlavných a vedľajších cieľov, ktoré sme si stanovili. Ďalej popisujeme metodiku práce, teda postupnosť krokov, vďaka ktorým sme stanovené ciele dosiahli. V poslednej časti kapitoly uvádzame použité metódy a techniky.

2.1 Cieľ práce

Cieľom práce je vytvoriť návrh softvéru na podporu procesov prebiehajúcich na personálnom oddelení konkrétneho podniku so zameraním na zozbieranie požiadaviek, modelovanie potrebných UML diagramov a tvorbu databázy.

Funkčný a efektívny softvér pomáha podniku k optimálnemu využívaniu zdrojov, šetrí čas, finančné prostriedky, ľudský kapitál a prispieva k celkovému zlepšeniu infraštruktúry organizácie. Ak podnik používa nevhodný softvér, všetky procesy môžu trvať niekoľkonásobne dlhšiu dobu a súčasne dosiahnutý výsledok nemusí byť najlepší. Cieľom tejto práce je oboznámenie sa s kultúrou, fungovaním a činnosťou personálneho oddelenia konkrétneho vybraného podniku, zistenie aktuálneho stavu a spokojnosti s používaným softvérom, zozbieranie a analýza ich požiadaviek na softvér a v neposlednom rade porozumenie organizačnej štruktúre a funkciám jednotlivých zamestnancov oddelenia.

Pre lepšie pochopenie a grafické znázornenie tejto problematiky je užitočné všetky uvedené kroky zobraziť pomocou rôznych modelov a diagramov, ktoré pomôžu k lepšiemu pochopeniu fungovania daného personálneho oddelenia a môžu dokonca viesť k zvýšeniu efektívnosti.

2.2 Metodika práce

Na základe uvedených hlavných a vedľajších cieľov diplomovej práce sme navrhli metodiku, podľa ktorej sme postupovali pri realizácii tejto práce. Táto metodika sa skladá z nasledujúcich krokov:

- oboznámenie sa s problematikou návrhu softvéru,
- výber vhodného podniku,
- kontaktovanie personálneho riaditeľa oddelenia human capital spoločnosti Miba Steeltec s.r.o.,
- získanie údajov o podniku a činnostiach personálneho oddelenia,
- zozbieranie požiadaviek na softvér,

- oboznámenie sa s aktuálne používaným softvérom a jeho výhodami a nedostatkami pre potreby daného personálneho oddelenia,
- analýza získaných údajov,
- výber vhodného programu na modelovanie diagramov,
- vytvorenie organizačného diagramu podniku, detailného organizačného diagramu pre personálne oddelenie, hierarchického diagramu funkcií a relačnej matice podľa organizačnej štruktúry daného podniku a získaných informácií,
- návrh relačnej databázy pomocou SQL, normalizácie, konceptuálneho, logického a fyzického modelu,
- namodelovanie vybraných UML diagramov – diagram tried, use case, stavové a sekvenčné diagramy,
- analýza možných zlepšení.

2.3 Metódy skúmania

V tejto diplomovej práci sme v prvej kapitole využili najmä literárnu rešerš, čo zahŕňalo zozbieranie a naštudovanie odbornej literatúry, ďalej sme použili analýzu, syntézu a komparáciu viacerých modelovacích techník. V tretej kapitole, v rámci návrhu softvéru, čo bolo predmetom tejto diplomovej práce, sme využili viaceré metódy a diagramové techniky, ako napríklad organizačný diagram, hierarchický diagram funkcií, relačná matica, normalizácia, konceptuálny model, logický model, fyzický model, diagram tried, stavový diagram, diagram prípadov použitia a sekvenčný diagram.

2.4 Popis podniku

Pri písaní tejto diplomovej práce sme spolupracovali s podnikom Miba Steeltec, s.r.o. Spoločnosť je súčasťou divízie Miba Friction Group, ktorá vznikla v roku 1927 zásluhou Franza Mitterbauera v rakúskom meste Laakirchen, kde má stále svoje hlavné sídlo. Od svojho vzniku sa Miba Friction Group postupne rozrastala a v roku 1989 začala expandovať aj do zahraničia. O dva roky neskôr, konkrétne v roku 1991, Miba Sintermetall vstúpila do spoločného podniku so slovenským partnerom ZVL v Dolnom Kubíne, vplyvom čoho založila na Slovensku výrobné prevádzky pre aglomerát. V priebehu štyroch rokov sa Miba stala jediným vlastníkom Miba Sinter Slovakia.

V roku 2007 bola založená dcérska spoločnosť Miba Steeltec, s.r.o., so sídlom vo Vrábľoch, ktorá sa stala dôležitou súčasťou skupiny Miba Friction Group. Tento krok znamenal pre spoločnosť ďalší posun v jej rozvoji a posilnenie jej vtedajšej pozície na trhu,

umožňujúc rozšírenie portfólia produktov a služieb. Miba Steeltec, s.r.o. sa špecializuje na výrobu oceľových súčiastok a komponentov pre rôzne odvetvia priemyslu. Svojou kvalitou a inovatívnymi riešeniami si získala dôveru zákazníkov nielen na Slovensku, ale aj v zahraničí.

Spoločnosť Miba AG má celosvetovo rozsiahlu pôsobnosť, pričom k 31. januáru 2024 zamestnávala 7546 kvalifikovaných pracovníkov. Táto spoločnosť je aktívna v desiatich krajinách a je držiteľom 252 patentov, čo svedčí o významnom výskume a vývoji, ktorý realizuje. Okrem toho Miba AG spolupracuje s viac ako štyridsiatimi univerzitami a vedeckými inštitúciami, čo podčiarkuje jej angažovanosť vo výskume a inováciách.

Miba Steeltec, s.r.o. vo Vrábľoch, ako súčasť skupiny Miba AG, zamestnáva 870 ľudí a významne prispieva k miestnemu hospodárstvu. Spoločnosť tiež podporuje duálne vzdelávanie prostredníctvom spolupráce s dvoma miestnymi strednými odbornými školami, čím poskytuje mladým ľuďom príležitosť získať kvalifikáciu a zamestnať sa v priemyselnom sektore. Duálnemu vzdelávaniu sa venujú od roku 2013, pričom každoročne vyprodukujú 8 absolventov maturitných programov mechanik nástrojov a mechanik elektrotechnik. Približne 80 % z uvedených študentov potom zostáva naďalej pracovať v podniku. V súčasnosti program navštevuje 34 študentov.

Miba Steeltec, s.r.o. sa špecializuje na technológie v oblasti trecích obložení pre prevodovky, brzdy a nápravy v banskej technike, stavebných strojoch a traktoroch. Okrem toho sa podnik zaoberá výrobnými procesmi ako je vysekávanie oceľových lamiel, strojné obrábanie, laserové vypaľovanie a výroba nástrojov. Spoločnosť Miba Steeltec, s.r.o. je dôležitým dodávateľom vysokokvalitných oceľových diskov pre stavebné stroje, automobilový a letecký priemysel, čím zohráva významnú úlohu v globálnom dodávateľskom reťazci.

Výrobný závod Miba Steeltec, s.r.o. v Vrábľoch sa rozprestiera na ploche 16 500 m^2 a disponuje špičkovým a moderným vybavením. Jeho hlavné technológie zahŕňajú tepelné spracovanie, povrchové úpravy, mechanické/hydraulické lisy, lasery, drážkovanie, tepelné lisovanie a mechanické obrábanie ako CNC-sústruženie, frézovanie, brúsenie, odvaľovanie, obrážanie a preťahovanie. Tieto technológie umožňujú spoločnosti Miba Steeltec, s.r.o. vyrábať rozmanité výrobky ako oceľové disky, kompozitné disky, molybdénové disky a ďalšie, čím spĺňa potreby zákazníkov na trhu.

Spolupracovali sme primárne s personálnym riaditeľom spoločnosti Miba Steeltec, s.r.o., ktorý nám poskytol prístup k niektorým kľúčovým informáciám, ako je organizačná štruktúra celého podniku, ako aj samotného personálneho oddelenia – human capital. Taktiež nám popísal funkcie pracovníkov na jednotlivých pozíciách jeho oddelenia ako aj prácu so softvérom SAP, ktorý podnik globálne používa na všetkých oddeleniach.

2.5 Aktuálny stav a SAP

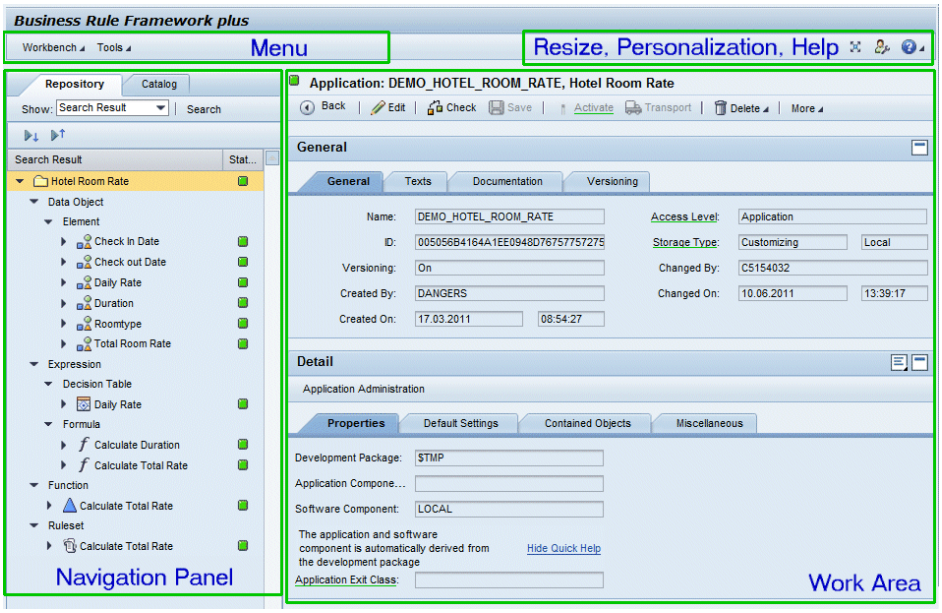
Spoločnosť Miba Steeltec, s.r.o. používa na všetkých svojich oddeleniach softvérový produkt spoločnosti SAP (system, applications and products), čo je jeden z najväčších a najobľúbenejších poskytovateľov softvéru pre podnikové informačné systémy na svete. Používa sa v rôznych oblastiach, napríklad v účtovníctve, výrobnom plánovaní, výrobe, logistike a skladovom hospodárstve, nákupe, riadení ľudských zdrojov, financiách a mnohých ďalších. SAP ponúka viaceré softvérové produkty v závislosti od oblasti pôsobenia podniku, pričom medzi najznámejšie patrí SAP ERP, SAP S/4HANA, SAP Business Network, SAP Business One, SAP Ariba, SAP Signavio a mnohé ďalšie [9, 36].

Medzi konkurenčné výhody používania SAP-u patrí napríklad škálovateľnosť a jednoduché prispôsobenie sa malým aj veľkým podnikom, možnosť komplexných riešení pre všetky aspekty podnikania, prítomnosť rôznych užitočných analytických nástrojov, vysoká spoľahlivosť a výkonnosť a nakoniec jeho flexibilita a jednoduchá prispôsobiteľnosť potrebám a požiadavkám každej organizácie [9].

Keďže každá minca má dve strany a všetko má svoje výhody, rovnako ako aj nevýhody, po konzultácii s riaditeľom personálneho oddelenia a viacerými zamestnancami sme získali ucelený pohľad na ich skúsenosti s používaním systému SAP. Zistili sme, že väčšina zamestnancov je pomerne spokojná s niektorými jeho výhodami, ako je centralizovaná správa údajov, možnosť generovať detailné správy a analytické informácie. Na druhej strane sme identifikovali aj niekoľko problémov a nevýhod, vrátane komplexnosti systému, kvôli čomu sa od používateľov vyžadujú pomerne odborné a zložité znalosti, čo spôsobuje problémy, spomaľuje všedné procesy a zvyšuje náklady na zaškolenie zamestnancov. Rozhranie softvérového produktu od SAP-u, ktorý je používaný na oddelení human capital navyše nie je veľmi užívateľsky prístupný, názvy funkcií sú pomenované laikovi nič nehovoriacimi výrazmi, ťažko sa v ňom orientuje, problematické je aj vytvorenie a export reportov, a pre human resources je teda dosť nepraktický. Ďalšími nevýhodami sú dlhý čas implementácie systému a s tým spojené oneskorenie projektov a plánovaných výsledkov a problémy s prispôsobením sa špecifickým potrebám oddelení, nakoľko niektoré

funkcionality pre konkrétne personálne oddelenie v SAP-e buď nie sú, alebo sa musia navyše vytvárať a dokupovať od externých dodávateľov z dôvodu, že tento systém nie je priamo vytváraný pre potreby personalistiky ani mzdového účtovania a o všetko je potrebné požiadať správcu alebo externého konzultanta. Všetky tieto nedostatky samozrejme zvyšujú aj finančné a časové náklady, čo nie je optimálne.

Z dôvodu, že kvôli ochrane osobných údajov (GDPR) nám nebolo umožnené zobraziť konkrétnu snímku rozhrania softvérového produktu od spoločnosti SAP používaného personálnym oddelením, pre ukážku uvádzame aspoň všeobecné rozhranie zo stránky SAP-u [37].



Obrázok 15: Rozhranie SAP-u [37]

Okrem SAP-u oddelenie human capital používa navyše ešte dochádzkový systém od spoločnosti Kaba, ktorý slúži na evidenciu dochádzky zamestnancov prostredníctvom čítačiek kariet a turniketov, pričom všetky údaje z tohto systému sa na konci každého mesiaca tiež posielajú do SAP-u, kde sú ďalej spracovávané. Ďalej je už asi 3 roky používaný ešte interný softvér mSolution, ktorý rieši digitalizáciu dochádzkového systému, dovoleniek, služobných ciest, elektronických výplatných pások, potvrdení o zdaniteľných príjmoch fyzickej osoby zo závislej činnosti, potvrdení o zaplatení dane z príjmov zo závislej činnosti na účely vyhlásenia o poukázaní sumy do výšky 2 % alebo 3 % zaplatenej dane fyzickej osoby, čo sa potom tiež automaticky prenáša do SAP-u a prostredníctvom neho odosiela zamestnancom.

Tieto zistenia nám umožnili lepšie pochopiť dynamiku používania systému SAP vo vybranej organizácii a identifikovať oblasti, ktoré možno potrebujú ďalšiu pozornosť a vylepšenie. Keďže sme zistili, že program SAP nie je pre činnosť a potreby daného personálneho oddelenia až tak vhodný, rozhodli sme sa vytvoriť návrh softvéru, ktorý by spĺňal všetky jeho požiadavky.

2.6 Požiadavky na softvér

Personálne oddelenie môže mať viaceré požiadavky na softvér, ktorý by mal podporovať jeho prácu. Medzi konkrétne požiadavky oddelenia human capital podniku Miba Steeltec, s.r.o. patrí napríklad prítomnosť funkcií personálnej administratívy ako vedenie a správa zamestnancov, evidencia ich osobných údajov, zamestnaneckých kariet, pracovných hodín a platových informácií. Dôležitá je aj podpora mzdového účtovníctva, ktorá by mala byť jednoduchá a prehľadná a mala by implementovať aktuálnu situáciu a platné zákony v danej krajine. Ďalej by softvér mal umožňovať vytváranie a sledovanie pracovných ponúk, riadenie pracovných pozícií a vyhľadávanie potenciálnych kandidátov. Ďalšou dôležitou požiadavkou je možnosť automatizácie procesov, ako je schvaľovanie dovoleniek, služobných ciest, hodnotenie výkonnosti a správa školení. Okrem toho by softvér mal poskytovať nástroje na monitorovanie a riadenie dodržiavania pracovných zákonov a predpisov týkajúcich sa zamestnaneckých vzťahov. Celkovo by mal softvér pre personálne oddelenie byť flexibilný, užívateľsky prívetivý, intuitívny, praktický a schopný integrácie s existujúcimi systémami a procesmi v podniku.

Medzi funkčné požiadavky na softvér pre podporu personálneho oddelenia patrí napríklad evidencia zamestnancov a všetkých zamestnaneckých údajov, správa pracovných ponúk, riadenie pracovných pozícií, automatizácia procesov, schvaľovanie dovoleniek a služobných ciest, mzdové účtovanie a monitorovanie dodržiavania právnych predpisov. Medzi nefunkčné požiadavky na softvér pre personálne oddelenie môžu patriť bezpečnosť a ochrana údajov, zabezpečenie integrity a dôvernosti, výkonnosť a škálovateľnosť, užívateľská prijateľnosť a jednoduchá manipulácia so systémom, integrácia a interoperabilita s inými softvérmi a dostupnosť technickej podpory, pravidelné aktualizácie a údržba.

3 Výsledky práce a diskusia

Tretia kapitola popisuje konkrétny návrh softvéru na podporu personálneho oddelenia konkrétneho podniku. V úvode tejto kapitoly sa venujeme organizačnej štruktúre oddelenia human capital podniku, ktorý sme analyzovali, ďalej aj jeho funkciám, rozdeleniu, normalizácii, tvorbe konceptuálneho, logického a fyzického modelu a vybraným UML diagramom. Na modelovanie všetkých diagramov sme si zvolili program draw.io, ktorý je pod doménou drawio.com alebo priamo na app.diagrams.net.

3.1 Organizačná štruktúra podniku

Prvým krokom k vytvoreniu akéhokoľvek funkčného softvéru jednoznačne musí byť oboznámenie sa s prostredím, podnikom, jeho oddeleniami, funkciami, ktoré plnia zamestnanci na jednotlivých pozíciách a ich požiadavkami. Z tohto dôvodu bolo potrebné zanalyzovať prostredie spoločnosti Miba Steeltec, s.r.o. a predovšetkým oboznámenie sa s oddelením human capital, ktorým sa táto diplomová práca zaoberá.

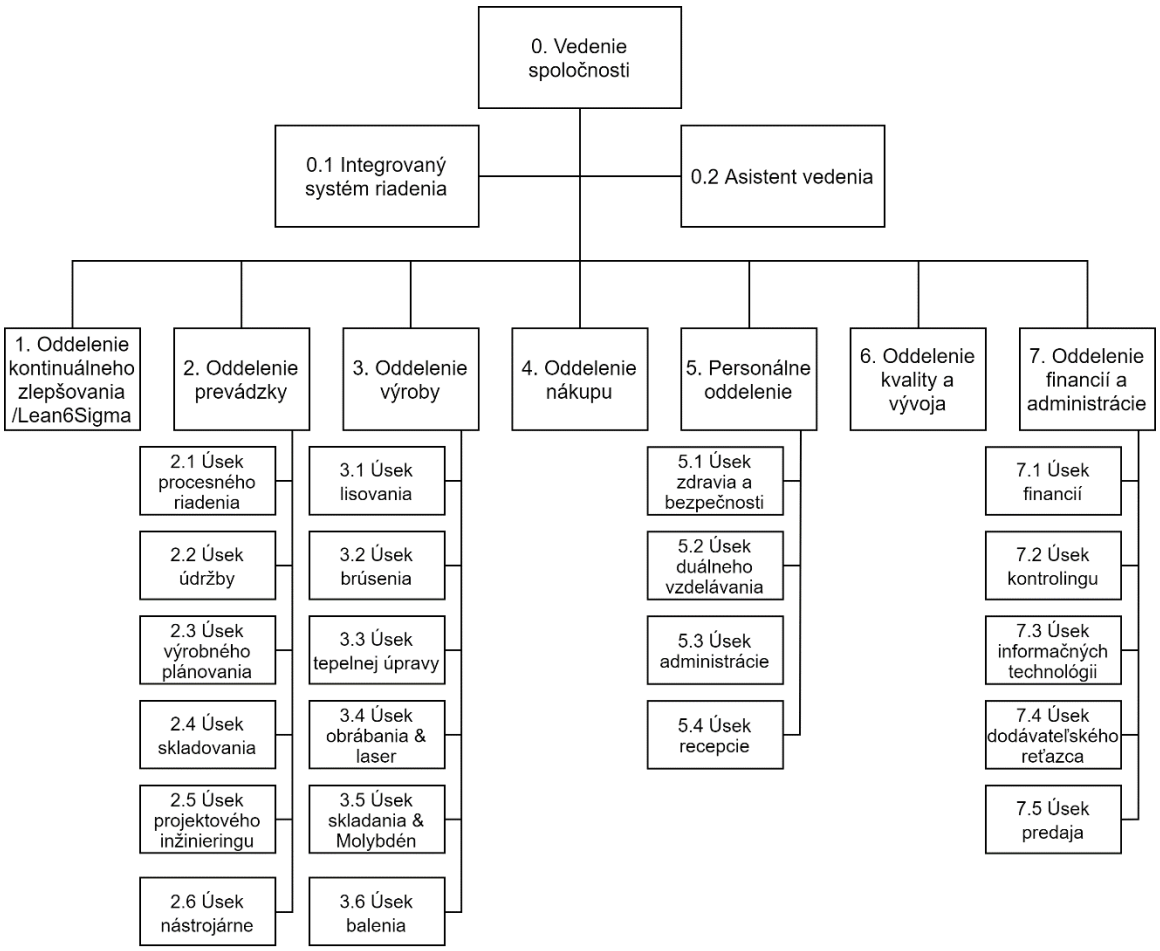
Na základe získaných vedomostí sme vytvorili diagram organizačnej štruktúry celého podniku, ako aj detailný organizačný diagram pre personálne oddelenie. Organizačný diagram zobrazuje organizačnú štruktúru podniku alebo jeho oddelenia, rovnako ako všetky úseky, sekcie, pracovné pozície a vzťahy medzi nimi, pričom pozície, ktoré sú horizontálne na rovnakej úrovni, sú si rovnocenné, zatiaľ čo vertikálne zobrazenie predstavuje vzťah nadradenosti a podradenosti. Jednotlivé organizačné jednotky sú modelované vo forme obdĺžnikov a presným názvom a číslom a prepojené sú pomocou neorientovaných spojníc. Číslovanie prvkov je dôležité kvôli hierarchii, pričom platí zásada, že pozícia generálneho riaditeľa je na nultej úrovni a zvyšné oddelenia sa číslujú postupne od jednotky a sú podriadené nultej úrovni. Ďalšie úseky na nižšej úrovni sa potom číslujú ako 1.1, 1.2, 1.3, čím je znázornená ich vzájomná rovnocennosť a súčasne aj podriadenosť oddeleniu s číslom 1. Ak by mal úsek ešte ďalšie sekcie, boli by označené podobným spôsobom – 1.1.1, 1.1.2 a podobne [20, 21].

Podnik Miba Steeltec, s.r.o. pozostáva zo siedmich oddelení:

1. Oddelenie kontinuálneho zlepšovania/Lean6Sigma
2. Oddelenie prevádzky
3. Oddelenie výroby
4. Oddelenie nákupu
5. Personálne oddelenie

6. Oddelenie kvality a vývoja
7. Oddelenie financií a administrácie.

Okrem vymenovaných oddelení má podnik na nultej úrovni organizácie samozrejme aj vedenie spoločnosti, asistenta vedenia a integrovaný systém riadenia. Organizačnú štruktúru podniku Miba Steeltec, s.r.o. uvádzame na obrázku 16.



Obrázok 16: Organizačný diagram pre podnik Miba Steeltec, s.r.o. *(vlastné spracovanie)*

Po vytvorení organizačného diagramu celého podniku sme sa presunuli k modelovaniu detailnejšieho diagramu organizačnej štruktúry konkrétne pre personálne oddelenie – human capital, ktorý nám pomohol lepšie pochopiť jeho fungovanie. Toto oddelenie nie je klasickým príkladom personálneho oddelenia podniku, pretože okrem klasických funkcií ako je nábor, prijímanie, integrácia a školenie zamestnancov, zahŕňa aj duálne vzdelávanie, vedenie miezd, odvodov, účtovníctva, organizovanie BOZP a rôznych iných školení, rovnako má na starosti aj vstupné, výstupné a periodické lekárske prehliadky, recepciu a v neposlednom rade riaditeľ oddelenia ľudského kapitálu sa osobne podieľa na veľkých firemných rozhodnutiach, ako sú investície, finančný plán, výroba nových

výrobkov alebo nákup strojov, nakoľko všetky dôležité rozhodnutie musí odsúhlasiť aj personálne oddelenie.

Organizačná štruktúra personálneho oddelenia spoločnosti Miba Steeltec, s.r.o. sa skladá zo štyroch úsekov, ktoré podliehajú personálnemu riaditeľovi, a na každom úseku je niekoľko pracovných pozícií:

5.1 Úsek zdravia a bezpečnosti

5.1.1 Manažér pre bezpečnosť

5.1.2. Technik bezpečnosti

5.2 Úsek duálneho vzdelávania

5.2.1 Senior inštruktor duálneho vzdelávania

5.2.2 Junior inštruktor duálneho vzdelávania

5.3 Úsek administrácie

5.3.1 Senior špecialista pre mzdy

5.3.2 Špecialista administrácie

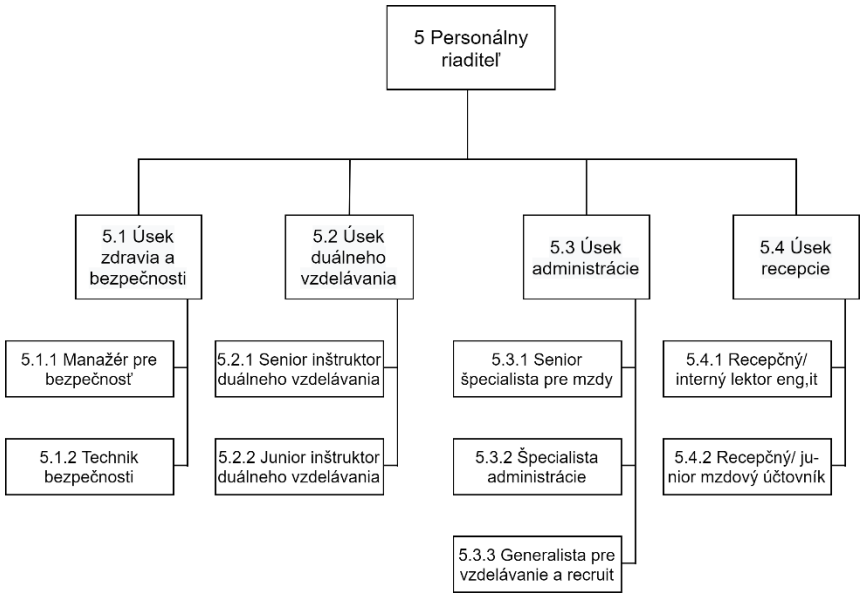
5.3.3 Generalista pre vzdelávanie a recruit

5.4 Úsek recepcie

5.4.1 Recepčný/interný lektor angličtiny a taliančiny

5.4.2 Recepčný/junior mzdový účtovník

Keďže chceme, aby mali organizačné diagramy pre celý podnik a detailný diagram pre oddelenie ľudského kapitálu čo najväčšiu výpovednú hodnotu, musia byť vzájomne konzistentné, takže počet, názvy a číslovanie jednotlivých úsekov sa musia zhodovať. Grafické znázornenie organizačnej štruktúry personálneho oddelenia spoločnosti Miba Steeltec, s.r.o. uvádzame na obrázku 17.



***Obrázok 17: Organizačný diagram pre personálne oddelenie Miba Steeltec, s.r.o.** (vlastné spracovanie)*

3.2 Funkcie a činnosti personálneho oddelenia

Ako sme už spomínali v predchádzajúcej kapitole, každý zamestnanec plní na oddelení rôzne funkcie. V organizačnom diagrame sme uviedli jednotlivé úseky a pozície personálneho oddelenia a v tejto časti sme ku každej z nich zistili a zadefinovali ich hlavné a vedľajšie funkcie, právomoci, povinnosti a činnosti, ktorými sa zaoberajú.

Úsek 5.1 vykonáva riadenie bezpečnosti a medzi jeho funkcie patrí napríklad organizovanie pravidelných zákonných školení ako bezpečnosť a ochrana zdravia pri práci (BOZP), kurzy prvej pomoci, vozíčkarské školenia, školenia požiarnej ochrany, školenia pre novoprijatých zamestnancov, rovnako ako aj samotné vedenie školení BOZP, pričom asi 50 % zabezpečuje navyše aj externá firma, a vedenie evidencie. Zamestnanci tohto úseku ďalej tiež vykonávajú dohľad nad bezpečnosťou pri práci, robia pravidelné kontroly a inšpekcie pracovísk s dôrazom na dodržiavanie bezpečnostných predpisov. Ďalšou funkciou je príprava a vykonávanie plánov krízového riadenia ako aj rýchle a efektívne riadenie nehôd a havárií, ak sa vyskytnú, a ich následné nahlásenie a s tým spojená administratíva. V neposlednom rade zabezpečujú vývoj a implementáciu bezpečnostných politík podniku.

Druhý úsek oddelenia human capital zastrešuje riadenie duálneho vzdelávania a medzi jeho funkcie patrí najmä samotné vyučovanie pracovných postupov a procesov, ktoré sú špecifické pre daný študijný program, spolu s prípravou zodpovedajúcich študijných materiálov a osnov, hodnotenie študentov na základe ich výsledkov a s tým spojená

komunikácia so strednou školou. Zamestnanci tohto oddelenia musia taktiež komunikovať aj s výrobným oddelením a manažmentom, aby sa zabezpečilo, že potreby študentov sú zaradené do pracovných postupov a výrobných procesov, výsledkom čoho študenti môžu dosiahnuť čo najlepšie výsledky a získať maximum skúseností. Úlohou inštruktorov duálneho vzdelávania je tiež podpora, mentorstvo a poskytovanie poradenstva študentom a rovnako aj vedenie evidencie a administratívy o študentoch, ich osobných údajoch, hodnoteniach, dochádzke a podobne.

Úsek 5.3 je zodpovedný za riadenie administrácie a vykonáva činnosti ako napríklad mzdové účtovanie, čo zahŕňa výpočet hrubých a čistých miezd zamestnancov na základe ich odpracovaných hodín, platovej triedy, odmien, príplatkov, dovoleniek, odvodov a iných faktorov, rovnako ako aj tvorbu výplatných pások a samotné vyplácanie miezd. Tento úsek sa zaoberá aj výpočtom a úhradou odvodov, spravuje dovolenky a odpočty, aby boli správne zaznamenané a odpočítané zo mzdy. Ďalším bodom je výpočet dane a tvorba daňového priznania, vykonávanie auditu spojené s kontrolou a evidencia platových údajov v databáze. Úsek administrácie taktiež plní funkcie ako inzerovanie a „pohovorovanie“ uchádzačov o zamestnanie, čo zahŕňa aktívne vyhľadávanie a rekrutáciu kvalifikovaných kandidátov na voľné pracovné pozície v podniku prostredníctvom inzerátov, pracovných veľtrhov a komunikáciu na online platformách a následný výber vhodných adeptov, vedenie nástupného procesu spojené so zaškolením, zoznámením s organizačnou štruktúrou a kultúrou, vypracovanie nástupných aj predlžovacích zmlúv, platových dekrétov, valorizácia a ďalšie administratívne úkony. Úlohou tretieho úseku je v neposlednom rade aj plánovanie, evidencia a vedenie všetkých lekárskych prehliadok a rôznych školení, aby sa zabezpečilo, že každý zamestnanec pravidelne absolvuje všetky potrebné vyšetrenia, na ktoré má zo zákona nárok a zúčastní sa potrebných kurzov. Poslednou funkciou je správa výkonnosti zamestnancov a rozvojových plánov, čo pomáha k identifikácii ich silných stránok a oblastí, ktoré vyžadujú zlepšenie a následné vypracovanie individuálnych rozvojových plánov.

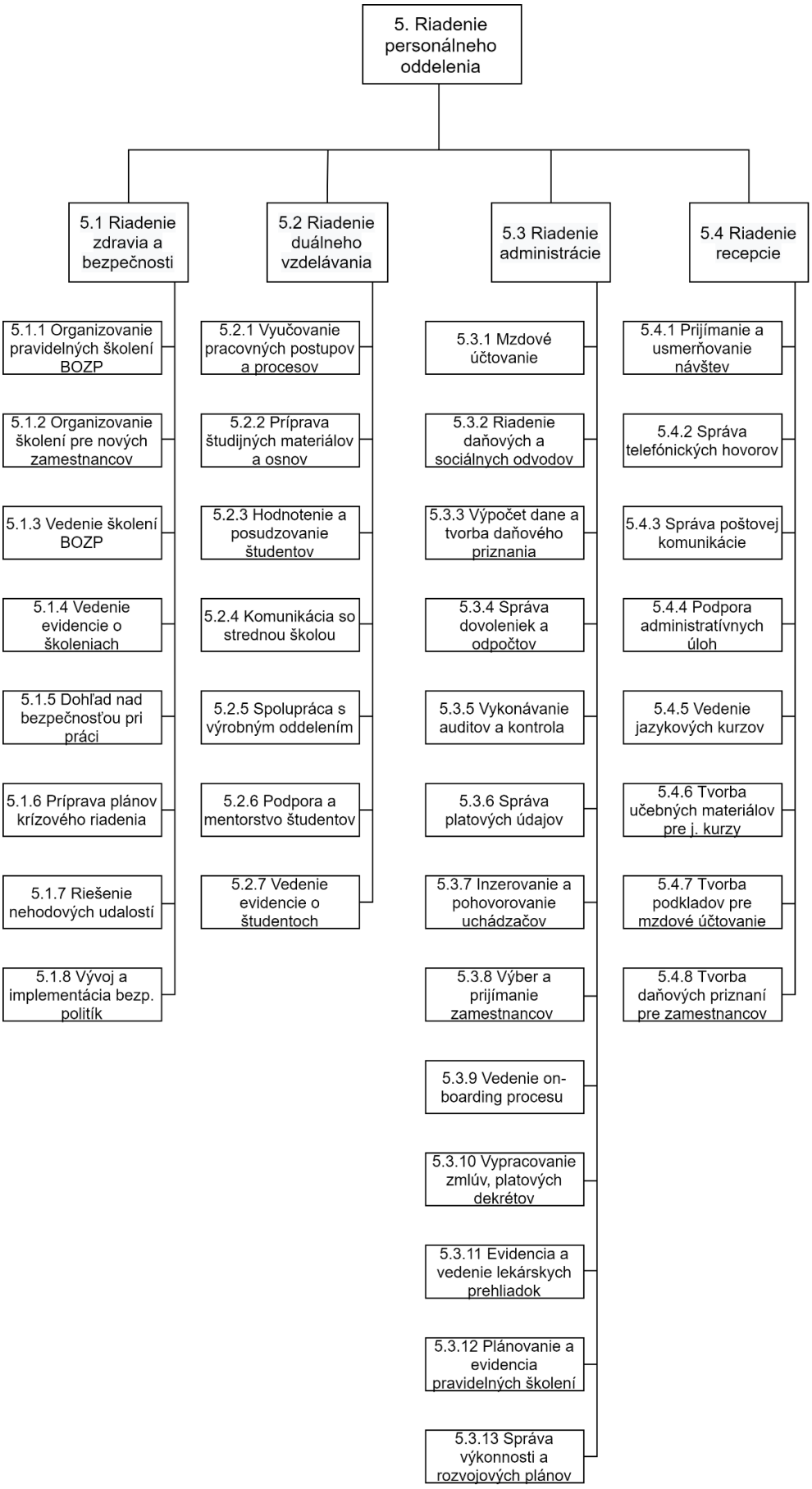
Posledný úsek personálneho oddelenia podniku Miba Steeltec, s.r.o. poskytuje funkcie spojené s riadením recepcie, a to konkrétne prijímanie a usmerňovanie návštev, správa telefonických hovorov a poštovej či mailovej komunikácie a ich prípadné presmerovanie alebo usmernenie na adekvátneho zamestnanca a takisto podpora pri rôznych administratívnych úlohách ako je aktualizovanie údajov v databázach, vytváranie a správa dokumentov a iné. Zamestnanci na úseku 5.4 majú zdvojené úväzky a do ich kompetencie okrem povinností recepčného navyše spadá aj vedenie jazykových kurzov anglického

a talianskeho jazyka spolu s tvorbou príslušných učebných materiálov a príprava podkladov pre mzdové účtovanie a príprava daňových priznaní pre zamestnancov, pri čom je potrebná spolupráca s tretím úsekom.

Všetky štyri úseky podliehajú riaditeľovi oddelenia human capital, ktorého prácou je plánovanie a implementácia personálnych stratégií, výber zamestnancov, vedenie a rozvoj tímu v rámci oddelenia, plánovanie a realizácia teambuildingových aktivít a samozrejme aj riadenie výkonnosti, kontrola a hodnotenie zamestnancov. Okrem toho úzko spolupracuje s manažmentom podniku a ako zástupca personálneho oddelenia sa trochu netradične osobne podieľa aj na všetkých dôležitých rozhodnutiach ohľadom investovania, nákupu nových strojov, finančného plánu, podnikovej stratégie, výroby a mnohých iných. Tieto funkcie sa obyčajne do diagramu priamo nevpisujú, pretože sú obsiahnuté pod všeobecným termínom „riadenie“.

Podnikové funkcie sa zväčša zakresľujú do hierarchického diagramu funkcií, kde je zároveň aj graficky znázornené, ktorého oddelenia sa dané funkcie týkajú. Tento diagram poskytuje prehľad všetkých funkcií v podniku, pričom jednotlivé funkcie sú usporiadané hierarchicky na základe dôležitosti a vzájomných vzťahov medzi jednotlivými funkciami, ktoré si môžu byť nadradené, rovnocenné alebo podradené iným funkciám v rámci organizácie. Jednotlivé funkcie sú rovnako ako pri organizačnom diagrame zakreslené pomocou obdĺžnikov a sú spojené neorientovanými spojnicami, pričom každý obdĺžnik sa skladá z názvu funkcie, ktorá musí byť pomenovaná ako činnosť, a musí byť tiež riadne očíslovaná, pričom platí, že funkcie nultej a prvej úrovne diagramu sa označujú ako „riadenie“ niečoho a číslovanie je založené na rovnakom princípe ako pri diagrame organizačnej štruktúry. Pravidlom je, že funkcií v hierarchickom diagrame funkcií by malo byť vždy viac ako je jednotiek v organizačnom diagrame, nakoľko zamestnanec alebo celý úsek poväčšine nevykonáva len jednu činnosť, ale má ich na starosti viac [20, 21].

Hierarchický diagram funkcii pre personálne oddelenie podniku Miba Steeltec, s.r.o. uvádzame na obrázku 18.



Obrázok 18: Hierarchický diagram funkcií pre personálne oddelenie Miba Steeltec, s.r.o. (vlastné spracovanie)

3.3 Zobrazenie vzťahov a rozdelenie funkcií

Ako bolo uvedené v predchádzajúcej kapitole, hierarchický diagram funkcií popisuje iba vzťahy medzi funkciami, no nedá sa z neho zistiť, ktorý organizačný útvar ich vykonáva. Závislosť medzi funkciami a ich vykonávateľmi sa môže zobrazovať napríklad pomocou relačnej matice, ktorá sa používa na priradenie funkcií jednotlivým útvarom a pracovným pozíciám, či konkrétnym zamestnancom z organizačnej štruktúry. Táto tabuľka teda umožňuje prepojiť organizačný diagram s hierarchickým diagramom funkcií a prispieť tak k ich komplexnejšiemu porozumeniu a efektívnejšiemu využitiu. Údaje sú usporiadané vo forme tabuľky, pričom stĺpce znázorňujú jednotlivé útvary organizačnej štruktúry, v našom prípade úseky a pozície na personálnom oddelení, a riadky predstavujú všetky funkcie, ktoré na danom oddelení prebiehajú. Na ich priesečníky sa potom vpisujú hodnoty „R“, „K“, „V“ alebo zostávajú prázdne. Do priesečníka útvaru a funkcie, ktorú riadi sa píše „R“, ak jej vykonanie len kontroluje, vyplní sa „K“ a ak je priamo zodpovedný za jej vykonanie, doplní sa „V“. Riadenie je širší pojem ako kontrola, takže riadiaci útvar okrem kontroly vykonania funkcie aj vydáva rozkazy, požiadavky a usmernenia. Relačná matica spravidla nesmie obsahovať ani jeden prázdny riadok, či stĺpec, pretože to by znamenalo, že nejaká funkcia nie je nikým vykonávaná alebo naopak, že určité oddelenie nič nerobí, a je teda pre daný podnik zbytočné [20, 21].

Relačnú maticu pre analyzované personálne oddelenie sme teda vytvorili tak, že do stĺpcov sme vypísali všetky útvary a do riadkov všetky prebiehajúce funkcie na oddelení a postupne sme vypĺňali hodnoty do jednotlivých priesečníkov so zreteľom na vyššie uvedené pravidlá a normy. Útvary priamo zodpovedné za vykonanie určitej činnosti dostali hodnotu „V“, zatiaľ čo kontrolujúcim jednotkám bola pridelená hodnota „K“. Útvar 5.0 priamo ovplyvňuje a riadi prvé tri úseky, kvôli čomu mu pri nich bola priradená hodnota „R“, no úsek recepcie iba kontroluje, keďže jeho činnosť nie je výhradne náplňou personálneho oddelenia, len ju zastrešuje, a kvôli tomu sme pri riadení recepcie zvolili hodnotu „K“. Jedným z pravidiel tvorby relačných matíc je, že každú funkciu vykonáva vždy práve jeden útvar, a teda každý riadok by mal obsahovať iba jedno „V“, avšak nakoľko sa na personálnom oddelení na úseku recepcie nachádzajú pozície so zdvojenými úväzkami, znamená to, že recepčné majú okrem svojich klasických povinností na starosti aj vedenie jazykových kurzov, respektíve mzdové účtovníctvo, z čoho vyplýva, že toto pravidlo je v tomto prípade porušené a funkcie týkajúce sa recepcie sú vykonávané oboma týmito pozíciami.

Relačná matica je veľmi užitočným nástrojom pri plánovaní zmien v podniku, riadení rizík alebo pri optimalizácii procesov, nakoľko zobrazuje vzťahy medzi jednotlivými útvarmi a tým, ako sa vzájomne ovplyvňujú. Namodelovanú relačnú maticu pre personálne oddelenie spoločnosti Miba Steeltec, s.r.o. uvádzame na obrázku 19.

Útvar / Funkcia	5	5.1	5.1.1	5.1.2	5.2	5.2.1	5.2.2	5.3	5.3.1	5.3.2	5.3.3	5.4	5.4.1	5.4.2
5	V													
5.1	R	V												
5.1.1		K	V											
5.1.2		K	V											
5.1.3		K		V										
5.1.4		K	V											
5.1.5		K		V										
5.1.6		K	V											
5.1.7		K		V										
5.1.8		K	V											
5.2	R				V									
5.2.1					K	V								
5.2.2					K	V								
5.2.3					K	V								
5.2.4					K	V								
5.2.5					K		V							
5.2.6					K		V							
5.2.7					K		V							
5.3	R							V						
5.3.1								K	V					
5.3.2								K	V					
5.3.3								K	V					
5.3.4								K	V					
5.3.5								K	V					
5.3.6								K	V					
5.3.7								K			V			
5.3.8								K			V			
5.3.9								K		V				
5.3.10								K		V				
5.3.11								K		V				
5.3.12								K		V				
5.3.13								K			V			
5.4	K											V		
5.4.1												K	V	V
5.4.2												K	V	V
5.4.3												K	V	V
5.4.4												K	V	V
5.4.5												K	V	
5.4.6												K	V	
5.4.7												K		V
5.4.8												K		V

Obrázok 19: Relačná matica pre personálne oddelenie Miba Steeltec, s.r.o. *(vlastné spracovanie)*

3.4 Návrh databázy a normalizácia

Pri vytváraní návrhu softvéru pre akýkoľvek podnik je nevyhnutná aj tvorba relačnej databázy, ktorá umožňuje štruktúrované ukladanie dát do tabuliek s definovanými vzájomnými vzťahmi, zabezpečuje integritu dát, umožňuje efektívne vyhľadávanie a manipuláciu s dátami prostredníctvom štruktúrovaných jazykov ako SQL, a súčasne sa používa aj na zdieľanie dát. Správne vytvorená databáza by mala spĺňať niekoľko vlastností, ako napríklad minimalizácia redundancie, zachovanie integrity dát, vysoká efektívnosť pri vyhľadávaní a manipulácii s dátami a ľahká údržba. Na dosiahnutie týchto vlastností sa používa normalizácia. Cieľom normalizácie je odstrániť opakujúce sa dáta a zabezpečiť, aby každá informácia bola uložená na jednom mieste. Tento proces zvyšuje efektivitu databázy a uľahčuje údržbu a manipuláciu s dátami. Normalizácia sa vykonáva sériou krokov, ktoré postupne rozdelia dáta do menších, lepšie štruktúrovaných súborov, ktoré sú logicky vzájomne prepojené. To vedie k lepšiemu porozumeniu dát a zlepšuje výkonnosť a spoľahlivosť databázy. Použitím normalizácie sa dosahuje lepšia organizácia dát, čo v konečnom dôsledku zlepšuje celkovú kvalitu a využiteľnosť systému.

V nasledujúcej časti uvádzame postupné kroky normalizácie od nultej až po tretiu normálnu formu v lineárnom textovom zápise s použitím pravidiel, opísaných v kapitole 1.7 tejto práce. Pri procese normalizácie sme teda vykonávali zmeny v atribútoch a entitách tak, aby vždy spĺňali požiadavky pre danú normálnu formu. Jednotlivé zmeny sme zaznačili tak, že vždy v tých dvoch normálnych formách, ktorých sa zmeny týkajú sú normalizované údaje (entity alebo atribúty) zvýraznené tou istou farbou.

V nultej normálnej forme sú vypísané všetky entity a ich atribúty, ktorých údaje bude potrebné uchovávať a s ktorými bude informačný softvér personálneho oddelenia pracovať. Vytvorili sme 5 entít (vyznačené hrubým písmom), a to zamestnanec, zmluva, skolenie, pozicia a uchadzac a ku každej entite sme vypísali jej atribúty s tým, že jednotlivé entity sa prepájajú cez primárne a cudzie kľúče. Primárny kľúč je pri atribúte vyznačený symbolom # a podčiarknutím, zatiaľ čo cudzí kľúč obsahuje len znak #.

0NF

zamestnanec (ID_zamestnanec#, ID_zmluva#, ID_pozicia#, ID_skolenie#, meno, priezvisko, pohlavie, datum_narodenia, rodne_cislo, adresa, kontaktne_udaje, datum_nastupu, cislo_kancelarie, stav_zamestnanca)

zmluva (ID_zmluva#, typ_pracovneho_pomeru, doba_platnosti_zmluvy, nastupna_mzda, IBAN, stav_zmluvy)

skolenie (ID_skolenie#, typ_skolenia, trvanie_skolenia, instruktor, miesto, stav_skolenia)

pozicia (ID_pozicia#, nazov_pozicie, poziadavka_na_poziciu, nazov_oddelenia, pocet_zamestnancov_oddelenia, meno_veduceho, datum_zverejnenia_inzeratu, stav_pozicie)

uchadzac (ID_uchadzac#, ID_pozicia#, meno, priezvisko, datum_narodenia, adresa, kontaktne_udaje, najvyssie_dosiahnute_vzdelanie, stav_uchadzaca)

Prvá normálna forma (1NF) si vyžaduje mať všetky atribúty atomické, čo znamená, že jednotlivé atribúty nemôžu obsahovať zložené hodnoty, aby sa zaručilo, že jednotlivé stĺpce budú obsahovať vždy len hodnoty z homogénnej dátovej domény (t. j. jedno políčko - jedna hodnota). Z nultej normálnej formy túto podmienku nespĺňali atribúty adresa a kontaktne_udaje v entitách zamestnanec a uchadzac, doba_platnosti_zmluvy v entite zmluva a atribút trvanie_skolenia v entite skolenie. Aby sme odstránili neatomické atribúty, rozbili sme atribút adresa na ulica, supisne_cislo a mesto, kontaktne_udaje na telefonne_cislo a e-mail, doba_platnosti_zmluvy na datum_uzatvorenia a datum_ukoncenia a trvanie_skolenia na datum_zaciatku_skolenia a datum_konca_skolenia.

1NF

zamestnanec (ID_zamestnanec#, ID_zmluva#, ID_pozicia#, ID_skolenie#, meno, priezvisko, pohlavie, datum_narodenia, rodne_cislo, ulica, supisne_cislo, mesto, telefonne_cislo, e-mail, datum_nastupu, cislo_kancelarie, stav_zamestnanca)

zmluva (ID_zmluva#, typ_pracovneho_pomeru, datum_uzatvorenia, datum_ukoncenia, nastupna_mzda, IBAN, stav_zmluvy)

skolenie (ID_skolenie#, typ_skolenia, datum_zaciatku_skolenia, datum_konca_skolenia, instruktor, miesto, stav_skolenia)

pozicia (ID_pozicia#, nazov_pozicie, poziadavka_na_poziciu, nazov_oddelenia, pocet_zamestnancov_oddelenia, meno_veduceho, datum_zverejnenia_inzeratu, stav_pozicie)

uchadzac (ID_uchadzac#, ID_pozicia#, meno, priezvisko, datum_narodenia, ulica, supisne_cislo, mesto, telefonne_cislo, e-mail, najvyssie_doasiahnute_vzdelanie, stav_uchadzaca)

Druhá normálna forma (2NF) vyžaduje, aby všetky nekľúčové hodnoty v riadku boli významovo úplne závislé od kľúčových hodnôt daného riadka. Túto podmienku porušovali atribúty poziadavka_na_poziciu, nazov_oddelenia, pocet_zamestnancov_oddelenia,

meno_veduceho a datum_zverejnenia_inzeratu v entite pozicia, lebo napríklad atribút meno_veduceho nie je závislý od kľúčovej hodnoty ID_pozicia# ale od oddelenia. Aby sme tento problém odstránili, vytvorili sme nové entity oddelenie, inzerat, poziadavka_na_poziciu a poziadavky, ktoré sme prepojili pomocou primárnych a cudzích kľúčov.

2NF

zamestnanec (ID_zamestnanec#, ID_zmluva#, ID_pozicia#, ID_skolenie#, meno, priezvisko, pohlavie, datum_narodenia, rodne_cislo, ulica, supisne_cislo, mesto, telefonne_cislo, e-mail, datum_nastupu, cislo_kancelarie, stav_zamestnanca)

zmluva (ID_zmluva#, typ_pracovneho_pomeru, datum_uzatvorenia, datum_ukoncenia, nastupna_mzda, IBAN, stav_zmluvy)

skolenie (ID_skolenie#, typ_skolenia, datum_zaciatku_skolenia, datum_konca_skolenia, instruktor, miesto, stav_skolenia)

pozicia (ID_pozicia#, ID_oddelenie#, nazov_pozicie, stav_pozicie)

oddelenie (ID_oddelenie#, nazov_oddelenia, pocet_zamestnancov, meno_veduceho)

poziadavka_na_poziciu (ID_pozicia#, ID_poziadavka#, splnenie_ano/nie)

poziadavky (ID_poziadavka#, nazov_poziadavky)

inzerat (ID_inzerat#, ID_pozicia#, datum_zverejnenia, stav_inzeratu)

uchadzac (ID_uchadzac#, ID_inzerat#, meno, priezvisko, datum_narodenia, ulica, supisne_cislo, mesto, telefonne_cislo, e-mail, najvyssie_dosiahnute_vzdelanie, stav_uchadzaca)

Tretia normálna forma (3NF) rieši problém tzv. tranzitívnej závislosti, čo znamená, že každá nekľúčová hodnota nesmie závisieť od inej nekľúčovej hodnoty v danom riadku, teda každý atribút je buď kľúčovým atribútom, alebo je svojou hodnotou závislý od celého kľúča (všetkých kľúčových hodnotách). V 2NF boli atribúty instruktor a miesto tranzitívne závislé od atribútu typ_skolenia, a preto bolo potrebné vytvoriť novú entitu typ_skolenia, ktorý sme prepojili s entitou školenie prostredníctvom cudzieho kľúča ID_typ_skolenia#.

3NF

zamestnanec (ID_zamestnanec#, ID_zmluva#, ID_pozicia#, ID_skolenie#, meno, priezvisko, pohlavie, datum_narodenia, rodne_cislo, ulica, supisne_cislo, mesto, telefonne_cislo, e-mail, datum_nastupu, cislo_kancelarie, stav_zamestnanca)

zmluva (ID_zmluva#, typ_pracovneho_pomeru, datum_uzatvorenia, datum_ukoncenia, nastupna_mzda, IBAN, stav_zmluvy)

skolenie (ID_skolenie#, ID_typ_skolenia#, datum_zaciatku_skolenia, datum_konca_skolenia, stav_skolenia)

typ_skolenia (ID_typ_skolenia #, nazov_typ_skolenia, instruktor, miesto)

pozicia (ID_pozicia#, ID_oddelenie#, nazov_pozicie, stav_pozicie)

oddelenie (ID_oddelenie#, nazov_oddelenia, pocet_zamestnancov, meno_veduceho)

poziadavka_na_poziciu (ID_pozicia#, ID_poziadavka#, splnenie_ano/nie)

poziadavky (ID_poziadavka#, nazov_poziadavky)

inzerat (ID_inzerat#, ID_pozicia#, datum_zverejnenia, stav_inzeratu)

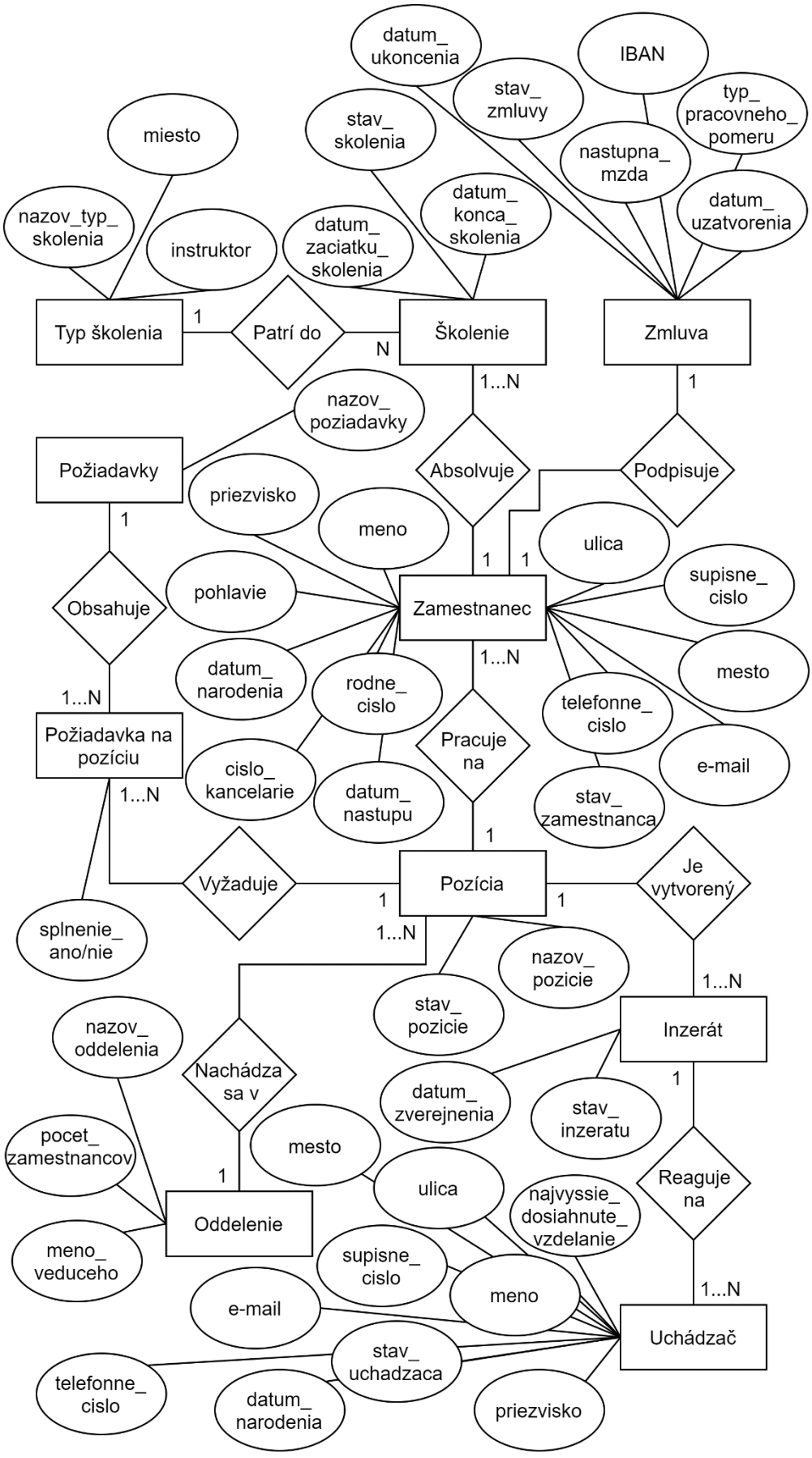
uchadzac (ID_uchadzac#, ID_inzerat#, meno, priezvisko, datum_narodenia, ulica, supisne_cislo, mesto, telefonne_cislo, e-mail, najvyssie_dosiahnute_vzdelanie, stav_uchadzaca)

3.5 Konceptuálny, logický a fyzický model

Keďže vďaka normalizácii sme už mali dáta organizované, mohli sme ďalej prejsť k tvorbe konceptuálneho, logického a fyzického modelu, ktoré sú využívané pri návrhu softvéru v oblasti databázového návrhu a sú tiež veľmi nápomocné a užitočné. Konceptuálny model nemá definované, v ktorej normálnej forme majú byť dáta uložené, no v logickom a fyzickom modeli musia všetky dáta byť v tretej normálnej forme. Tieto tri modely sú používané na definovanie štruktúry a organizácie dát v softvérových aplikáciách a pomáhajú softvérovým architektom a vývojárom pri plánovaní a implementácii dobre štruktúrovaných databázových systémov.

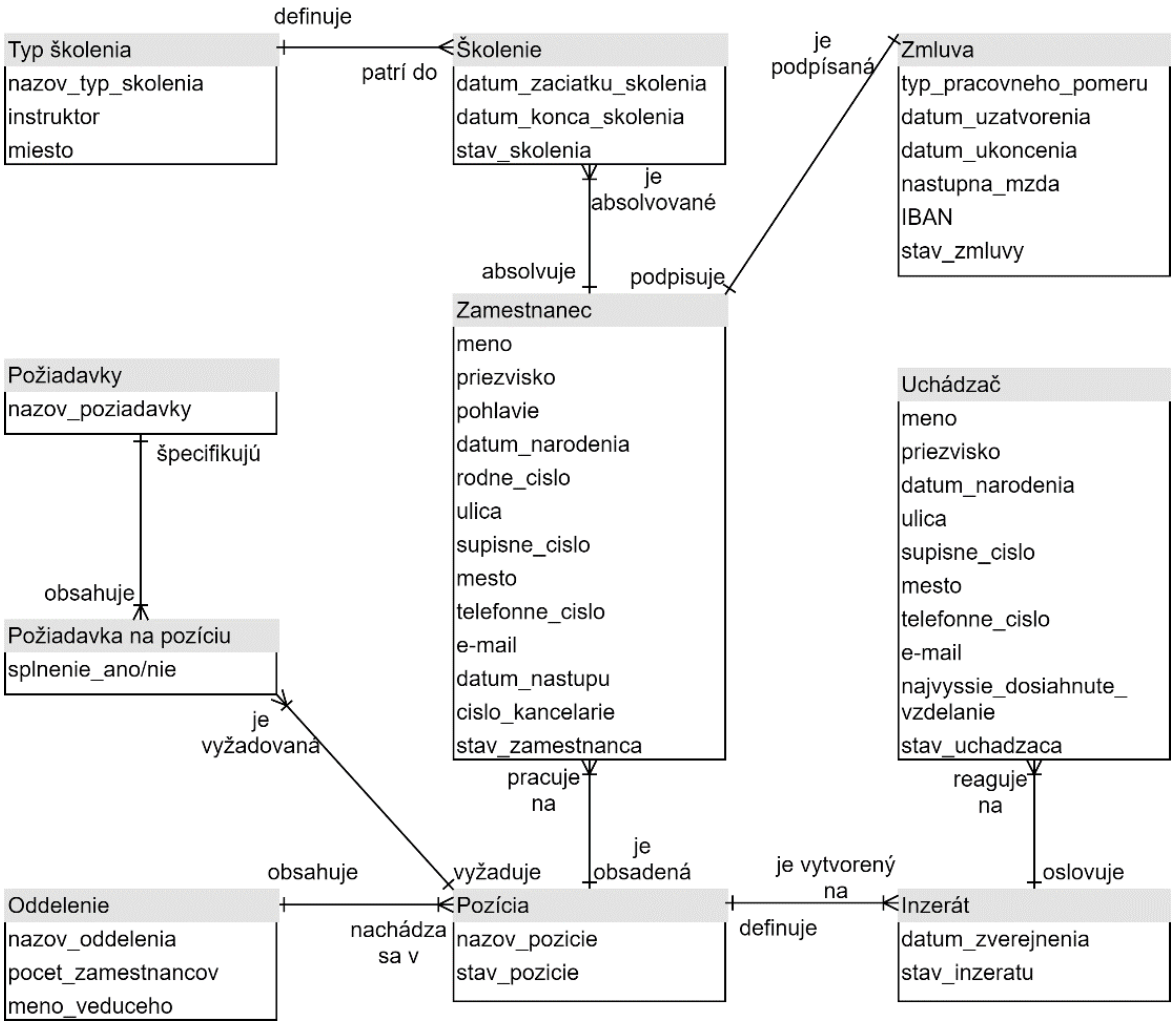
Konceptuálny model je z uvedených troch modelov najjednoduchší a poskytuje abstraktný pohľad na štruktúru dát a vzťahy medzi nimi a môže sa vytvoriť vo forme ERD (Entitno-relačný diagram).Tento model je nezávislý od implementačných detailov a pomáha najmä pri identifikácii hlavných entít a vzťahov medzi nimi, zobrazuje kardinalitu, ale nezobrazuje ešte fyzickú štruktúru databázy.

Na tvorbu konceptuálneho modelu sme použili dve z najpoužívanejších notácií a to Chenovu notáciu a Crow's foot notáciu. Chenov formát, ktorý je pomenovaný po jeho zakladateľovi Petrovi Chenovi, používa ako označenie pre entitu obdĺžnik, atribúty sú zaznačené do symbolu elipsa, zatiaľ čo relácie (vzťahy) majú symbol kosoštvorca. Na znázornenie výskytu entít navzájom sa používa kardinalita (násobnosť) typu M:N. Konceptuálny model vytvorený s použitím Chenovej notácie je zobrazený na obrázku 20 [8, 28].



***Obrázok 20: Konceptuálny model pre personálne oddelenie Miba Steeltec, s.r.o. – Chenova notácia** (vlastné spracovanie)*

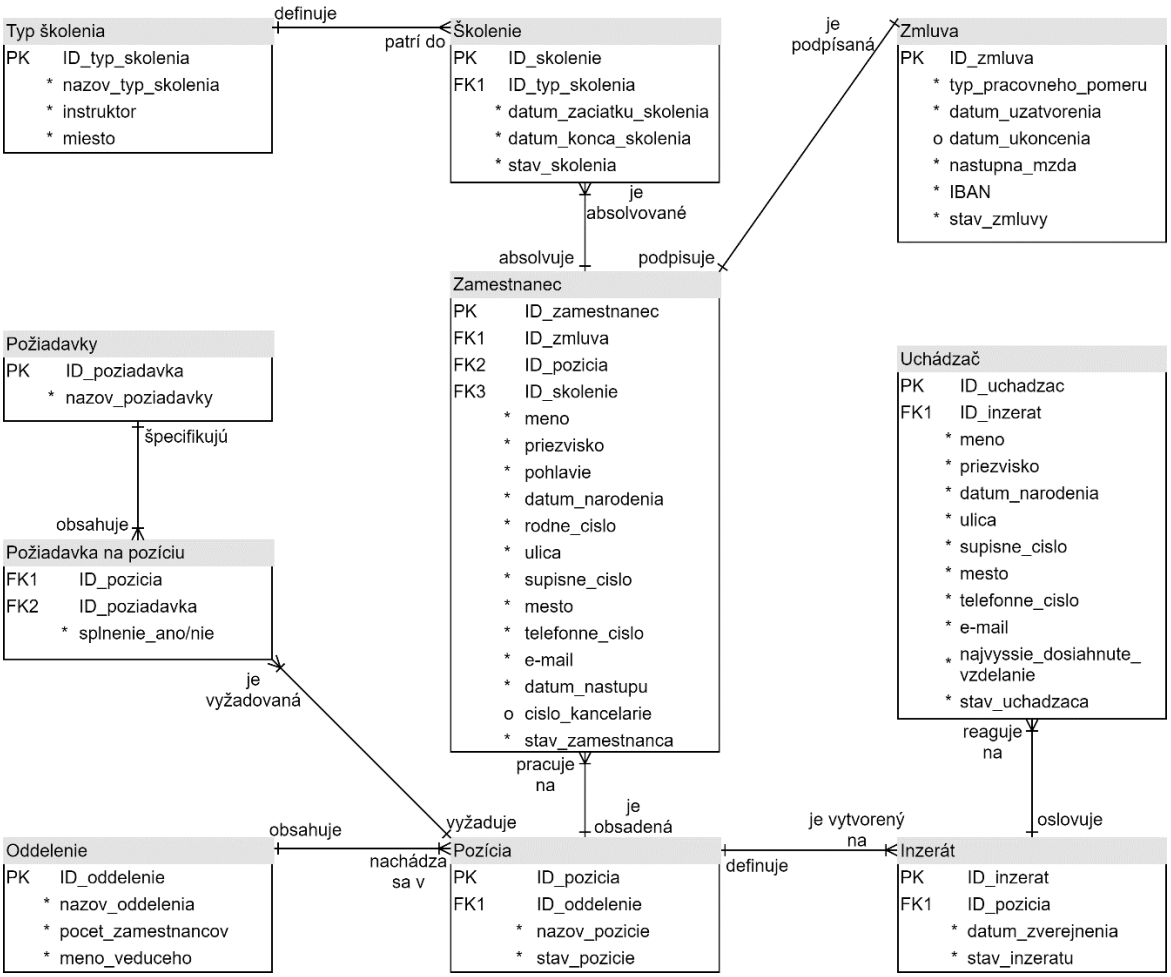
Keďže Chenova notácia modelovania diagramov je priestorovo veľmi rozsiahla a môže pôsobiť chaoticky a neprehľadne, lebo každý atribút a relácia sa zapisujú zvlášť, nie je pri zložitejších modeloch až tak používaná a využíva sa skôr relačný formát diagramov a Crow's foot notácia. Diagramy v relačnom formáte majú atribúty zapísané priamo v príslušných entitách. Konceptuálny model vytvorený týmto spôsobom pôsobí oveľa prehľadnejšie a logický s fyzickým modelom naň môžu jednoducho nadväzovať. Konceptuálny model vytvorený pomocou Crow's foot notácie je uvedený na obrázku 21.



Obrázok 21: Konceptuálny model pre personálne oddelenie Miba Steeltec, s.r.o. – Crow's foot notácia (vlastné spracovanie)

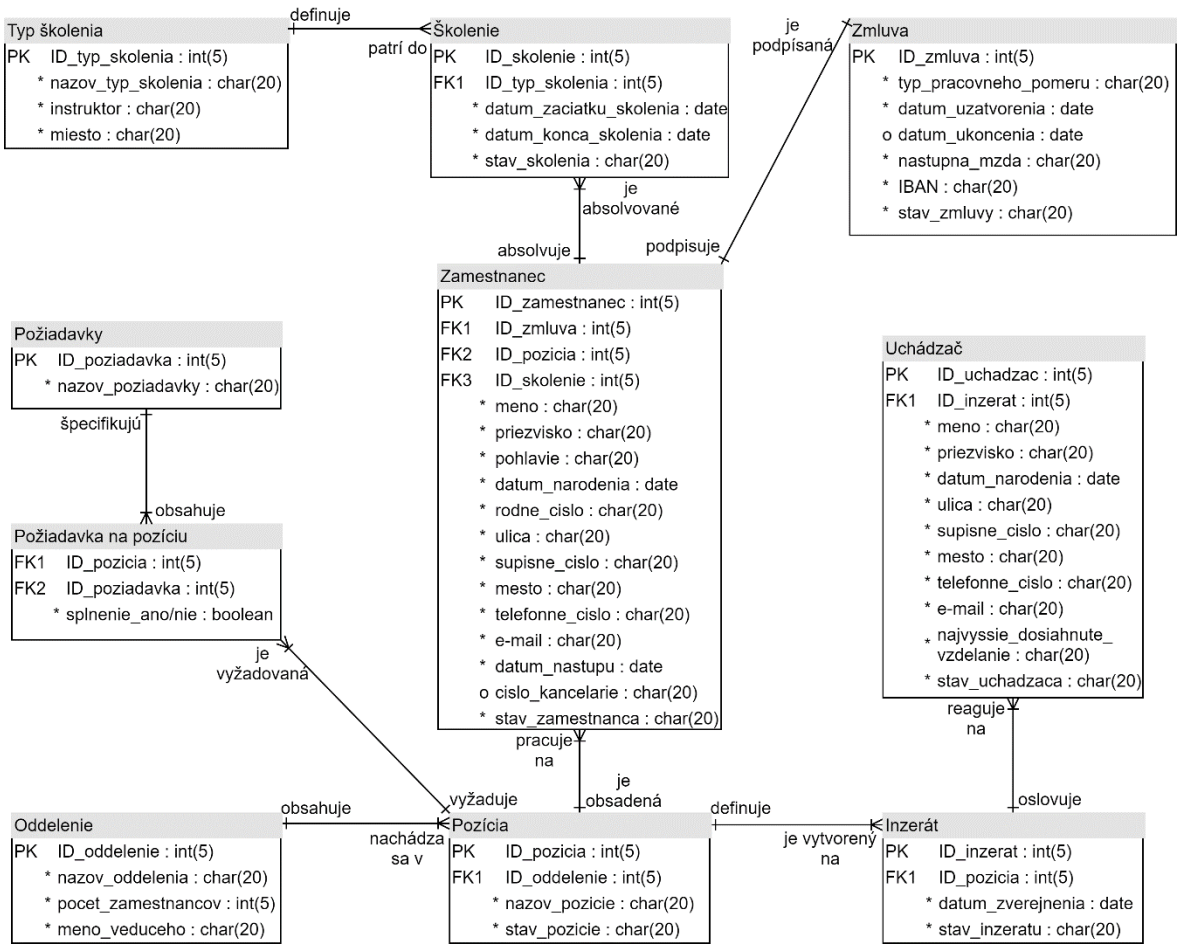
Logický model je o niečo bližšie k implementácii databázy, no definuje štruktúru dát v systéme stále nezávisle od konkrétnej implementácie. Zahrňuje detailnejšie popisy entít, ich atribúty, vzťahy medzi nimi a obsahuje už aj primárne a cudzie kľúče. Logický model zvykne nadväzovať na konceptuálny model a slúži ako základ pre fyzickú implementáciu

databázy. V logickom a fyzickom modeli sa nesmú nachádzať vzťahy typu M:N, pretože údaje musia byť v tretej normálnej forme. Logický model, uvedený na obrázku 22, vyzerá podobne ako predchádzajúci konceptuálny model, pribudli do neho však primárne a cudzie kľúče a obsahuje aj informáciu o nutnosti (*) respektíve voliteľnosti (o) údajov. Atribúty označené hviezdičkou musia byť vždy vyplnené, zatiaľ čo atribúty so symbolom o môžu zostať aj prázdne.



***Obrázok 22: Logický model pre personálne oddelenie Miba Steeltec, s.r.o.** (vlastné spracovanie)*

Posledný z uvedených modelov, fyzický model, sa zameriava práve na implementáciu, a teda opisuje, ako budú dáta uložené a organizované v konkrétnom databázovom systéme. Zahŕňa všetko, čo obsahoval logický model a navyše sa v ňom zvyknú nachádzať aj dátové typy, indexy, štruktúra tabuliek a optimalizácia výkonu. Tento model je najkonkrétnejší a poskytuje technické podrobnosti o tom, ako budú dáta uložené v databáze, keďže je priamočiaro spojený s konkrétnym hardvérom a so softvérovými platformami. Fyzický model je znázornený na obrázku 23.



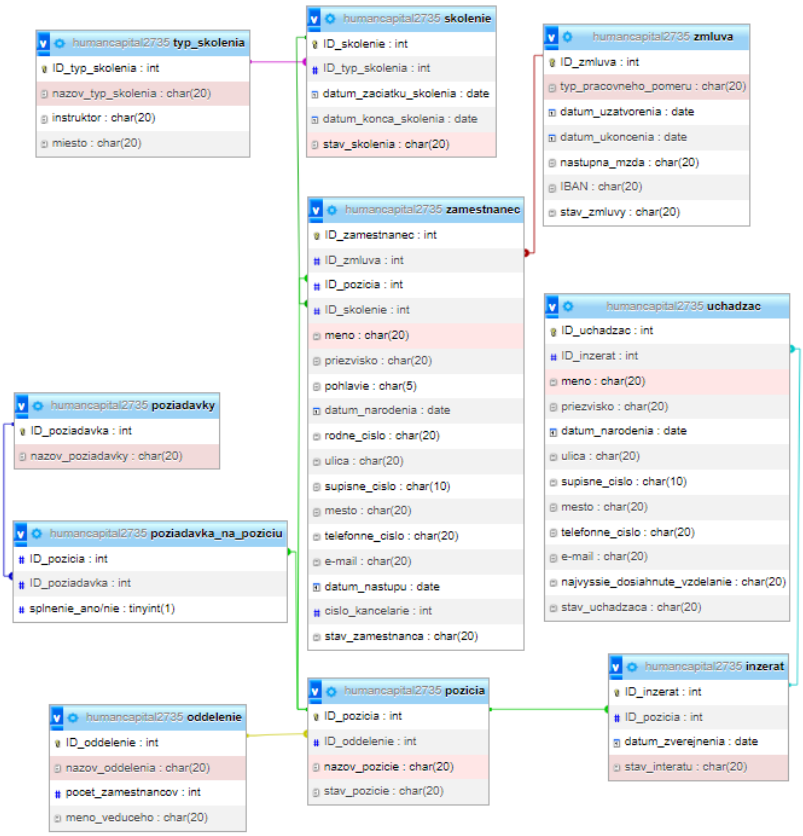
Obrázok 23: Fyzický model pre personálne oddelenie Miba Steeltec, s.r.o. (vlastné spracovanie)

Následne sme sa rozhodli vytvoriť relačnú databázu prostredníctvom www.webzdarma.cz, konkrétne cez phpMyAdmin, kam sme po samotnom vytvorení databázy zadali príkazy v štruktúrovanom vyhľadávacom jazyku SQL, pomocou ktorých sme vytvorili všetky potrebné entity aj s príslušnými atribútmi, dátovými typmi a primárnymi a cudzími kľúčmi. V tejto databáze budú uchovávané všetky dáta nevyhnutné pre potreby personálneho oddelenia. SQL kód vyzeral nasledovne:

```
CREATE TABLE IF NOT EXISTS
`zamestnanec` (
 `ID_zamestnanec` int(5) NOT NULL,
 `ID_zmluva` int(5) NOT NULL,
 `ID_pozicia` int(5) NOT NULL,
 `ID_skolenie` int(5) NOT NULL,
 `meno` char(20) NOT NULL,
 `priezvisko` char(20) NOT NULL,
 `pohlavie` char(5) NOT NULL,
 `datum_narodenia` date NOT NULL,
 `rodne_cislo` char(20) NOT NULL,
 `ulica` char(20) NOT NULL,
 `supisne_cislo` char(10) NOT NULL,
 `mesto` char(20) NOT NULL,
 `telefonne_cislo` char(20) NOT NULL,
 `e-mail` char(20) NOT NULL,
```

```
`datum_nastupu` date NOT NULL,
`cislo_kancelarie` int(5) NOT NULL,
`stav_zamestnanca` char(20) NOT
NULL,
PRIMARY KEY (`ID_zamestnanec`)
) ENGINE=InnoDB DEFAULT
CHARSET=latin1;

CREATE TABLE IF NOT EXISTS
`zmluva` (
`ID_zmluva` int(5) NOT NULL,
`typ_pracovneho_pomeru` char(20) NOT
NULL,
`datum_uzatvorenia` date NOT NULL,
`datum_ukoncenia` date NOT NULL,
`nastupna_mzda` char(20)  NOT NULL,
`IBAN` char(20)  NOT NULL,
`stav_zmluvy ` char(20) NOT NULL,

PRIMARY KEY (`ID_zmluva`)
) ENGINE=InnoDB DEFAULT
CHARSET=latin1;

ALTER TABLE zamestnanec
ADD FOREIGN KEY (ID_zmluva)
REFERENCES zmluva(ID_zmluva);
```

Pre prehľadnosť sme v tejto časti práce uviedli SQL kód len pre prvé dve entity a celý kód uvádzame v prílohe č.3. Pomocou uvedených SQL príkazov sme teda vytvorili databázu a cez záložku dizajnér sme zobrazili fyzický model systému pre personálne oddelenie podniku Miba Steeltec, s.r.o. (Obrázok 24 – zväčšený uvádzame v prílohe č.4.)



***Obrázok 24: Fyzický model pre personálne oddelenie Miba Steeltec, s.r.o. - phpMyAdmin** (vlastné spracovanie)*

3.6 UML diagramy

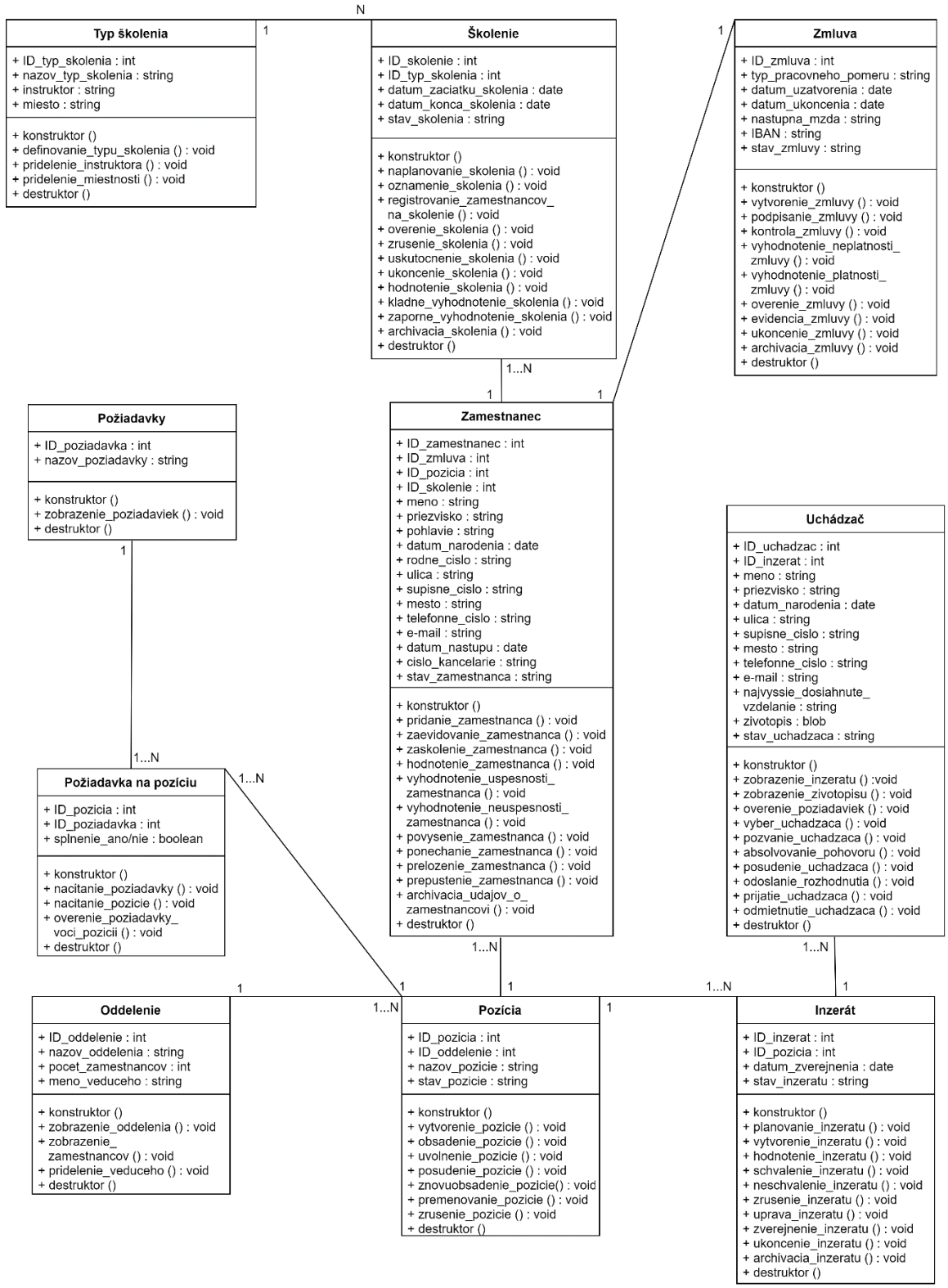
Ďalším a zároveň posledným krokom pri tvorbe praktickej časti tejto diplomovej práce bolo vytvorenie vybraných UML diagramov, ktoré nám pomohli lepšie pochopiť a vizualizovať štruktúru a fungovanie personálneho oddelenia. Spomedzi všetkých diagramov štandardu UML sme si zvolili 4 typy diagramov, a to diagram tried, vďaka ktorému sme popísali všetky triedy, stavové diagramy, ktoré nám pomohli priblížiť všetky stavy, ktorými jednotlivé triedy môžu prechádzať, ďalej sme zvolili use case, ktorý je považovaný za jeden z najdôležitejších a najpoužívanejších UML diagramov, pretože zobrazuje všetky funkcie navrhovaného systému, ako aj jeho vykonávateľov – aktérov, a na následné priblíženie jednotlivých prípadov použitia sme využili sekvenčné diagramy.

3.6.1 Diagram tried

Ako prvý spomedzi UML diagramov sme sa rozhodli vytvoriť diagram tried z dôvodu, že poskytuje štruktúrovaný pohľad na triedy, vzťahy medzi nimi a zaznamenáva aj ich atribúty a metódy.

Na začiatku sme identifikovali všetky triedy, o ktorých je v rámci navrhovaného softvéru na podporu personálneho oddelenia potrebné uchovávať informácie. Keďže softvér sa má týkať „personalistických“ činností a procesov spojených s tvorbou inzerátov, výberom uchádzačov, prijímaním zamestnancov, on-boarding procesom a zaškolením zamestnancov, ostatné kompetencie tohto konkrétneho personálneho oddelenia ako mzdové účtovníctvo, jazykové kurzy, duálne vzdelávanie a recepcia nie sú súčasťou týchto modelov. V tomto prípade sme teda vytvorili diagram s desiatimi triedami, medzi ktoré patrí: inzerát, uchádzač, zamestnanec, oddelenie, pozícia, požiadavka na pozíciu, požiadavky, zmluva, školenie a typ školenia. Ku každej triede sme priradili jej atribúty aj s príslušnými dátovými typmi. Slovným atribútom ako napríklad meno, priezvisko, instruktor, miesto, nazov_pozicie, najvyssie_dosiahnute_vzdelanie, nazov_poziadavky, e-mail, ako aj všetkým atribútom popisujúcim stavy tried sme priradili dátový typ *string*, ktorý sa na uchovávanie reťazcov používa asi najčastejšie. Použili sme ho aj pri atribútoch, ktoré sa na prvý pohľad môžu javiť ako číselné, ako napríklad telefonne_cislo, supisne_cislo, rodne_cislo alebo IBAN, pretože tieto atribúty zvyknú obsahovať písmená (SK, 103b) alebo špeciálne znaky (+, /). Pre číselné atribúty ako pocet_zamestnancov a všetky identifikátory sme použili dátový typ *int* a pre atribúty uchovávajúce dátumy sme použili *date*. V jednom prípade, pri atribúte splnenie_poziadavky_ano/nie sme použili *boolean*, ktorý nadobúda hodnoty true a false a pri atribúte zivotopis sme použili *blob*, pomocou ktorého sú zaznamenávané súbory.

V ďalšom kroku sme ku každej triede doplnili jej metódy (zverejnenie_inzeratu, vytvorenie_pozicie, pridanie_zamestnanca, evidencia_zmluvy, pridelenie_instruktora) a ku každej z tried sme pridali aj konštruktor a deštruktor triedy. Posledným krokom tvorby diagramu tried bolo ich vzájomné prepojenie a určenie vzťahov a kardinality. (Obrázok 25)

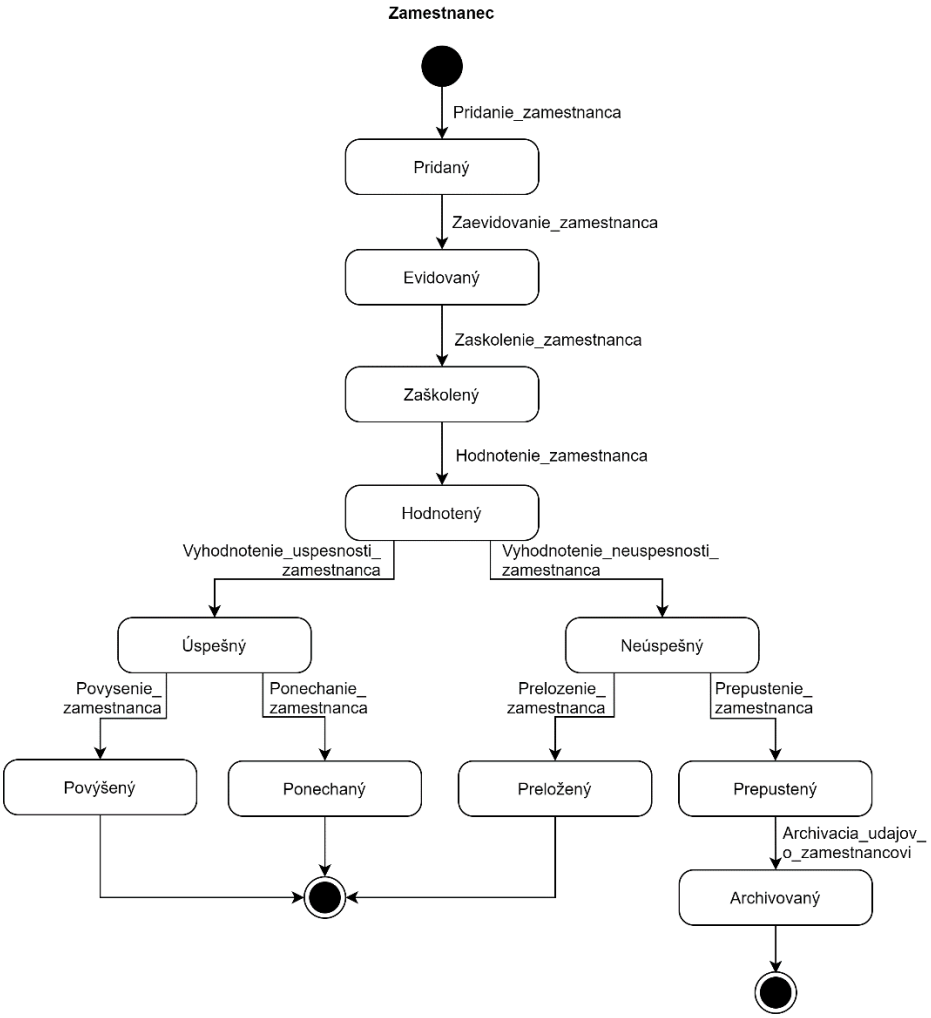


Obrázok 25: Diagram tried pre personálne oddelenie Miba Steeltec, s.r.o. *(vlastné spracovanie)*

3.6.2 Stavové diagramy

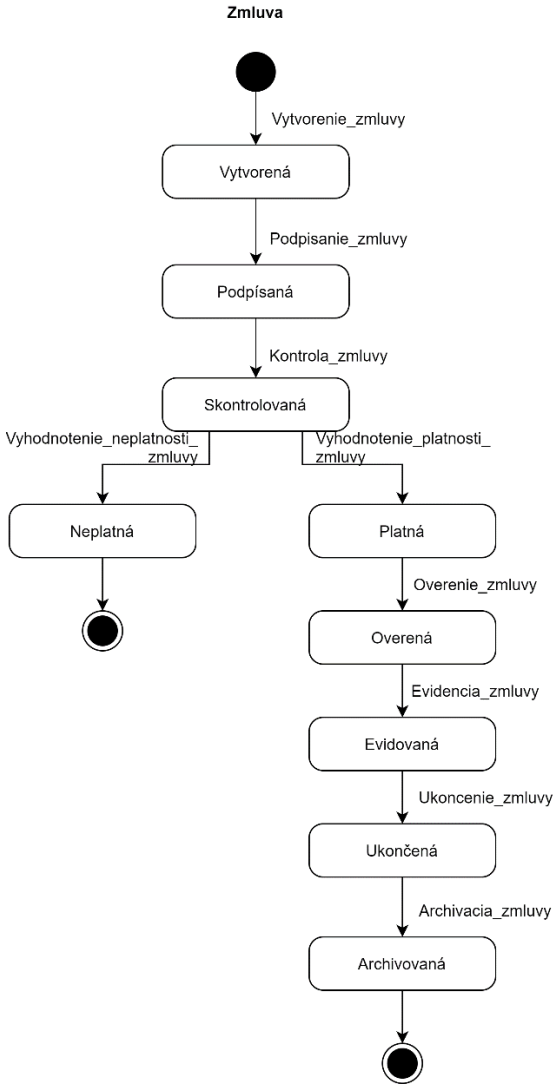
Na diagram tried sme nadviazali prostredníctvom stavových diagramov, ktoré sa používajú na reprezentáciu správania systému v závislosti od udalostí, ktoré naň pôsobia a ich úlohou je identifikovať všetky stavy a prechody medzi stavmi, ktoré môžu pri danej triede nastať. Diagram začína začiatočným a končí koncovým stavom. Stavové diagramy sa zväčša vytvárajú len pre niektoré triedy, v závislosti od ich zložitosti správania, komplexnosti a začlenenia do architektúry systému. V tejto práci sme vytvárali šesť stavových diagramov pre triedy zamestnanec, zmluva, inzerát, pozícia, školenie a uchádzač.

Pre triedu zamestnanec sme na začiatku identifikovali stavy ako pridaný, evidovaný, zaškolený a hodnotený, spolu s príslušnými prechodmi medzi týmito stavmi. Zamestnanec môže byť vzhľadom na hodnotenie úspešnosti buď úspešný, kedy je buď ponechaný na tej pracovnej pozícii, kde bol aj predtým, alebo môže byť povýšený. V prípade neúspešnosti sa však môže stať, že bude preložený na inú pozíciu alebo bude prepustený. Model popisujúci prechod jednotlivými stavmi je uvedený na obrázku 26.



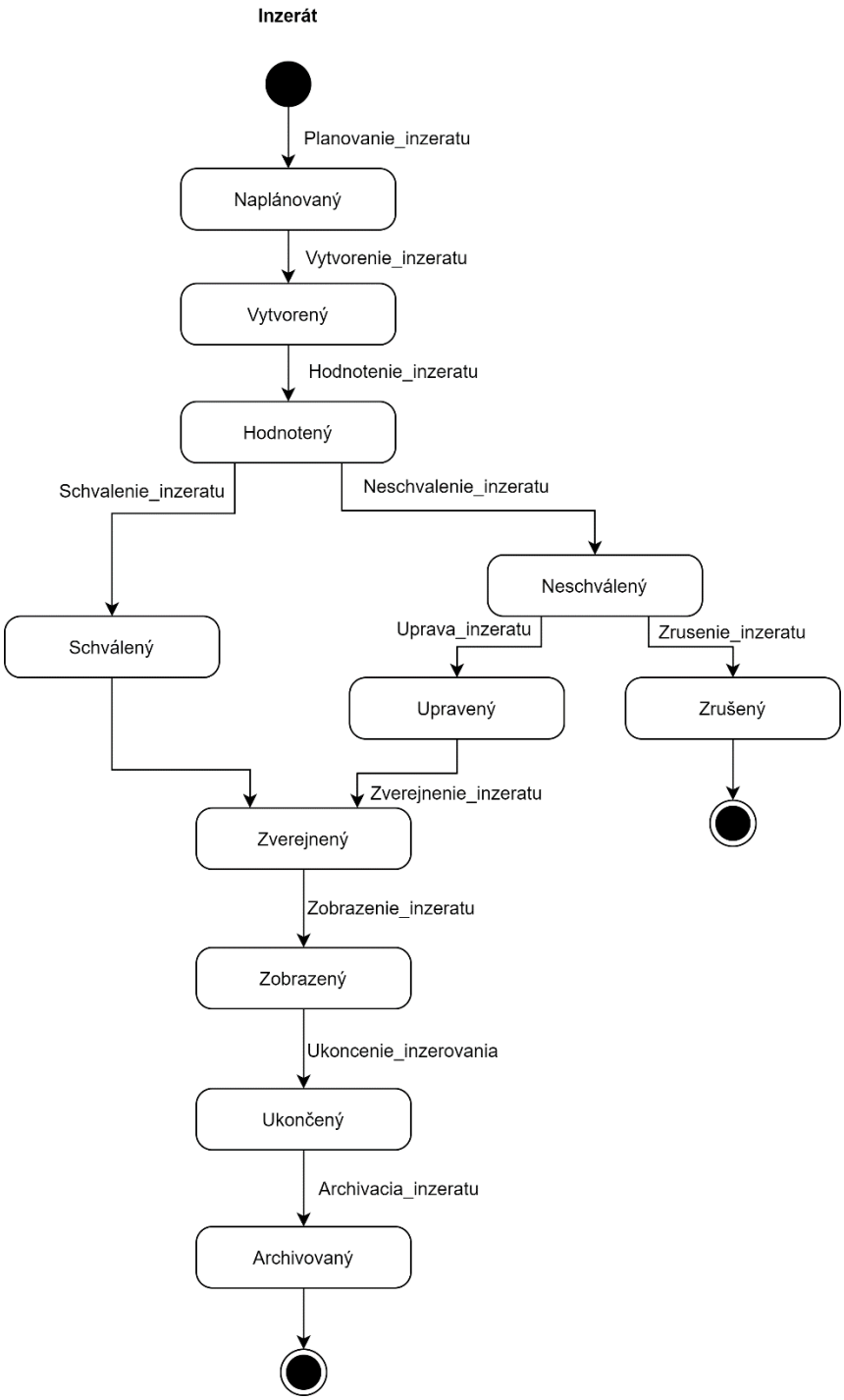
Obrázok 26: Stavový diagram - zamestnanec (vlastné spracovanie)

Stavový diagram pre zmluvu obsahuje stavy ako vytvorená, podpísaná a následne nasleduje kontrola, pričom sa môže zistiť, že zmluva je neplatná alebo ak je všetko v poriadku, pokračuje sa ďalej k overeniu, evidencii, ukončeniu a archivácii. Všetky stavy spolu s prechodmi medzi nimi sú zobrazené na obrázku 27.



Obrázok 27: Stavový diagram - zmluva (vlastné spracovanie)

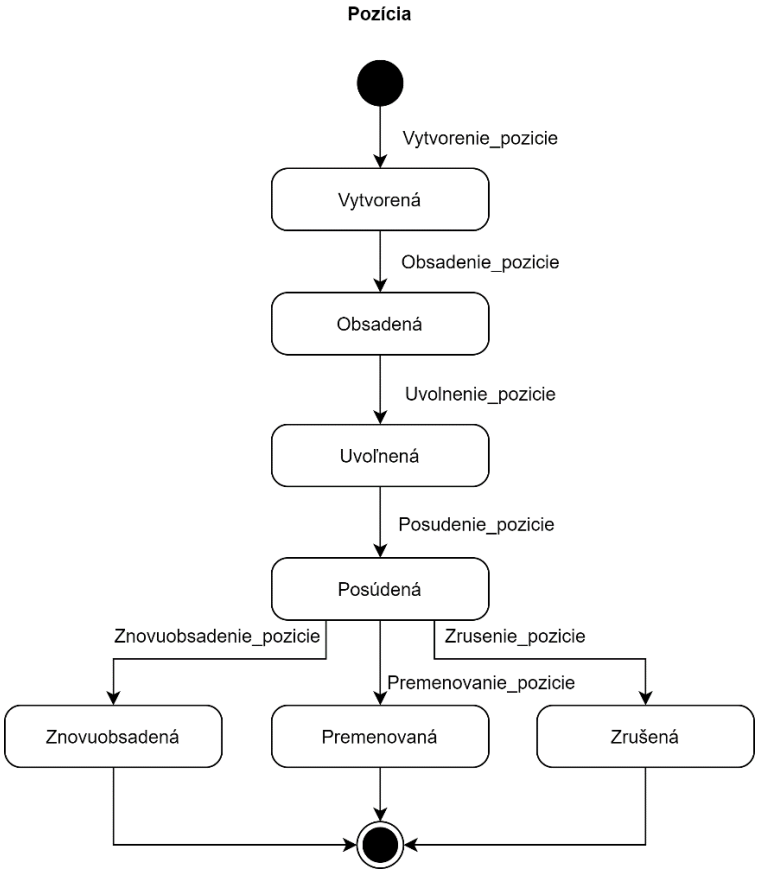
Pri triede inzerát sme postupovali tak, že sme identifikovali stavy naplánovaný, vytvorený a hodnotený a rovnako ako pri stavovom diagrame pre triedu zamestnanec môžu po hodnotení inzerátu nastať 2 možnosti. Buď je inzerát hneď schválený a pokračuje ďalej k zverejneniu, zobrazeniu, ukončeniu a archivácii, alebo nastane druhá možnosť, kedy je neschválený a je buď úplne zrušený alebo upravený, pričom po úprave môže ďalej pokračovať k zverejneniu. Stavový diagram obsahuje samozrejme aj prechody medzi stavmi, ktoré predstavujú správanie sa systému v reakcii na rôzne udalosti alebo podmienky. (Obrázok 28)



Obrázok 28: Stavový diagram - inzerát (vlastné spracovanie)

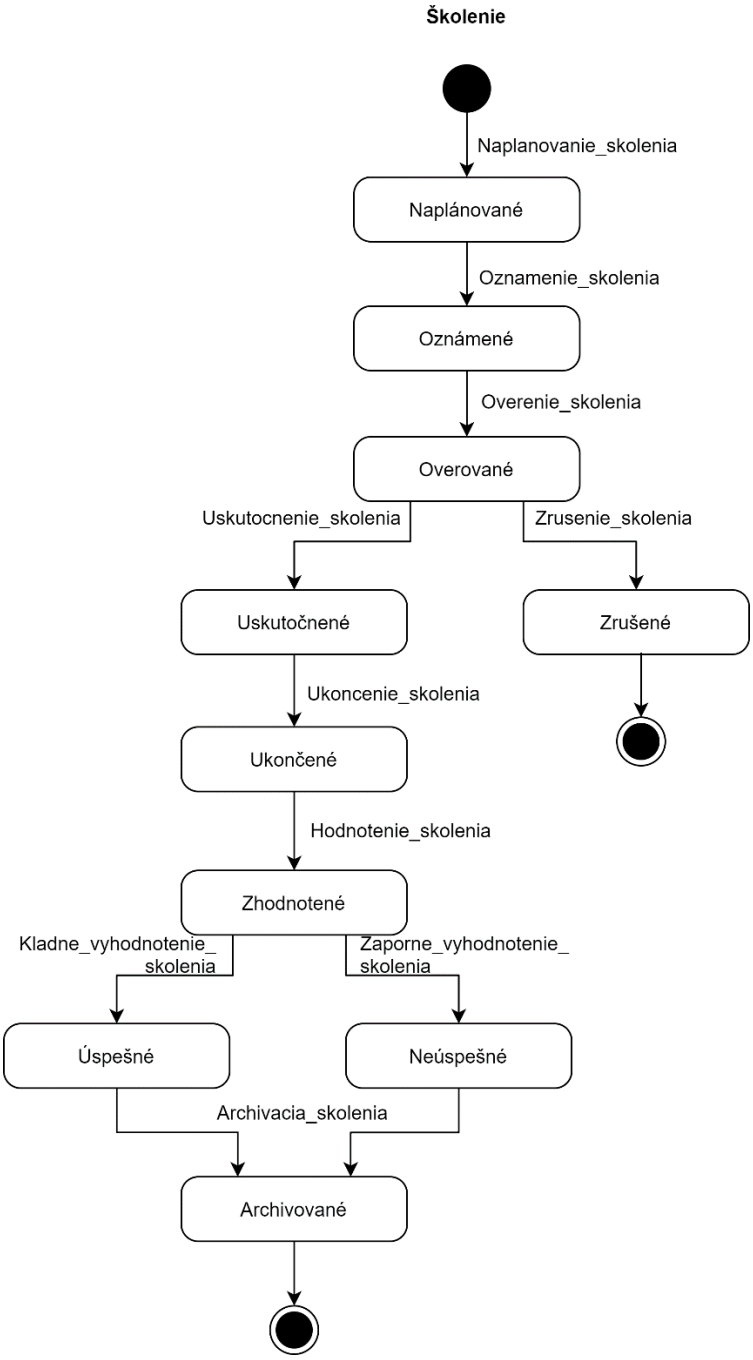
Trieda pozícia môže mať stavy vytvorená, obsadená, uvoľnená a následne je pozícia vždy posúdená, pričom po posúdení môže nastať jeden z troch variantov. Pozícia môže byť znovuobsadená, kedy sa na danú pozíciu vytvorí výberové konanie a príjme sa nový zamestnanec alebo sa na ňu preradí zamestnanec z inej pozície, alebo premenovaná, či úplne zrušená. Pozícia sa zvyčajne ruší z dôvodu reštrukturalizácie organizácie, automatizácie procesov, ktoré predtým vykonával pracovník na danej pozícii, zníženia nákladov alebo pri

zmene podnikovej stratégie. Stavový diagram pre triedu pozícia so všetkými stavmi a prechodmi medzi nimi uvádzame na obrázku 29.



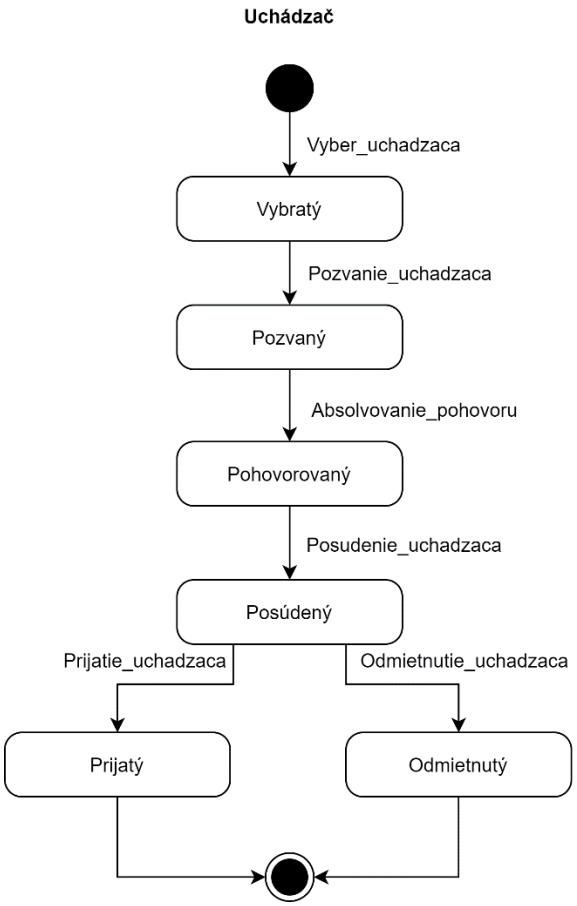
Obrázok 29: Stavový diagram - pozícia (vlastné spracovanie)

Stavový diagram pre triedu školenie začína naplánovaním, oznámením a overením školenia, pričom v tomto kroku môže byť školenie zrušené technikom bezpečnosti (detailnejší postup činností je uvedený neskôr v sekvenčnom diagrame evidencia zaškolených zamestnancov). Ak overenie prebehne bez komplikácií, školenie je uskutočnené, ukončené a prechádza sa k hodnoteniu školenia, pričom vyhodnotenie školenia môže byť buď kladné alebo záporné a v závislosti od toho sa určí, či bolo školenie úspešné alebo neúspešné. Týmto spôsobom je v stavových diagramoch zachytené vetvenie, ktoré zobrazuje situáciu, kedy môže trieda v určitej situácii zareagovať viacerými spôsobmi na základe určitých podmienok alebo udalostí. V každom prípade je posledným krokom stavového diagramu pre triedu školenie archivácia školenia, nakoľko archivácia a evidencia sú jedny z najdôležitejších činností pri návrhu ale aj samotnej prevádzke akéhokoľvek systému či databázy. Tento model, rovnako ako všetky stavové diagramy, obsahuje aj začiatočný a koncový stav a prechody medzi jednotlivými stavmi. Celý stavový diagram pre triedu školenie je zobrazený na obrázku 30.



***Obrázok 30: Stavový diagram - školenie** (vlastné spracovanie)*

Posledný stavový diagram, ktorý sme pri návrhu softvéru na podporu personálneho oddelenia spoločnosti Miba Steeltec, s.r.o. vytvárali, sa týkal triedy uchádzač. Pri tejto triede sme identifikovali možné stavy ako napríklad vybratý, pozvaný, „pohovorovaný“ (keď uchádzač absolvuje pohovor) a posúdený, pričom na základe rôznych faktorov ako sú pohovor, životopis, skúsenosti, najvyššie dosiahnuté vzdelanie, splnenie požiadaviek na pozíciu a v niektorých prípadoch aj zvládnutá praktická skúška, bude uchádzač buď prijatý alebo odmietnutý. V diagrame sú graficky znázornené samotné stavy systému spolu s prechodmi medzi nimi. Stavový diagram pre triedu uchádzač uvádzame na obrázku 31.



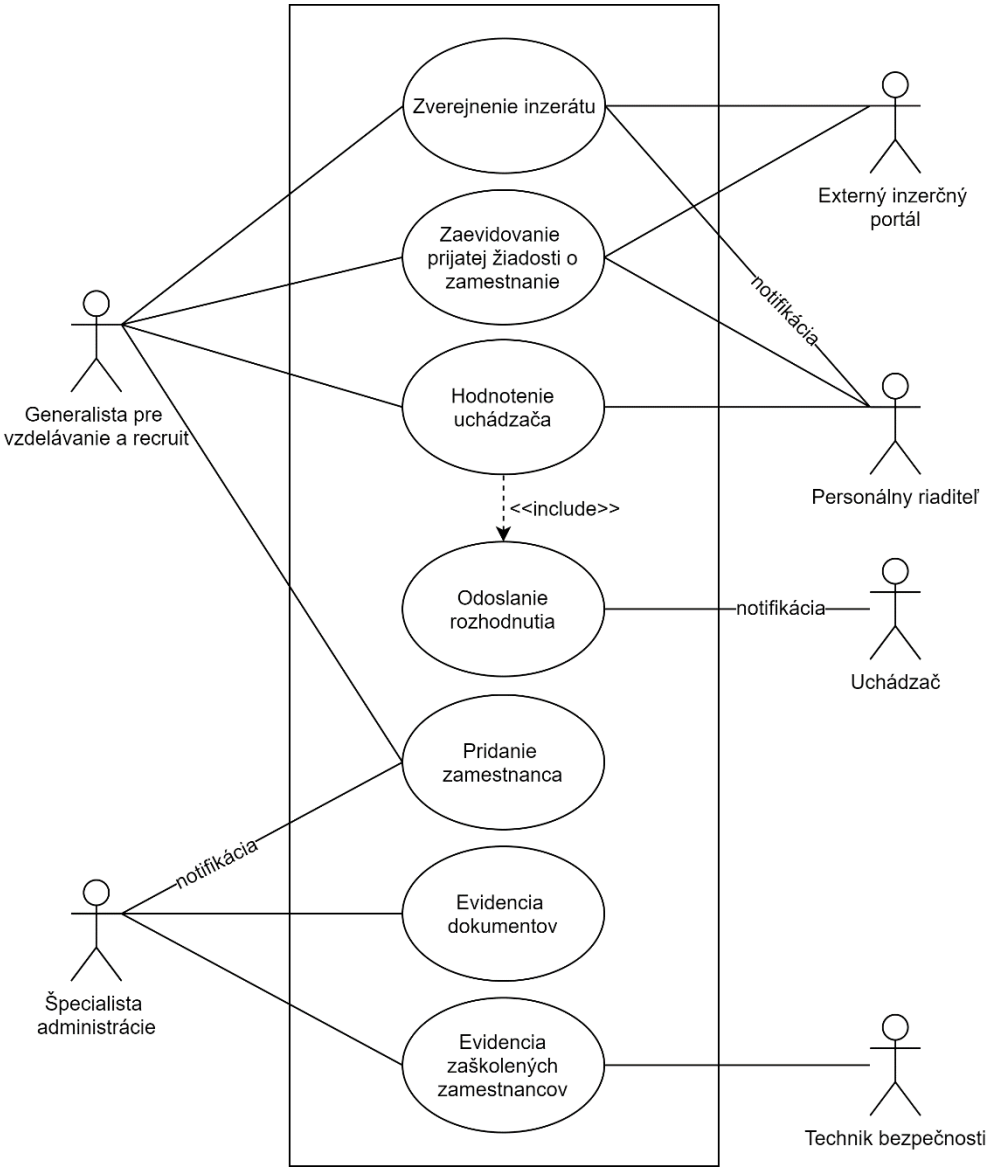
Obrázok 31: Stavový diagram - uchádzač (vlastné spracovanie)

3.6.3 Diagram prípadov použitia

Diagram prípadov použitia (use case diagram) patrí medzi UML diagramy správania a zobrazuje interakcie medzi aktérmi a navrhovaným systémom na dosiahnutie určitého cieľa. Každý use case reprezentuje konkrétnu funkcionalitu, ktorú vykonáva systém prostredníctvom aktérov. Aktéri sú vonkajšie entity ako užívatelia alebo iné systémy, ktoré interagujú s daným systémom a vykonávajú jednotlivé scenáre. Dôležitou súčasťou správneho diagramu prípadov použitia je zachytenie vzťahov medzi aktérmi a scenármi. Use case zvyčajne obsahuje hranicu systému, ktorá graficky znázorňuje oblasť, v ktorej sa nachádza systém a jeho funkcie a interakcie medzi aktérmi.

Ďalším krokom pri návrhu softvéru bolo teda vytvorenie use case modelu pre personálne oddelenie podniku Miba Steeltec, s.r.o. Na začiatku sme identifikovali všetkých aktérov, ktorí budú mať prístup do systému alebo s ním budú interagovať. V našom prípade to boli generalista pre vzdelávanie a recruit, špecialista administrácie, technik bezpečnosti a personálny riaditeľ. Títo aktéri priamo zasahujú do systému a sú zodpovední za vykonanie prípadov použitia a ich jednotlivých činností. Ďalšími aktérmi v našom systéme sú externý

inzerčný portál (Profesia), prostredníctvom ktorého sa zverejňujú inzeráty a ktorý do systému odosiela údaje o uchádzačoch, ktorí na inzerát zareagovali a majú záujem o pracovnú pozíciu a v neposlednom rade je aktér uchádzač, ktorý síce do systému prístup nemá a nevykonáva v ňom žiadne operácie, len so systémom interaguje a sú mu posielané notifikácie zo systému. Následne sme identifikovali ciele aktérov a funkcie, ktoré by mal systém vykonávať. Vytvorili sme tak sedem prípadov použitia, ktoré zachytávajú fungovanie systému, a to zverejnenie inzerátu, zaevidovanie prijatej žiadosti o zamestnanie, hodnotenie uchádzača spolu s prípadom zahrnutia odoslanie rozhodnutia, pridanie zamestnanca, evidencia dokumentov a evidencia zaškolených zamestnancov. Diagram prípadov použitia uvádzame na obrázku 32. Po samotnom vytvorení use case diagramu je nutné každý scenár detailne popísať, čo sme podporili tvorbou sekvenčných diagramov.

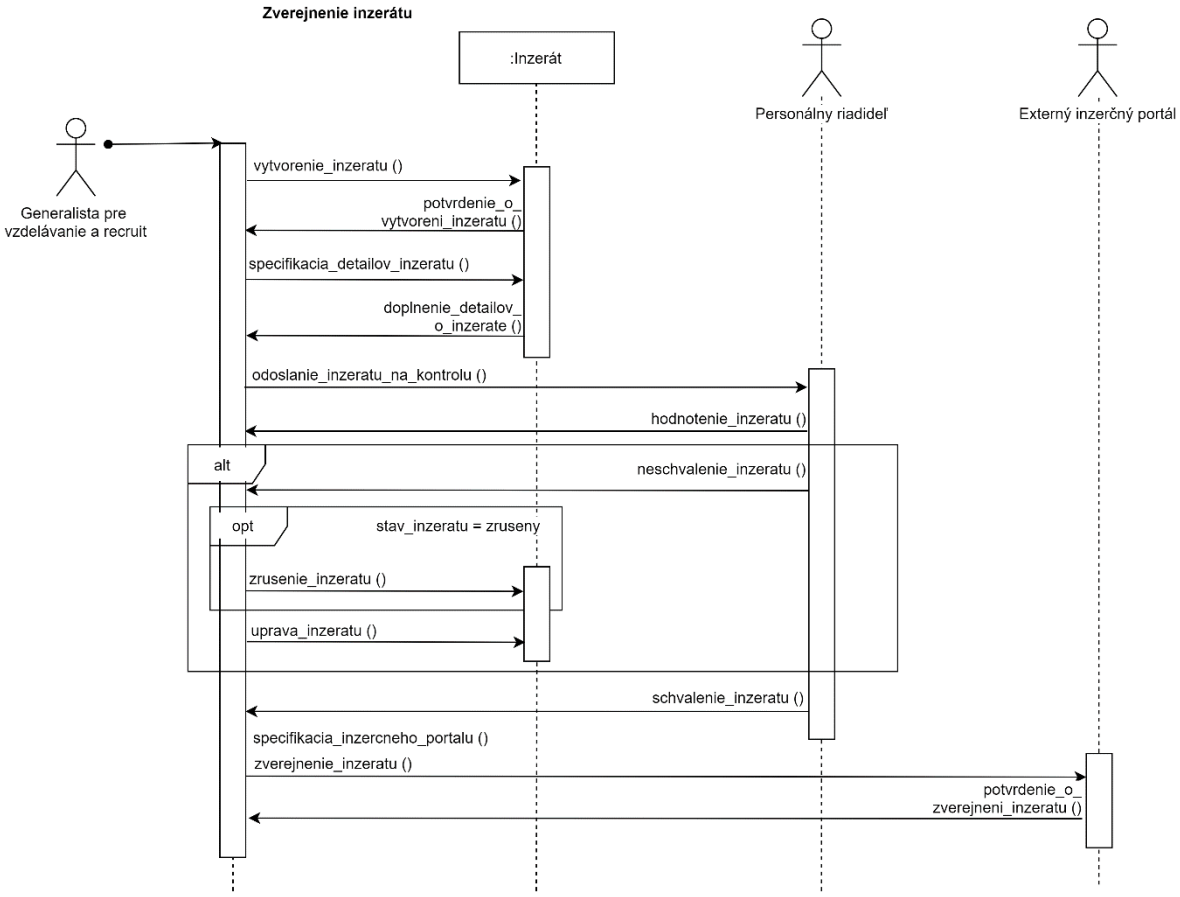


Obrázok 32: Diagram prípadov použitia pre personálne oddelenie Miba Steeltec, s.r.o. *(vlastné spracovanie)*

3.6.4 Sekvenčné diagramy

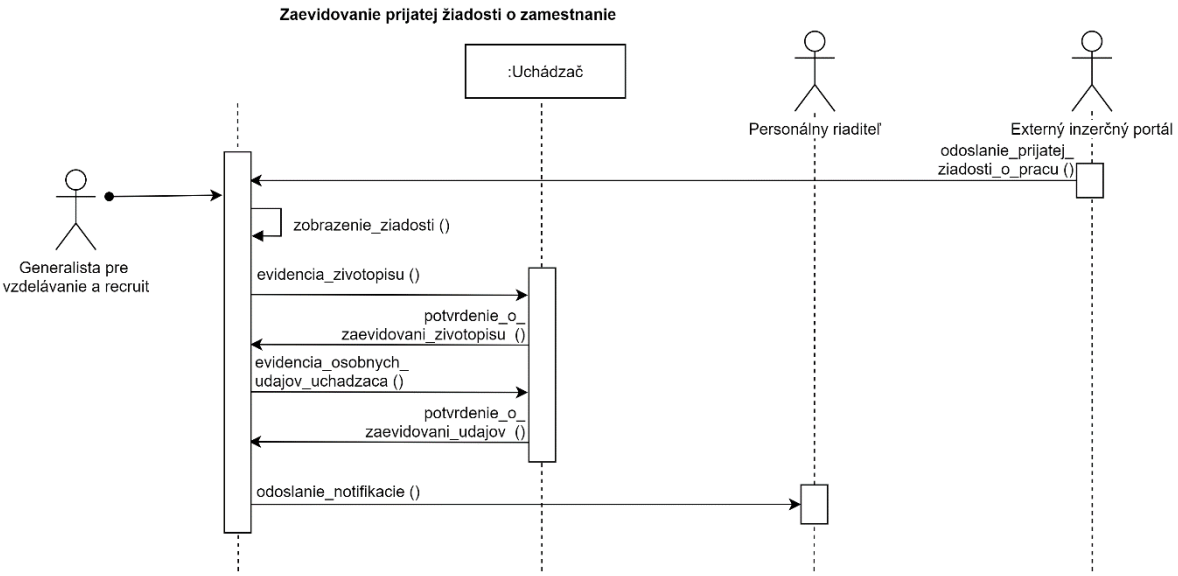
Ako sme spomenuli v predchádzajúcej časti, sekvenčné diagramy sa používajú na vizuálnu interpretáciu interakcií medzi objektmi v systéme. Z toho dôvodu sme sa rozhodli pomocou nich vysvetliť a opísať jednotlivé prípady použitia. Každý sekvenčný diagram popisuje jeden zo siedmich prípadov použitia.

Prvým prípadom použitia, ktorý sme vysvetľovali prostredníctvom sekvenčných diagramov, bolo **zverejnenie inzerátu**. V tomto sekvenčnom diagrame je aktérom generalista pre vzdelávanie a recruit a vystupujú tu aj ďalšie objekty, a to inzerát, personálny riaditeľ a externý inzerčný portál. Na začiatku sekvenčného diagramu použije aktér metódu na vytvorenie inzerátu, špecifikuje detaily o inzeráte a odošle inzerát na kontrolu personálnemu riaditeľovi, ktorý inzerát hodnotí. V závislosti od jeho rozhodnutia môže byť inzerát neschválený, čo znamená, že sa musí buď upraviť alebo je rovno zrušený. V opačnom prípade, ak je inzerát schválený, generalista ďalej špecifikuje zvolený inzerčný portál a inzerát naň odošle. Systém po každej správe alebo činnosti vždy posiela spätné potvrdenia, ktoré signalizujú, že akcia prebehla úspešne. Sekvenčný diagram pre zverejnenie inzerátu uvádzame na obrázku 33.



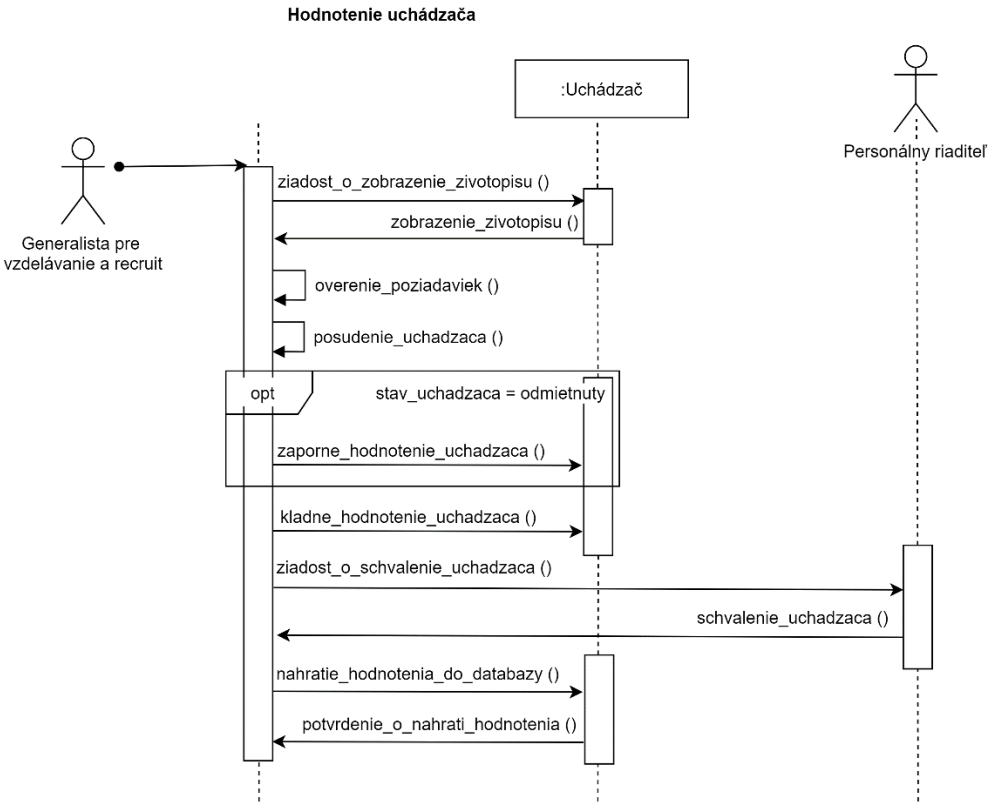
***Obrázok 33: Sekvenčný diagram – zverejnenie inzerátu** (vlastné spracovanie)*

Po zverejnení inzerátu nasleduje **zaevidovanie prijatej žiadosti o zamestnanie**. V tomto sekvenčnom diagrame je aktérom opäť generalista pre vzdelávanie a recruit a figurujú tu objekty uchádzač spolu s personálnym riaditeľom, ktorý však v tomto procese priamo nevystupuje, je mu len odoslaná notifikácia, a externý inzerčný portál, ktorý do systému odošle prijatú žiadosť o zamestnanie na pozíciu podniku Miba Steeltec, s.r.o. Po prijatí údajov o uchádzačovi od externého inzerčného portálu generalista žiadosť zobrazí, zaeviduje do systému životopis uchádzača, zaeviduje aj uchádzačove osobné údaje a odošle notifikáciu o novom uchádzačovi personálnemu riaditeľovi. (Obrázok 34)



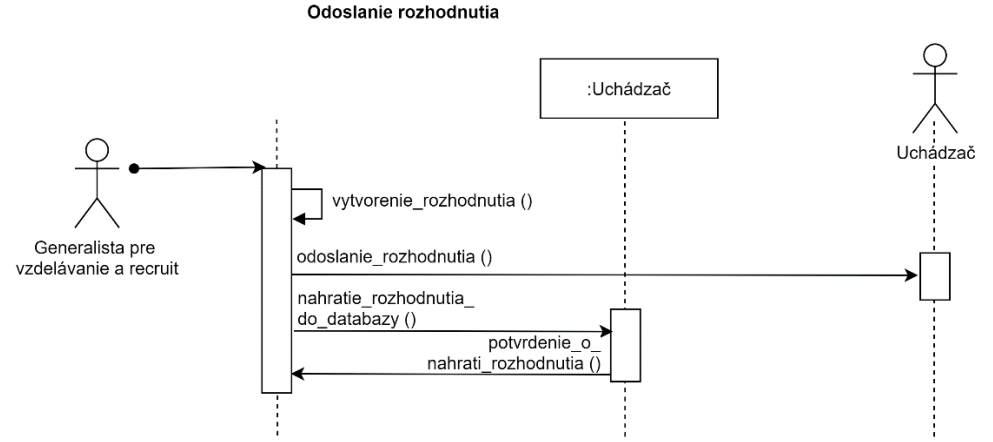
***Obrázok 34: Sekvenčný diagram – zaevidovanie prijatej žiadosti o zamestnanie** (vlastné spracovanie)*

Po tom, čo bola do systému zaevidovaná žiadosť od uchádzača, bol uchádzač kontaktovaný, dohodol sa s ním termín pohovoru a bol realizovaný pohovor, no tieto činnosti sa nevykonávajú prostredníctvom softvéru ale realizuje ich priamo zamestnanec personálneho oddelenia. Ďalším sekvenčným diagramom je teda **hodnotenie uchádzača**, ktoré vykonáva tiež generalista pre vzdelávanie a recruit a v tomto diagrame vystupuje objekt uchádzač a personálny riaditeľ, ktorý uchádzača musí schváliť. Prvým krokom je zobrazenie životopisu uchádzača z databázy, overenie, či daný uchádzač spĺňa požiadavky a posúdenie uchádzača generalistom. Ak dá generalista pre vzdelávanie a recruit uchádzačovi záporné hodnotenie (nespĺňa požiadavky, nemá dostatočné vzdelanie, skúsenosti...), uchádzač je zamietnutý. Ak však dostane kladné hodnotenie, generalista pošle personálnemu riaditeľovi žiadosť o schválenie uchádzača, čo je síce formalita, ale musí byť vykonaná, a následne môže generalista nahrať hodnotenie uchádzača do databázy. Proces hodnotenia uchádzača je znázornený na obrázku 35.



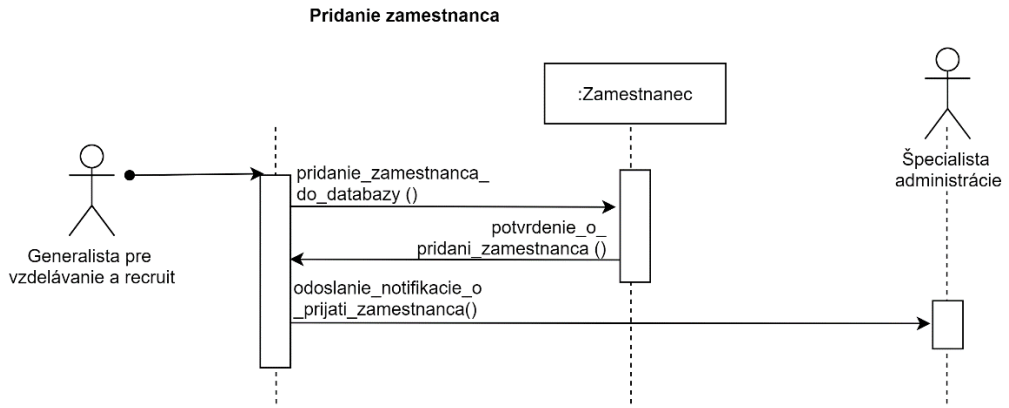
***Obrázok 35: Sekvenčný diagram – hodnotenie uchádzača** (vlastné spracovanie)*

Na prípad použitia hodnotenie uchádzača nadväzuje jeho prípad zahrnutia **odoslanie rozhodnutia**, ktorý je s ním prepojený pomocou vzťahu include, čo znamená, že sa musí vykonať vždy, keď je vykonaný základný prípad použitia, pretože po hodnotení uchádzača mu personálne oddelenie musí oznámiť, či bol prijatý alebo odmietnutý. Tento prípad použitia je pomerne jednoduchý a zahŕňa vytvorenie rozhodnutia generalistom pre vzdelávanie a recruit, následné odoslanie rozhodnutia uchádzačovi prostredníctvom notifikácie a nahratie rozhodnutia do databázy spolu s potvrdením systému. Tento sekvenčný diagram je uvedený na obrázku 36.



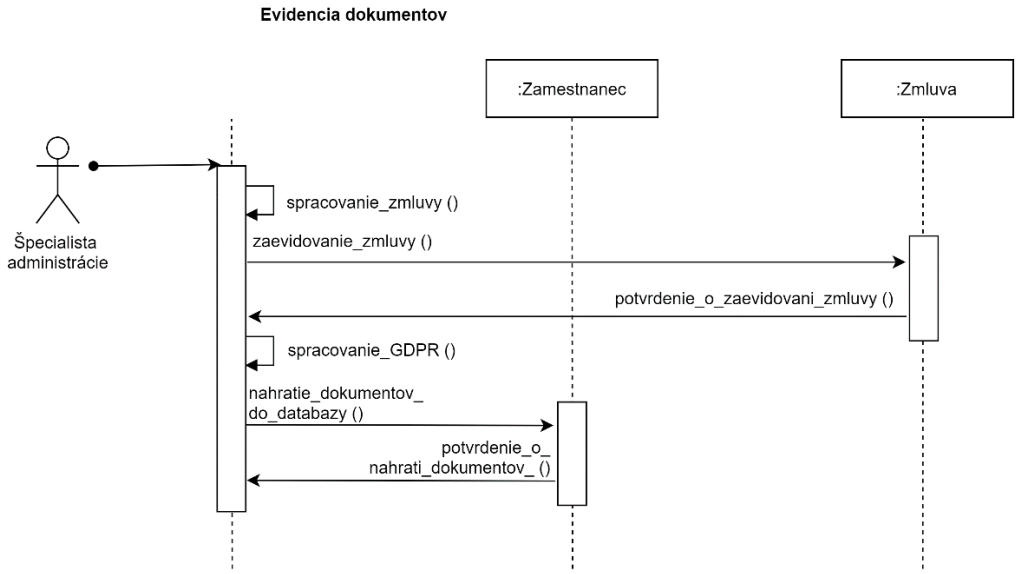
***Obrázok 36: Sekvenčný diagram – odoslanie rozhodnutia** (vlastné spracovanie)*

Ak dostal uchádzač kladné rozhodnutie a bol prijatý do zamestnania, je potrebné **pridanie zamestnanca** do databázy. V tomto sekvenčnom diagrame generalista pre vzdelávanie a recruit pridá nového zamestnanca do databázy zamestnancov, priradí mu jedinečný identifikátor, zaeviduje aj všetky jeho osobné údaje a odošle notifikáciu o prijatí zamestnanca špecialistovi administrácie, ktorého zodpovednosťou je evidencia dokumentov u novoprijatých zamestnancov. Sekvenčný diagram pre pridanie zamestnanca uvádzame na obrázku 37.



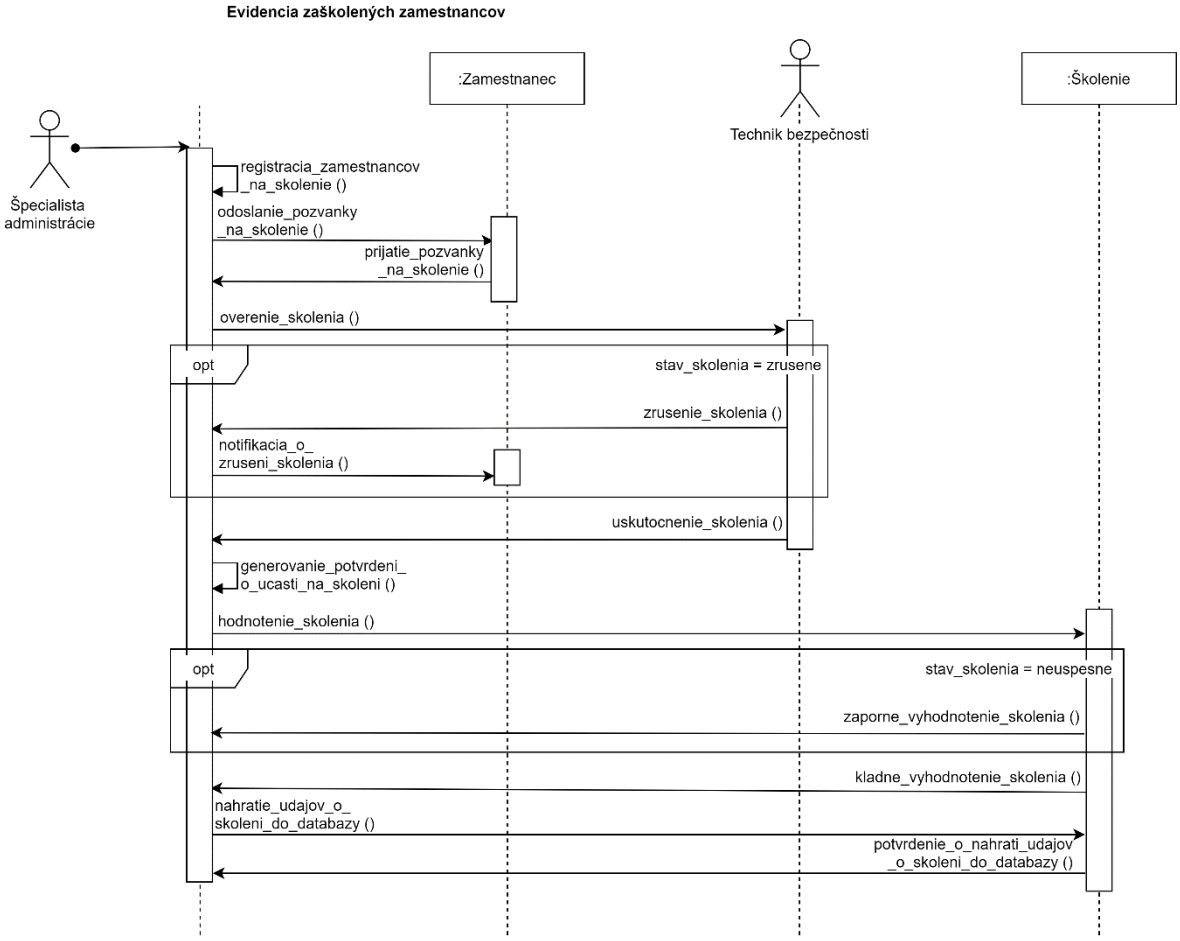
***Obrázok 37: Sekvenčný diagram – pridanie zamestnanca** (vlastné spracovanie)*

Ďalším sekvenčným diagramom je **evidencia dokumentov**, ktorého aktérom je špecialista administrácie a sú v ňom použité ďalšie dva objekty, a to zamestnanec a zmluva. Evidenciu dokumentov začína špecialista administrácie spracovaním už podpísanej a overenej zmluvy, ktorú zaeviduje do databázy k ostatným zmluvám. V ďalšom kroku spracuje formuláre k GDPR a ostatné dokumenty súvisiace s nástupom do zamestnania a všetky nahrá do databázy k príslušnému zamestnancovi. (Obrázok 38)



***Obrázok 38: Sekvenčný diagram – evidencia dokumentov** (vlastné spracovanie)*

Posledným prípadom použitia, ktorý sme znázorňovali pomocou sekvenčného diagramu je **evidencia zaškolených zamestnancov**. V tomto diagrame je aktérom špecialista administrácie a okrem neho v ňom vystupujú objekty zamestnanec a školenie, a technik bezpečnosti. Aktér najprv zaregistruje zamestnancov, ktorí sa majú zúčastniť školenia a následne im odošle pozvánky s údajmi o realizovanom školení. U technika bezpečnosti overí, či sa dané školenie uskutoční alebo bude musieť byť zrušené, o čom následne informuje zamestnancov. Ak školenie nebolo zrušené, po jeho uskutočnení špecialista administrácie vygeneruje potvrdenia o účasti zamestnancov, odošle údaje na hodnotenie školenia, pričom vyhodnotenie školenia môže byť kladné alebo záporné. Na konci špecialista nahrá údaje o školení do databázy a dostane naspäť potvrdenie o ich úspešnom nahratí. Sekvenčný diagram pre prípad použitia evidencia zaškolených zamestnancov uvádzame na obrázku 39.



Obrázok 39: Sekvenčný diagram – evidencia zaškolených zamestnancov (vlastné spracovanie)

3.7 Diskusia

Ako sme uviedli v prvej kapitole, v súčasnosti je softvér používaný takmer v každej oblasti života človeka a je to veľmi užitočný nástroj, ktorý pomáha podnikom pri automatizácii procesov, skrátení doby trvania procesov, šetrí náklady, zvyšuje produktivitu, spoľahlivosť a presnosť, poskytuje mnohé analytické nástroje a pomáha pri reportingu, komunikácii medzi zamestnancami, oddeleniami a aj viacerými organizáciami a zvyšuje konkurencieschopnosť daného podniku. Toto sú len niektoré z mnohých dôvodov, prečo podniky na svoju činnosť už niekoľko rokov nepoužívajú len ľudskú pracovnú silu, ale pri svojom fungovaní implementujú aj rôzne systémy. Niektoré organizácie sa vyberú cestou tvorby vlastného interného systému, ktorý bude navrhnutý presne podľa ich požiadaviek, môžu ho pravidelne aktualizovať, podľa potreby vylepšovať a upravovať. Takto vytvorený systém bude síce efektívnejší a presnejší, no jeho nevýhodou môže byť veľmi zložitý vývoj, vysoké počiatočné náklady, dlhý čas potrebný na jeho tvorbu a náročné udržiavanie. Kvôli tomu sa väčšina podnikov radšej rozhodne pre zakúpenie a implementáciu už vytvoreného softvérového riešenia od externého dodávateľa, ktorý má v danej oblasti skúsenosti a je overený mnohými používateľmi. Tento softvér si čo najlepšie prispôsobia na základe svojich potrieb, no v konečnom dôsledku nikdy nebude dosahovať také výsledky, ako softvér vytvorený na mieru.

V tejto diplomovej práci sme sa pokúsili o vytvorenie analýzy a návrhu softvéru pre personálne oddelenie podniku Miba Steeltec, s.r.o., ktorý by v budúcnosti mohol byť ďalej naprogramovaný a implementovaný do podniku. Počas fázy analýzy a oboznamovania sa s podnikom a personálnym oddelením sme zistili, že oddelenie v súčasnosti používa softvérový produkt od jednej z najpopulárnejších a najväčších organizácií v oblasti softvérového inžinierstva, a to od spoločnosti SAP. Systémy od SAP-u sú bezpochyby vybavené špičkovou funkcionalitou, sú rýchle, flexibilné, prispôsobujú sa trhu a požiadavkám používateľov. Využívajú sa najmä v oblasti financií, plánovania výroby, nákupu, riadenia skladových zásob, výroby a logistiky, CRM aj v riadení projektov, no po rozhovore s personálnym riaditeľom a zamestnancami personálneho oddelenia sme zistili, že pre potreby personálneho oddelenia nie sú ideálne. Medzi nedostatky, ktoré sme identifikovali, patrí najmä jeho veľká komplexita, kvôli ktorej je systém pomerne zložitý, navigácia v ňom je komplikovaná a človeku, ktorý so SAP-om nepracoval trvá veľmi dlhý čas, kým sa v ňom zorientuje a naučí sa v ňom pracovať, lebo nie je dostatočne intuitívny a používateľsky prívetivý. Ďalšou významnou nevýhodou sú vysoké náklady na jeho

implementáciu a udržiavanie a nízka personalizovateľnosť, pretože systém sa len ťažko prispôsobuje potrebám personálneho oddelenia a je menej flexibilný ako jeho konkurenti. Ďalším problémom môže byť aj závislosť od dodávateľa pri potrebe aktualizácií, alebo implementácie novej funkcionality, a v neposlednom rade zložitá integrácia s inými systémami a s tým spojené problémy s presunom dát a konzistentnosťou softvéru.

Na základe zistených nedostatkov a problémov s aktuálnym softvérovým riešením sme sa rozhodli vytvoriť návrh softvéru na podporu personálneho oddelenia, ktorý obsahuje všetky potrebné údaje a funkcionality, ktoré dané oddelenie potrebuje, je intuitívny, personalizovateľný, používateľsky prívetivý a jednoduchý na prácu v ňom. Táto diplomová práca popisuje však len prvé dve fázy tvorby softvéru, a to analýzu a návrh. Zamerali sme sa na analýzu prostredia podniku, organizačnej štruktúry, činností personálneho oddelenia, jeho fungovania, podnikových procesov a požiadaviek, ktoré sú preň dôležité pri správnom a efektívnom návrhu softvéru a vytvorili sme rôzne modely a diagramy, ktoré graficky popisujú skúmané oddelenie, jeho štruktúru a činnosti a sú nesmierne dôležité pre ďalšie fázy vývoja softvéru ako programovanie, testovanie, implementácia a udržiavanie. Správna analýza a návrh softvéru sú základom pre efektívne fungujúci systém, čo potvrdzuje aj fakt, že chyby urobené v týchto fázach tvorby softvéru si v neskorších fázach vyžadujú najdlhší čas a najväčšie náklady na ich odstránenie a je teda potrebné im venovať najvyššiu pozornosť.

ZÁVER

V tejto diplomovej práci sme sa venovali analýze a návrhu softvéru na podporu personálneho oddelenia podniku Miba Steeltec, s.r.o. Hlavným cieľom práce bolo oboznámenie sa s podnikom a jeho personálnym oddelením, pochopenie jeho fungovania, štruktúry, činností, ktoré vykonáva a ich priradenie k jednotlivým pozíciám, zozbieranie požiadaviek na softvér, získanie obrazu o aktuálne používanom softvérovom riešení na oddelení spolu s jeho výhodami a miestami, ktoré si vyžadujú zlepšenie. Na základe týchto zistení sme následne vytvorili návrh softvéru, ktorý spĺňa všetky požiadavky oddelenia spolu s návrhom databázy, v ktorej budú uchovávané potrebné dáta.

Táto práca je rozdelená na tri časti. Kapitola jeden sa týka súčasného stavu riešenej problematiky doma a v zahraničí a obsahuje charakteristiku softvéru, softvérového inžinierstva, postup vývoja softvéru a časté chyby pri jeho tvorbe, venuje sa tiež rôznym prístupom riadenia, softvérovým požiadavkám, opisuje relačné databázy, normalizáciu a s tým spojený konceptuálny, logický a fyzický model a vysvetľuje štandard UML a 4 vybrané UML diagramy – diagram tried, stavový diagram, diagram prípadov použitia a sekvenčný diagram.

Druhá kapitola opisuje cieľ práce, metodiku práce a postup, ktorý sme použili pri vytváraní tejto práce od výberu podniku, cez zozbieranie požiadaviek, výber vhodného modelovacieho nástroja, tvorbu relačnej databázy, samotný návrh systému a tvorbu diagramov, až po analýzu možných zlepšení. Druhá kapitola upriamuje pozornosť aj na metódy skúmania, opis vybraného podniku a popis softvérového produktu od spoločnosti SAP, ktorý podnik globálne používa, aj s identifikáciou jeho slabých stránok a nedostatkov, ktoré predstavujú ťažkosti pri práci personálneho oddelenia. V tejto kapitole sú uvedené aj funkčné a nefunkčné požiadavky na softvér, ktoré sme identifikovali na základe komunikácie s personálnym riaditeľom a viacerými zamestnancami oddelenia human capital.

Tretia kapitola sa zameriava na výsledky práce a diskusiu, pričom hlavnou prioritou bol samotný návrh softvéru spojený s modelovaním diagramov. Ako prvé sme v praktickej časti vytvorili organizačný diagram podniku Miba Steeltec, s.r.o. spolu s detailnejším organizačným diagramom pre jeho personálne oddelenie, ktoré spoločne zobrazujú organizačnú štruktúru podniku, jednotlivé pozície na oddelení human capital a ich hierarchiu. Nasledoval hierarchický diagram funkcií, charakterizujúci činnosti a funkcie personálneho oddelenia a ich príslušnosť k jednotlivým úsekom oddelenia. Na prepojenie týchto dvoch

diagramov sme použili relačnú maticu, vďaka ktorej je každá funkcia priradená ku konkrétnej pozícii a zobrazuje aj kto je zodpovedný za riadenie a kontrolu. Ďalším krokom v procese tvorby návrhu softvéru bolo vytvorenie relačnej databázy, normalizácia dát a ich následné zobrazenie prostredníctvom konceptuálneho, logického a fyzického modelu, pričom každý z nich poskytuje iný pohľad na dátový model a slúži inému účelu. Poslednou fázou praktickej časti tejto práce bola tvorba štyroch vybraných typov UML diagramov. Zvolili sme diagram tried, ktorý zachytáva triedy, vzťahy medzi nimi, atribúty a metódy. Druhý vybraný bol stavový diagram, ktorý opisuje všetky stavy tried, ktoré môžu nastať, spolu s prechodmi medzi jednotlivými stavmi. Ďalší v poradí bol diagram prípadov použitia, ktorý je nesmierne dôležitý, keďže zobrazuje aktérov systému a scenáre, ktoré vykonávajú a na priblíženie jednotlivých prípadov použitia sme zvolili sekvenčné diagramy, vďaka ktorým sú všetky scenáre prípadov použitia graficky znázornené.

Na modelovanie všetkých spomenutých diagramov a modelov sme použili program drawio, ktorý je dostupný vo forme aplikácie alebo aj webového rozhrania. Je to bezplatný nástroj na tvorbu diagramov, ktorý umožňuje jednoduché modelovanie procesov a rôznych iných modelov. Obsahuje rozsiahlu knižnicu symbolov a značiek, ktoré sa v diagramoch často používajú.

Výsledkom diplomovej práce bolo vytvorenie návrhu softvéru na podporu personálneho oddelenia konkrétneho podniku, ktorý spĺňa všetky jeho požiadavky, plní dôležité funkcie personálneho oddelenia, je intuitívny, efektívny a v budúcnosti je možné tento návrh ďalej rozšíriť, naprogramovať a implementovať na dané oddelenie.

ZOZNAM POUŽITEJ LITERATÚRY

[1] AGGARWAL, K. K. *Software engineering*. 2. vyd. New Delhi: New Age International, 2005. ISBN 978-8122423600.

[2] APPLE, Robert. *Conceptual, Logical and Physical Modeling--A Short Discussion*. 2007. 5 s.

[3] BASILI, Victor R. - PERRICONE, Barry T. Software errors and complexity: an empirical investigation0. In: *Communications of the ACM*. [online]. New York: Association for Computing Machinery, 1984. s. 42-52. [cit. 2024-01-12] ISSN 0001-0782. Dostupné na: https://dl.acm.org/doi/abs/10.1145/69605.2085

[4] BOEHM, Barry - BASILI, Victor R. Software defect reduction top 10 list. In: *Software engineering: Barry W. Boehm's lifetime contributions to software development, management, and research.* [online]. 2001. s. 135-137. [cit. 2024-02-27] Dostupné na: https://www.cs.umd.edu/projects/SoftEng/ESEG/papers/82.78.pdf

[5] CODD, Edgar F. Further normalization of the data base relational model. In: *Research Report.* [online]. San Jose, 1971. 33s. [cit. 2024-03-04] Dostupné na: https://forum.thethirdmanifesto.com/wp-content/uploads/asgarosforum/987737/00-efc-further-normalization.pdf

[6] DEMBA, Moussa. Algorithm for relational database normalization up to 3NF. In: *International Journal of Database Management Systems*. [online]. 2013. s. 39-51. [cit. 2024-03-21] ISSN 0975-5705. Dostupné na: https://www.researchgate.net/publication/269674039_Algorithm_for_Relational_Database_Normalization_Up_to_3NF

[7] DRANIDIS, Dimitris. Evaluation of studentuml: an educational tool for consistent modelling with uml. In: *Proceedings of the Informatics Education Europe II Conference*. [online]. 2007. s. 248-256. [cit. 2024-03-10] Dostupné na: https://www.researchgate.net/profile/Dimitris-Dranidis/publication/233779315_Evaluation_of_StudentUML_an_Educational_Tool_for_Consistent_Modelling_with_UML/links/56767e7d08aebcdda0e85a15/Evaluation-of-StudentUML-an-Educational-Tool-for-Consistent-Modelling-with-UML.pdf

[8] ĎURAČIOVÁ, Renata - CIBULKA, Dušan. *DATABÁZOVÉ SYSTÉMY V GIS Návody na cvičenia*. Bratislava: Nakladateľstvo STU, 2015. 135 s. ISBN 978-80-227-4502-4.

[9] ELBAHRI, F. M. - AL-SANJARY, O. I. - ALI, M. A. - NAIF, Z. A. - IBRAHIM, O. A. - Mohammed, M. N. Difference comparison of SAP, Oracle, and Microsoft solutions based on cloud ERP systems: A review. In: *IEEE 15th International Colloquium on Signal Processing & Its Applications (CSPA)*. 2019. s. 65-70. ISSN 978-1-5386-7563-2.

[10] FLEISCHMANN, Albert - SCHMIDT, Werner - STARY, Christian. Requirements Specification as Executable Software Design-A Behavior Perspective. In: *REFSQ Workshops*. [online]. 2015. s. 9-18. [cit. 2024-03-19]. Dostupné na: https://ceur-ws.org/Vol-1342/01-CRE.pdf

[11] FRANĚK, Zdeněk. *Ojektové metody modelování v příkladech: Distanční studijní text*. Opava: Slezská univerzita v Opavě, 2018. 129 s. ISBN 978-80-7510-295-9.

[12] FRITZ, Bauer a kol. Software Engineering. A report on a conference sponsored by NATO Science Committee. In: *NATO*. 1968, 1968.

[13] HODA, Rashina - NOBLE, James - MARSHALL, Stuart. Self-organizing roles on agile software development teams. In: *IEEE Transactions on Software Engineering*. 2012. ISSN 422-444.

[14] CHREN, Stanislav a kol. Mistakes in UML diagrams: analysis of student projects in a software engineering course. In: *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)*. IEEE, 2019. s. 100-109.

[15] IBM. *Extend relationships*. [online]. 2023. [cit. 2024-04-10]. Dostupné na: https://www.ibm.com/docs/en/dma?topic=diagrams-extend-relationships

[16] IBM. *Include relationships*. [online]. 2023. [cit. 2024-04-10]. Dostupné na: https://www.ibm.com/docs/en/dma?topic=diagrams-include-relationships

[17] IBM. *What is software development*. [online]. 2017. [cit. 2024-01-16]. Dostupné na: https://www.ibm.com/topics/software-development

[18] INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. *Systems and software engineering — Vocabulary*. In: *iso.org* [online]. [cit. 2024-01-17]. Dostupné na: https://www.iso.org/obp/ui/#iso:std:iso-iec-ieee:24765:ed-2:v1:en

[19] JACOBSON, Ivar - BOOCH, Grady - RUMBAUGH, James. *The unified modeling language reference manual*. 2.vyd. 2005. 568 s. ISBN 978-0321718952.

[20] JURÍK, Pavol. *Informačné systémy v podnikovej praxi*. 2. vyd. Nové Zámky: Tlačiareň MERKUR, s.r.o., 2018. 186 s. ISBN 978-80-970233-7-9.

[21] JURÍK, Pavol. *Servisne orientovaná architektúra v procesne riadenom podniku*. 1. vyd. Nové Zámky: Tlačiareň MERKUR, s.r.o., 2020. 178 s. ISBN 978-80-89996-06-3.

[22] KOÇ, H. - ERDOĞAN, A. - BARJAKLY, Y. - PEKER, S. UML diagrams in software engineering research: a systematic literature review. In: *Proceedings*. MDPI, 2021. 8 s. [cit. 2024-01-19]. Dostupné na: https://www.mdpi.com/2504-3900/74/1/13

[23] KREMEŇOVÁ, Iveta - SÚLOVEC, Roman. *Uplatnenie princípov objektového modelovania pri budovaní informačných systémov*. 2020. 6 s.

[24] KURNIAWAN, Tri Astoto - LAM-SON, LÃª - PRIYAMBADHA, Bayu. Challenges in developing sequence diagrams (UML). In: *Journal of Information Technology and Computer Science*. 2020. 14 s. ISSN 221-234.

[25] MANAGEMENTMANIA. *Software*. [online]. 2015. [cit. 2024-01-17]. Dostupné na: https://managementmania.com/sk/software

[26] RODRIGUEZ-MARTINEZ, Laura C. - MORA, Manuel - FRANCISCO, J. Alvarez. A Descriptive/Comparative Study of the Evolution of Process Models of Software Development Life Cycles. In: *Proceedings of the 2009 Mexican International Conference on Computer Science IEEE Computer Society*. Washington, DC, USA. 2009.

[27] MICROSOFT. *Jednoduchá príručka tvorby diagramov a modelovania databáz UML*. [online]. 2019. [cit. 2024-01-20]. Dostupné na: https://www.microsoft.com/sk-sk/microsoft-365/business-insights-ideas/resources/guide-to-uml-diagramming-and-database-modeling

[28] MICROSOFT. *Vytvorenie diagramu s zápisom databázy Chen*. [online]. 2021. [cit. 2024-04-15]. Dostupné na: https://support.microsoft.com/sk-sk/topic/vytvorenie-diagramu-s-z%C3%A1pisom-datab%C3%A1zy-chen-75d28eff-2509-4faf-8cd9-3eda5fb4327b

[29] MISHRA, Apoorva - DUBEY, Deepty. A comparative study of different software development life cycle models in different scenarios. In: *International Journal of Advance research in computer science and management studies*. [Online]. 2013. [cit. 2024-04-15]. ISSN: 2321-7782. Dostupné na: https://www.researchgate.net/publication/289526047_A_Comparative_Study_of_Different_Software_Development_Life_Cycle_Models_in_Different_Scenarios

[30] NAGY, Roman. Procesné modely testovania softvéru. In: *AT&P Journal INFORMATIKA 12 2003*. [online]. 2003. s. 17-19. [cit. 2024-02-23] ISSN 1336-233X. Dostupné na: https://www.atpjournal.sk/buxus/docs/atp-2003-12-17_19.pdf

[31] OBJECT MANAGEMENT GROUP a kol. OMG. *Unified Modeling Language (OMG UML) Version 2.5. 1*, 2017.

[32] OSTERWEIL, Leon J. What is software? The Role of Empirical Methods in Answering the Question. In: *The Essence of Software Engineering*. 2018. s. 59-76.

[33] OZKAYA, Mert. Are the UML modelling tools powerful enough for practitioners? A literature review. In: *IET Software*, 2019. [online]. [cit. 2024-02-12]. ISSN 338-354. Dostupné na: https://ietresearch.onlinelibrary.wiley.com/doi/full/10.1049/iet-sen.2018.5409

[34] RICHTA, Karel. Novinky ve standardu UML 2.0. In: *Moderní databáze 2005*. [online]. Praha, 2005. [cit. 2024-01-21]. Dostupné na: https://adoc.pub/novinky-ve-standardu-uml-20.html

[35] RICHTA, Karel. Unifikovaný modelovací jazyk UML. In: *Systems Integration*. 2003. 386 s.

[36] SAP. *Produkty spoločnosti SAP*. [online]. 2020. [cit. 2024-04-15]. Dostupné na: https://www.sap.com/sk/products.html

[37] SAP. *Understanding the User Interface*. [online]. 2016. [cit. 2024-04-15]. Dostupné na: *https://help.sap.com/doc/saphelp_nw75/7.5.5/enUS/87/bd5c5b6ab34a4f93c134a5fa153db1/content.htm?no_cache=true*

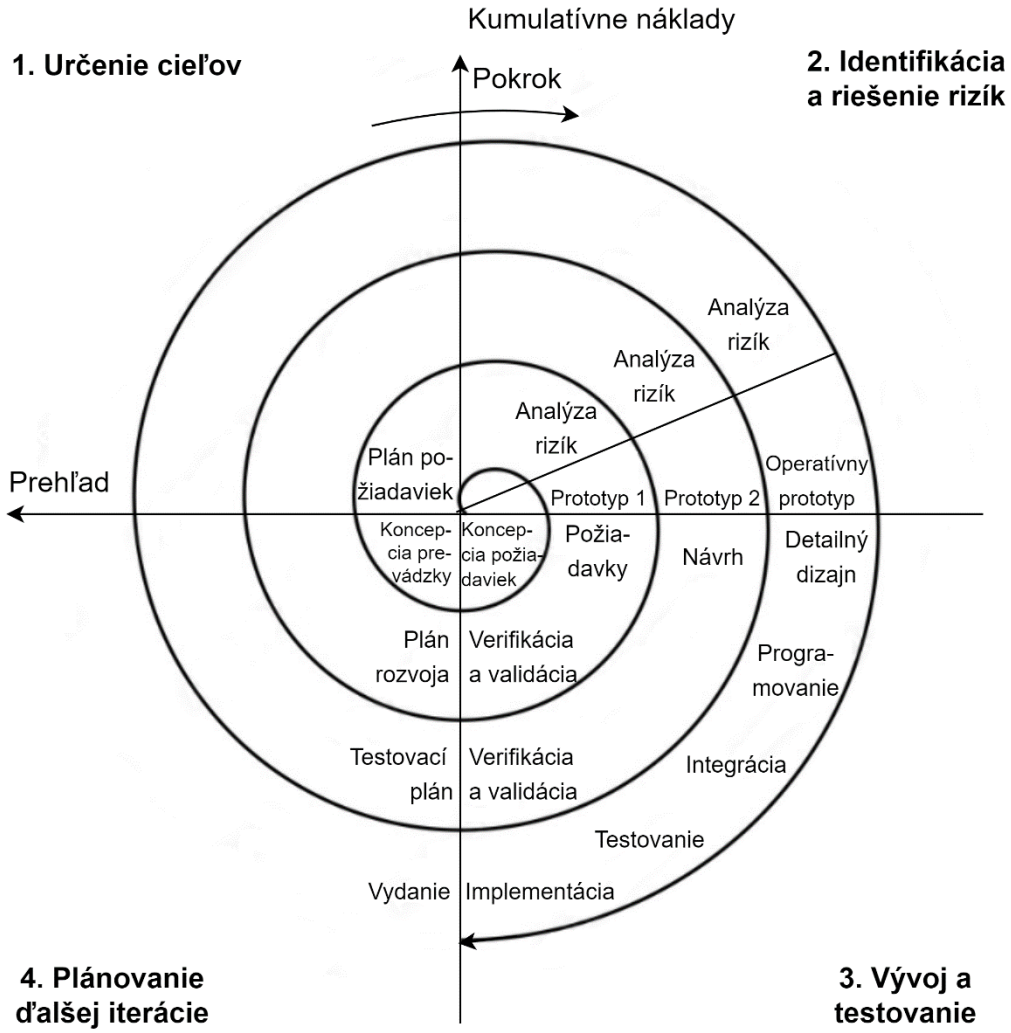
[38] SCHACH, Stephen R. *Object-oriented software engineering*. Aksen associates, 1990. 499 s. ISBN 978-0-256-08515-0.

[39] SOMMERVILLE, Ian. *Software engineering 9th Edition*. [online]. 2011, 790 s. [cit. 2024-01-19]. ISBN: 0-13-703515-2. Dostupné na: https://engineering.futureuniversity.com/BOOKS%20FOR%20IT/Software-Engineering-9th-Edition-by-Ian-Sommerville.pdf

[40] STATISTA: *Most used programming languages among developers worldwide as of 2023.* [online]. 2023. [cit. 2024-01-17]. Dostupné na: https://www.statista.com/statistics/793628/worldwide-developer-survey-most-used-languages/?fbclid=IwAR24Q0qn-updV2cRhdLUPzNYQVkmc28vPV727LahNW3W85nZDtbDJtlVTIE

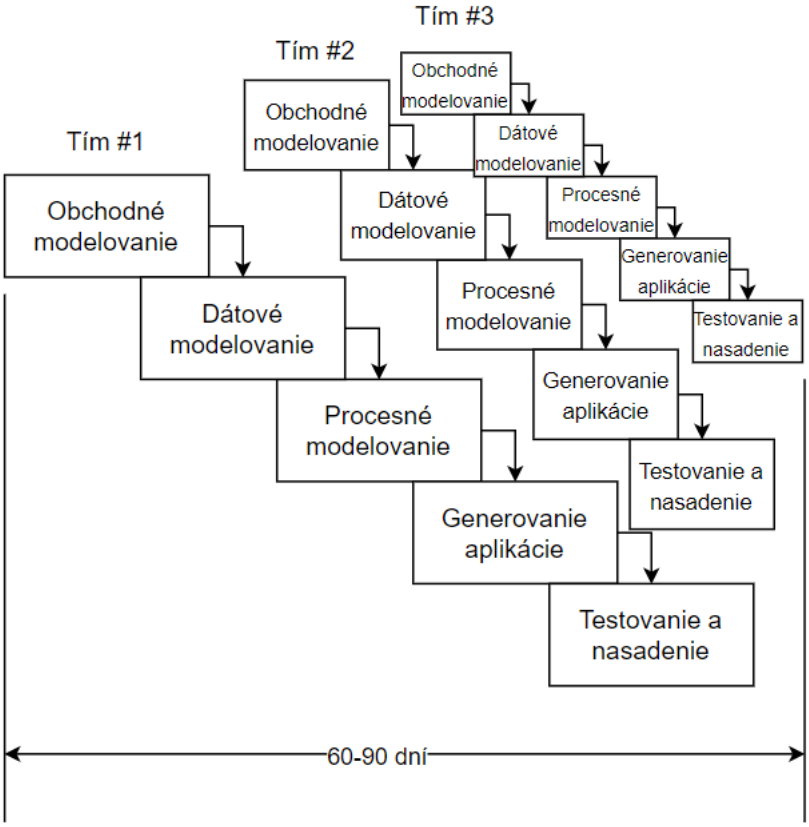
ZOZNAM PRÍLOH

Príloha č.1: Zväčšená verzia obrázka 5



Obrázok 5: Špirálový model SDLC (vlastné spracovanie podľa [29])

Príloha č.2: Zväčšená verzia obrázka 6



***Obrázok 6: RAD model SDLC** (vlastné spracovanie podľa [29])*

Príloha č.3: SQL kód na tvorbu databázy v phpMyAdmin

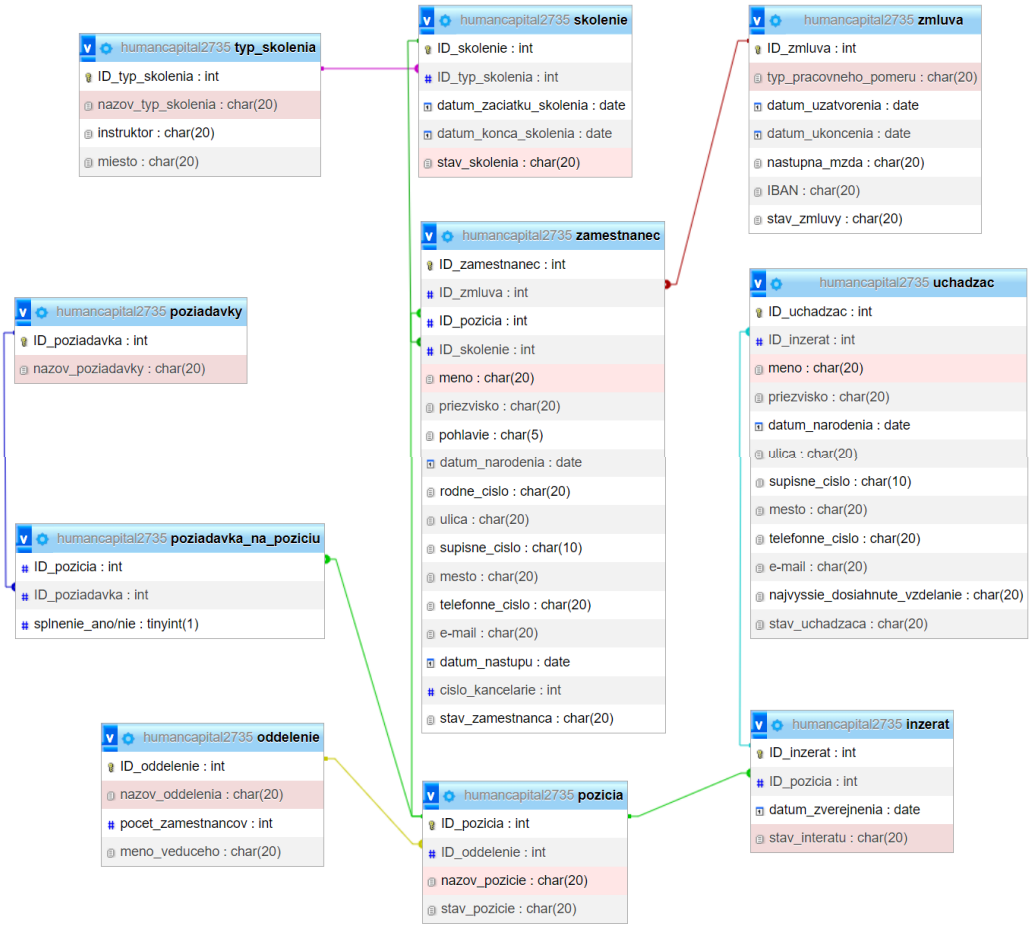
```
CREATE TABLE IF NOT EXISTS
`zamestnanec` (
  `ID_zamestnanec` int(5) NOT NULL,
  `ID_zmluva` int(5) NOT NULL,
  `ID_pozicia` int(5) NOT NULL,
  `ID_skolenie` int(5) NOT NULL,
  `meno` char(20) NOT NULL,
  `priezvisko` char(20) NOT NULL,
  `pohlavie` char(5) NOT NULL,
  `datum_narodenia` date NOT NULL,
  `rodne_cislo` char(20) NOT NULL,
  `ulica` char(20) NOT NULL,
  `supisne_cislo` char(10) NOT NULL,
  `mesto` char(20) NOT NULL,
  `telefonne_cislo` char(20) NOT NULL,
  `e-mail` char(20) NOT NULL,
  `datum_nastupu` date NOT NULL,
  `cislo_kancelarie` int(5) NOT NULL,
  `stav_zamestnanca` char(20) NOT
NULL,
  PRIMARY KEY (`ID_zamestnanec`)
) ENGINE=InnoDB DEFAULT
CHARSET=latin1;

CREATE TABLE IF NOT EXISTS
`zmluva` (
`ID_zmluva` int(5) NOT NULL,
`typ_pracovneho_pomeru` char(20) NOT
NULL,
`datum_uzatvorenia` date NOT NULL,
`datum_ukoncenia` date NOT NULL,
`nastupna_mzda` char(20)  NOT NULL,
`IBAN` char(20)  NOT NULL,
`stav_zmluvy ` char(20) NOT NULL,

  PRIMARY KEY (`ID_zmluva`)
) ENGINE=InnoDB DEFAULT
CHARSET=latin1;

ALTER TABLE zamestnanec
ADD FOREIGN KEY (ID_zmluva)
REFERENCES zmluva(ID_zmluva);

CREATE TABLE IF NOT EXISTS
`skolenie` (
  `ID_skolenie` int(5) NOT NULL,
  `ID_typ_skolenia` int(5) NOT NULL,
`datum_zaciatku_skolenia` date NOT
NULL,
`datum_konca_skolenia` date NOT
NULL,
  `stav_skolenia` char(20) NOT NULL,
  PRIMARY KEY (`ID_skolenie`)
) ENGINE=InnoDB DEFAULT
CHARSET=latin1;

ALTER TABLE zamestnanec
ADD FOREIGN KEY (ID_skolenie)
REFERENCES skolenie(ID_skolenie);

CREATE TABLE IF NOT EXISTS
`typ_skolenia` (
  `ID_typ_skolenia` int(5) NOT NULL,
```

```
  `nazov_typ_skolenia` char(20) NOT
NULL,
  `instruktor` char(20) NOT NULL,
  `miesto` char(20) NOT NULL,
  PRIMARY KEY (`ID_typ_skolenia`)
) ENGINE=InnoDB DEFAULT
CHARSET=latin1;

ALTER TABLE skolenie
ADD FOREIGN KEY (ID_typ_skolenia)
REFERENCES typ_skolenia
(ID_typ_skolenia);

CREATE TABLE IF NOT EXISTS
`pozicia` (
  `ID_pozicia` int(5) NOT NULL,
  `ID_oddelenie` int(5) NOT NULL,
  `nazov_pozicie` char(20) NOT NULL,
  `stav_pozicie` char(20) NOT NULL,
  PRIMARY KEY (`ID_pozicia`)
) ENGINE=InnoDB DEFAULT
CHARSET=latin1;

ALTER TABLE zamestnanec
ADD FOREIGN KEY (ID_pozicia)
REFERENCES pozicia(ID_pozicia);

CREATE TABLE IF NOT EXISTS
`oddelenie` (
  `ID_oddelenie` int(5) NOT NULL,
  `nazov_oddelenia` char(20) NOT
NULL,
  `pocet_zamestnancov` int(5) NOT
NULL,
  `meno_veduceho` char(20) NOT NULL,
  PRIMARY KEY (`ID_oddelenie`)
) ENGINE=InnoDB DEFAULT
CHARSET=latin1;

ALTER TABLE pozicia
ADD FOREIGN KEY (ID_oddelenie)
REFERENCES oddelenie
(ID_oddelenie);

CREATE TABLE IF NOT EXISTS
`poziadavka_na_poziciu` (
  `ID_pozicia` int(5) NOT NULL,
  `ID_poziadavka` int(5) NOT NULL,
  `splnenie_ano/nie` boolean NOT NULL
) ENGINE=InnoDB DEFAULT
CHARSET=latin1;

ALTER TABLE poziadavka_na_poziciu
ADD FOREIGN KEY (ID_pozicia)
REFERENCES pozicia (ID_pozicia);

CREATE TABLE IF NOT EXISTS
`poziadavky` (
  `ID_poziadavka` int(5) NOT NULL,
  `nazov_poziadavky` char(20) NOT
NULL,
  PRIMARY KEY (`ID_poziadavka`)
) ENGINE=InnoDB DEFAULT
CHARSET=latin1;

ALTER TABLE poziadavka_na_poziciu
```

```
ADD FOREIGN KEY (ID_poziadavka)
REFERENCES poziadavky
(ID_poziadavka);

CREATE TABLE IF NOT EXISTS
`inzerat` (
  `ID_inzerat` int(5) NOT NULL,
  `ID_pozicia` int(5) NOT NULL,
  `datum_zverejnenia` date NOT NULL,
  `stav_interatu` char(20) NOT NULL,
  PRIMARY KEY (`ID_inzerat`)
) ENGINE=InnoDB DEFAULT
CHARSET=latin1;

ALTER TABLE inzerat
ADD FOREIGN KEY (ID_pozicia)
REFERENCES pozicia (ID_pozicia);

CREATE TABLE IF NOT EXISTS
`uchadzac` (
  `ID_uchadzac` int(5) NOT NULL,
  `ID_inzerat` int(5) NOT NULL,
  `meno` char(20) NOT NULL,
  `priezvisko` char(20) NOT NULL,
  `datum_narodenia` date NOT NULL,
  `ulica` char(20) NOT NULL,
  `supisne_cislo` char(10) NOT NULL,
  `mesto` char(20) NOT NULL,
  `telefonne_cislo` char(20) NOT NULL,
  `e-mail` char(20) NOT NULL,
  `najvyssie_dosiahnute_vzdelanie`
char(20) NOT NULL,
  `stav_uchadzaca` char(20) NOT NULL,
  PRIMARY KEY (`ID_uchadzac`)
) ENGINE=InnoDB DEFAULT
CHARSET=latin1;

ALTER TABLE uchadzac
ADD FOREIGN KEY (ID_inzerat)
REFERENCES inzerat (ID_inzerat);
```

Zdroj: vlastné spracovanie

Príloha č.4: Zväčšená verzia obrázka 24



***Obrázok 24: Fyzický model pre personálne oddelenie Miba Steeltec, s.r.o. - phpMyAdmin** (vlastné spracovanie)*